Reinforcement Learning for Active Visual Perception

Reinforcement Learning for Active Visual Perception

by Aleksis Pirinen

Thesis for the degree of Doctor of Philosophy
Thesis advisors: Prof. Cristian Sminchisescu, Dr. Carl Olsson
Faculty opponent:
Prof. Hedvig Kjellström, KTH Royal Institute of Technology, Stockholm

To be presented, with the permission of the Faculty of Engineering at Lund University, for public criticism in lecture hall MH:Hörmander at the Centre for Mathematical Sciences, Sölvegatan 18, Lund, on Thursday, the 10th of June 2021 at 13:15.

Organization **LUND UNIVERSITY** Centre for Mathematical Sciences Box 118 SE–221 00 LUND Sweden	Document name DOCTORATE THESIS IN MATHEMATICAL SCIENCES Date of disputation 2021-06-10
Author(s) Aleksis Pirinen	Sponsoring organization

Title and subtitle
Reinforcement Learning for Active Visual Perception

Abstract

Visual perception refers to automatically recognizing, detecting, or otherwise sensing the content of an image, video or scene. The most common contemporary approach to tackle a visual perception task is by training a deep neural network on a pre-existing dataset which provides examples of task success and failure, respectively. Despite remarkable recent progress across a wide range of vision tasks, many standard methodologies are static in that they lack mechanisms for adapting to any particular settings or constraints of the task at hand. The ability to adapt is desirable in many practical scenarios, since the operating regime often differs from the training setup. For example, a robot which has learnt to recognize a static set of training images may perform poorly in real-world settings, where it may view objects from unusual angles or explore poorly illuminated environments. The robot should then ideally be able to actively position itself to observe the scene from viewpoints where it is more confident, or refine its perception with only a limited amount of training data for its present operating conditions.

In this thesis we demonstrate how reinforcement learning (RL) can be integrated with three fundamental visual perception tasks – object detection, human pose estimation, and semantic segmentation – in order to make the resulting pipelines more adaptive, accurate and/or faster. In the first part we provide object detectors with the capacity to actively select what parts of a given image to analyze and when to terminate the detection process. Several ideas are proposed and empirically evaluated, such as explicitly including the speed-accuracy trade-off in the training process, which makes it possible to specify this trade-off during inference. In the second part we consider active multi-view 3d human pose estimation in complex scenarios with multiple people. We explore this in two different contexts: i) active triangulation, which requires carefully observing each body joint from multiple viewpoints, and ii) active viewpoint selection for monocular 3d estimators, which requires considering which viewpoints yield accurate fused estimates when combined. In both settings the viewpoint selection systems face several challenges, such as partial observability resulting e.g. from occlusions. We show that RL-based methods outperform heuristic ones in accuracy, with negligible computational overhead. Finally, the thesis concludes with establishing a framework for embodied visual active learning in the context of semantic segmentation, where an agent should explore a 3d environment and actively query annotations to refine its visual perception. Our empirical results suggest that reinforcement learning can be successfully applied within this framework as well.

Key words
computer vision, reinforcement learning, deep learning, active vision, object detection, human pose estimation, semantic segmentation

Classification system and/or index terms (if any)		
Supplementary bibliographical information		Language English
ISSN and key title 1404-0034		ISBN 978-91-7895-795-8 (print) 978-91-7895-796-5 (pdf)
Recipient's notes	Number of pages 219	Price
	Security classification	

Reinforcement Learning for Active Visual Perception

by Aleksis Pirinen

LUND
UNIVERSITY

Funding information: The thesis work was financially supported by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535 and PCCF-2016-0180, and the EU Horizon 2020 Grant DE-ENIGMA.

Faculty of Engineering, Centre for Mathematical Sciences
Doctoral Thesis in Mathematical Sciences 2021:2
LUTFMA-1071-2021
ISBN: 978-91-7895-795-8 (print)
ISBN: 978-91-7895-796-5 (pdf)
ISSN: 1404-0034

Printed in Sweden by Media-Tryck, Lund University, Lund 2021

403.96 | 421.21 | 350.00

The first value is the average of the monthly measurements of atmospheric carbon dioxide concentration (in ppm) at Mauna Loa in May 2015, the year in which I began my doctoral studies.

The second value is the atmospheric carbon dioxide concentration (in ppm) measured at Mauna Loa on the third of April 2021, the month in which I submitted this thesis for printing.

The third value is the upper limit of atmospheric carbon dioxide concentration (in ppm) considered 'safe'.
It was breached in 1988.

Abstract

Visual perception refers to automatically recognizing, detecting, or otherwise sensing the content of an image, video or scene. The most common contemporary approach to tackle a visual perception task is by training a deep neural network on a pre-existing dataset which provides examples of task success and failure, respectively. Despite remarkable recent progress across a wide range of vision tasks, many standard methodologies are static in that they lack mechanisms for adapting to any particular settings or constraints of the task at hand. The ability to adapt is desirable in many practical scenarios, since the operating regime often differs from the training setup. For example, a robot which has learnt to recognize a static set of training images may perform poorly in real-world settings, where it may view objects from unusual angles or explore poorly illuminated environments. The robot should then ideally be able to actively position itself to observe the scene from viewpoints where it is more confident, or refine its perception with only a limited amount of training data for its present operating conditions.

In this thesis we demonstrate how reinforcement learning (RL) can be integrated with three fundamental visual perception tasks – object detection, human pose estimation, and semantic segmentation – in order to make the resulting pipelines more adaptive, accurate and/or faster. In the first part we provide object detectors with the capacity to actively select what parts of a given image to analyze and when to terminate the detection process. Several ideas are proposed and empirically evaluated, such as explicitly including the speed-accuracy trade-off in the training process, which makes it possible to specify this trade-off during inference. In the second part we consider active multi-view 3d human pose estimation in complex scenarios with multiple people. We explore this in two different contexts: i) active triangulation, which requires carefully observing each body joint from multiple viewpoints, and ii) active viewpoint selection for monocular 3d estimators, which requires considering which viewpoints yield accurate fused estimates when combined. In both settings the viewpoint selection systems face several challenges, such as partial observability resulting e.g. from occlusions. We show that RL-based methods outperform heuristic ones in accuracy, with negligible computational overhead. Finally, the thesis concludes with establishing a framework for embodied visual active learning in the context of semantic segmentation, where an agent should explore a 3d environment and actively query annotations to refine its visual perception. Our empirical results suggest that reinforcement learning can be successfully applied within this framework as well.

Popular Summary

Visually unimpaired people often take the ability to perceive the world with their eyes for granted. When visiting a house for the first time we can recognize the size and layout of the hallway, where to put our shoes, where pets and other people in the room are, and so on. We can also make predictions and decisions based on the visual input we receive. A core challenge of computer vision and artificial intelligence is to develop machines capable of the same thing – perceiving the world around them and acting rationally based on what they observe. In this thesis we study different types of artificial visual perception systems, which can be used for example to automatically detect objects in images or understand human poses and motion in videos.

Today's visual perception systems are typically powered by so called deep neural networks, which are inspired by the human brain with its neurons and complex web of connections. While deep networks yield remarkable results in many applications, they are often computationally expensive and time-consuming to use. This can be especially problematic in real-time scenarios such as video surveillance, or in robotics where an agent may have to quickly explore a large and unknown environment. Also, to make a deep network function properly it is first trained on large amounts of data, typically annotated images. Annotation is a tedious process that costs time and money, as it involves humans describing what the data contains, for example by drawing object boundaries in images. Finally, even when a perception system has been trained it may work poorly in circumstances that differ from the training data. For example, if the perception system is mostly trained on images that depict objects from the front it may fail to recognize them from the side.

In this thesis we study and develop *active* methods for visual perception. By focusing a pre-trained perception model on the most relevant aspects of a scene or an image, computational costs can be reduced and/or conditions where the model is inaccurate can be avoided. We also show how similar ideas can be applied when training perception systems, which reduces the effort associated with data annotation. The active visual perception methods we develop are based on reinforcement learning, a trial-and-error approach for discovering desirable behaviour by means of a reward function. For illustration, consider a self-driving car that should drive from a start location to a given destination within a specific time limit. In practice there may exist several paths between the two locations, such as when driving

in a large city. A simple[1] reward function for this task is the negative distance between the destination and the location of the car when the time is over. This implies that the maximum reward is obtained when the car reaches its goal on time. Note that the reward does not specify *how* the car should drive, only *what* its objective is. Thus the car has to try many different strategies to figure out what works and what does not. Reinforcement learning is suitable in scenarios like this, where an agent may have to perform several actions until it knows whether or not it has succeeded.

This thesis explores active visual perception in three different settings. In the first two we propose methods that actively select what parts of a given input or set of inputs to analyze (from which viewpoints to observe a scene, and where to look in an image, respectively) so that a pre-trained perception system performs well, and/or to reduce the amount of computation that is required. In the third setup we develop and study agents which are tasked to refine a given perception model by actively exploring a given scene, such as a floor plan of a house. As these agents move around the scene they are allowed to ask for annotations (training data), which are then used to refine their perception models. The crux is that the agents are allowed to request only a limited amount of training data, so they should be careful regarding which data they select for training. We show in each setting that active visual perception methods trained with reinforcement learning match or outperform alternative approaches, typically at the same or lower computational costs.

[1]The author of this thesis recommends providing also a negative reward for collisions.

Populärvetenskaplig sammanfattning

De av oss som inte lider av någon synskada tar ofta förmågan att se vår omgivning för given. När vi besöker ett hus för första gången kan vi med synen uppfatta hallens storlek och utformning, var vi kan ställa våra skor, var i hallen som husdjur och andra människor befinner sig, och så vidare. Baserat på vad vi ser kan vi dessutom förutsäga saker och ta relevanta beslut. En av de stora utmaningarna och förhoppningarna inom datorseende och artificiell intelligens är att utveckla maskiner som kan göra samma sak – att se världen omkring dem och agera rationellt baserat på vad de ser. Förmågan att se kallas ofta i mer tekniska sammanhang för visuell perception. I denna avhandling studerar vi olika typer av system för artificiell visuell perception, vilka kan användas exempelvis till att automatiskt detektera objekt i bilder eller förstå människors hållningar (poser) och rörelser i videor.

Dagens visuella perceptionssystem drivs oftast av så kallade djupa neuronnät, vilka är inspirerade av den mänskliga hjärnan med sina neuroner och neuronsammankopplingar. Djupa neuronnät ger idag utmärkta resultat i många tillämpningar, men de är ofta beräkningsmässigt dyra och tidskrävande att använda. Detta kan bli särskilt problematiskt i sammanhang som kräver effektiv bearbetning av data (exempelvis videoövervakning), eller inom robotik där en agent snabbt kan behöva utforska en stor och okänd omgivning. För att få ett djupt neuronnät att fungera som det ska behöver det dessutom tränas på stora mängder data, vanligen annoterade bilder. Annotering är en mödosam process som kostar både tid och pengar, eftersom den involverar människor som beskriver vad bilderna föreställer, exempelvis genom rita konturer kring olika objekt för att markera var i bilden de är och vilken form de har. Ett ytterligare problem är att när ett perceptionssystem väl har tränats kan det fungera betydligt sämre om det används under omständigheter som skiljer sig från träningsdatan. Exempelvis kan ett perceptionssystem som mestadels tränats på bilder av objekt framifrån misslyckas att känna igen dem från sidan.

I denna avhandling studerar och utvecklar vi *aktiva* metoder för visuell perception. Genom att fokusera en redan tränad perceptionsmodell på de mest relevanta aspekterna av en scen eller bild kan man minska mängden beräkningar som behöver göras och/eller undvika omständigheter där modellen ger opålitliga resultat. Vi visar även hur liknande idéer kan appliceras när man tränar ett perceptionssystem, vilket kan reducera mängden dataannotering som krävs. Våra aktiva visuella perceptionsmodeller baseras på förstärkningsinlärning, ett slags prova-och-se-metod för att upptäcka önskvärt beteende baserat på en given belönings-

signal. För att illustrera detta koncept kan man föreställa sig exempelvis en självkörande bil vars uppgift är att köra från en startposition till en given målposition inom en viss tidsram. I praktiken kan det finnas flera vägar mellan de två platserna, till exempel om bilen navigerar i en större stad. En enkel[2] belöningssignal för denna uppgift är den negativa distansen mellan målpositionen och bilens position när tiden är över. Detta innebär att den maximala belöningen erhålls när bilen når sitt mål i tid. Notera att belöningssignalen inte specificerar *hur* bilen bör köra, bara *vad* dess ultimata uppgift är (i det här fallet att nå målpositionen inom en viss tid). Således måste bilen utforska flera olika strategier för att lista ut vad som fungerar och vad som inte gör det. Förstärkningsinlärning lämpar sig väl i den här typen av situationer, det vill säga när en agent behöver utföra flera olika handlingar innan den vet om den lyckats eller inte.

Denna avhandling utforskar aktiv visuell perception i tre olika kontexter. I de första två utvecklar vi metoder som aktivt väljer vilka delar av en insignal eller uppsättning insignaler som ska analyseras (från vilka vyer en scen ska betraktas, respektive var man ska titta i en given bild) för att ett på förhand tränat perceptionssystem ska ge mer pålitliga resultat och/eller för att minska mängden utförda bräkningar. I den tredje kontexten utvecklar och studerar vi agenter vars uppgift är att förbättra en given perceptionsmodell genom att aktivt gå runt och utforska en scen, såsom ett våningsplan i ett hus. Under tiden som agenterna utforskar scenen har de också möjlighet att be om annoteringar (träningsdata), vilka sedan används till att förbättra deras perceptionsmodeller. Kruxet är att agenterna bara tillåts be om en begränsad mängd träningsdata, så de bör vara selektiva angående vilken träningsdata de väljer. Vi visar i samtliga kontexter att aktiva visuella perceptionsmetoder som tränats med förstärkningsinlärning matchar eller förbättrar alternativa metoder, och dessutom oftast till samma eller lägre beräkningsmässiga kostnader.

[2]Författaren till denna avhandling rekommenderar att man även ger en negativ belöning för kollisioner.

List of Publications

This thesis is based on the following publications:

- S. Mathe, **A. Pirinen**, C. Sminchisescu, "Reinforcement Learning for Visual Object Detection", *Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, 2016.
- **A. Pirinen**, C. Sminchisescu, "Deep Reinforcement Learning of Region Proposal Networks for Object Detection", *Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, USA, 2018.
- **A. Pirinen**,[1] E. Gärtner,[1] C. Sminchisescu, "Domes to Drones: Self-Supervised Active Triangulation for 3D Human Pose Reconstruction", *Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, 2019.
- E. Gärtner,[1] **A. Pirinen**,[1] C. Sminchisescu, "Deep Reinforcement Learning for Active Human Pose Estimation", *Association for the Advancement of Artificial Intelligence (AAAI)*, New York, USA, 2020.
- D. Nilsson, **A. Pirinen**, E. Gärtner, C. Sminchisescu, "Embodied Visual Active Learning for Semantic Segmentation", *Association for the Advancement of Artificial Intelligence (AAAI)*, Virtual conference, 2021.

The author also contributed to the following publications:

- **A. Pirinen**, B. Ames, "Exact Clustering of Weighted Graphs via Semidefinite Programming", *Journal of Machine Learning Research (JMLR)*, 2019.
- M. Priisalu, C. Paduraru, **A. Pirinen**, C. Sminchisescu, "Semantic Synthesis of Pedestrian Locomotion", *Asian Conference on Computer Vision (ACCV)*, Virtual conference, 2020.

All papers are reproduced with permission of their respective publishers.

[1]Equal contribution.

Acknowledgements

It has been nearly six years since I began as a PhD student. Those six years could have been five, had I not been blessed with my daughter Juni towards the end of my doctoral studies. Thank you, Juni, for being my main daytime companion (sorry Yoshi!) during parental leave in the early days of the pandemic. Thank you also for giving me a research idea one of those nights when we were both awake – this idea later made its way into one of the papers of this thesis. Most of all however, I want to thank you for all those times you have made me *not* think about research because I've been too busy observing you exploring the world.

There are many more people without which this thesis would not have been what it is. I would like to thank my supervisor Cristian Sminchisescu for encouraging me to pursue this interesting line of research. Thank you for all valuable input and feedback and for all the ideas you have suggested to and investigated with me during these years. I'm also grateful for having been given opportunities to visit conferences and other places where the latest research is discussed.

I want to further thank the other people in Cristian's group in Lund. David, for being my steady office partner over the last years and for collaborating on interesting research. Maria, for being my fellow traveller and office mate that summer in California, for discussions about linguistic and cultural differences between Estonia and Finland, and for rewarding research discussions. Erik, for the endless amount of internal laughs around deadlines (I'm pretty sure you're reading this acknowledgements page after NIPS). Martin, for open-hearted existential conversations as well as research-related discussions, and for being a terrific guide during our train trip to Zurich. Henning, for many discussions about the future and GANs. Ted, for being a great project supervisor in that spatial statistics course I once took, and – last but certainly not least – for putting Veberöd on the map. I'm not sure the world needs another superhero movie, but the next one should definitely star *Attention Aleksis*, *Deep Dave*, *Generative Gärtner*, *Hyper Henning*, *Multi-Modal Maria*, *Multi-Modal Martin* and *Time-Invariant Ted*.

A big thank you also to the rest of my colleagues at the Centre for Mathematical Sciences. Particular shout-outs go to Carl Olsson for being my co-supervisor, Carl-Gustav Werner for leaving a late Friday night dinner to assist with reluctant computers a few hours before a deadline, Lena Lööf for consistently excellent practical advice, Patrik Persson for his drone control mastery, and Tomas Persson for bröling at Stortorget. Looking abroad, I am also grateful for the valuable input and support provided by Stefan Mathe, Alin Popa, Andrei Zanfir, Mihai Zanfir and Elisabeta Oneata.

I would next like to thank my parents for encouraging me to work hard while at the same time not pressuring me into a pre-specified direction. I always appreciated that freedom and will strive to give Juni the same feeling as she grows up. To my sister Karolina, thank you for being there for me throughout my childhood (and to Magnus, thank you for being there for my sister today). I would also like to mention Erik, Monty, Robin, Marielle and Angelos, close friends who in their respective ways have contributed to bringing this thesis to fruition. I'm further grateful for and happy to be part of my dear 'Skåne family' – cheers, Annicka, Gert, Jonna, Emmie and Rikard! Finally, a big, warm and loving thank you to you, Jonna, for all your love and support, for listening deeply and advising wisely, and for being kind-hearted to friend and stranger alike. I look forward to many more adventures with our little quartet.

Översättning av första och sista stycket:

Det har gått nästan sex år sedan jag började som doktorand. Dessa sex år kunde ha varit fem, om det inte var för att min dotter Juni gjorde entré mot slutet av mina studier. Tack, Juni, för att du var min närmaste dagtidskompanjon (ledsen Yoshi!) under min föräldraledighet i början av pandemin. Jag vill även tacka dig för att du gav mig en forskningsidé en av de där många nätterna då vi båda var vakna – denna idé hittade sig till slut in i en av artiklarna i denna avhandling. Mest av allt vill jag dock tacka dig för alla de gånger du fått mig att inte *tänka på forskning, eftersom jag varit alltför upptagen med att observera ditt utforskande av världen.*

Härnäst vill jag tacka mina föräldrar för att de uppmuntrade mig att jobba hårt men samtidigt inte pressade mig i någon förutbestämd riktning. Jag uppskattade alltid denna frihet och kommer sträva mot att Juni också ska få känna så när hon växer upp. Kiitos teille molemmille! Till min syster Karolina, tack för att du fanns där för mig genom hela min barndom (och till Magnus, tack för att du finns där för min syster idag). Jag vill även nämna Erik, Monty, Robin, Marielle och Angelos, nära vänner som på olika vis har bidragit till att denna avhandling nu har skickats till tryck. Vidare är jag glad och tacksam för att jag får vara en del av min kära 'Skånefamilj' – skål, Annicka, Gert, Jonna, Emmie och Rikard! Till sist, ett stort, varmt och hjärtligt tack till dig, Jonna, för all kärlek och allt stöd, för ditt djupa lyssnande och kloka rådgivande, och för att du alltid bemöter såväl vänner som främlingar med en sån värme. Jag ser fram emot många nya äventyr med vår lilla kvartett.

Contents

Chapter 1

Overview and Preliminaries

1 Introduction

Many autonomous systems rely on being able to 'see' the world. For instance, a self-driving vehicle should be able to accurately detect both static and dynamic objects, such as lane markings and other cars, respectively. Interactions between objects should also be recognized and predicted to enable safety-critical behavior, e.g. predicting that a pedestrian may suddenly show up from behind an occluding bus. In the field of computer vision the ability to see is more formally called *visual perception*. Nowadays, visual perception systems are most commonly based on deep neural networks, which are expressive computational models that can produce representations suitable for prediction and decision making.

Impressive recent progress has been made across a plethora of visual perception tasks and applications. However, many of the notable breakthroughs have required large-scale annotated datasets for training the deep neural network-based perception models. Since data annotation is often expensive, the size and variability of many contemporary datasets are limited and hence these do not capture all aspects of the rich, noisy and high-dimensional world for which they are proxies. These practically unavoidable limitations of existing datasets are problematic because many standard approaches for visual perception lack mechanisms for adapting to unusual or entirely novel circumstances. Returning to the example of the self-driving vehicle, it would be useful for the underlying visual perception system to be able to refine itself if new data becomes available for its present operating conditions, and/or finely inspect only those parts of the environment which are deemed relevant for decision making. The latter may involve selectively analyzing only the parts of the input stream where the visual perception system is sufficiently confident, to avoid making decisions based on unreliable perception. Note that such a selective processing also has the potential to reduce computational costs and could thus enable faster decision making.

Motivated by the above, in this thesis we present *active* approaches for three different core visual perception tasks: object detection, human pose estimation, and semantic segmentation – see Figure 1.1. The active framings of all tasks are studied in a reinforcement learning (RL) setting. At a high level, RL can be seen as an interaction between an agent and an environment in which the agent takes actions based on what it perceives in the environment and where the state of the environment is subsequently updated. The environment may also provide feedback in terms of rewards, but this feedback is typically delayed with respect to the actions taken by the agent. Thus one of the challenges in RL is to adequately assign credit to actions in order to improve the behavior of the agent (the behavior is assessed by the cumulative reward that the agent obtains during its interaction with an environment). At the same time, being able to learn from delayed rewards is one of the strengths of RL, since one avoids having to manually specify for each state which action is most appropriate.

In the first parts of the thesis we explore how to effectively deploy a given pre-trained perception system, with the goal of reducing computational costs and/or enabling the perception system to focus on those parts where it is most confident. In the last part we study

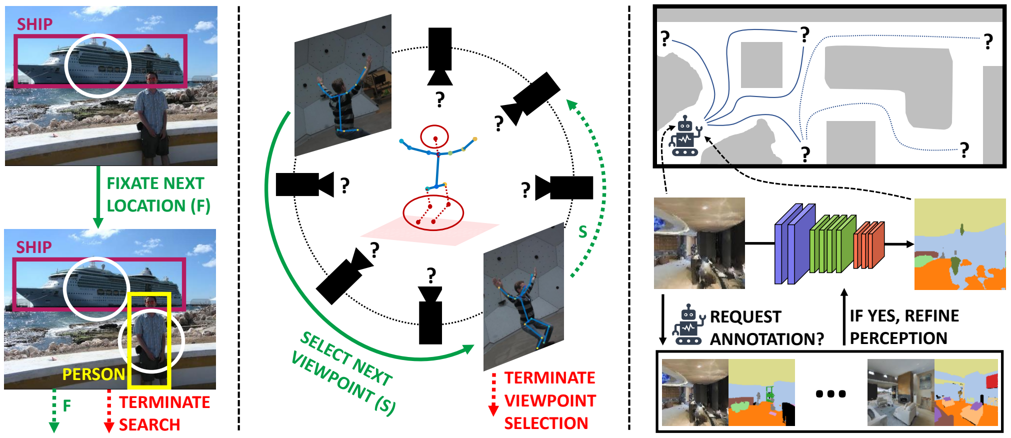

SHIP
FIXATE NEXT LOCATION (F)
SHIP
PERSON
F
TERMINATE SEARCH
SELECT NEXT VIEWPOINT (S)
S
TERMINATE VIEWPOINT SELECTION
REQUEST ANNOTATION?
IF YES, REFINE PERCEPTION

Figure 1.1: In this thesis we have developed reinforcement learning methods (agents) to tackle three different visual perception tasks. Left: Active object detection. Given an image, an agent sequentially fixates different image locations where it detects objects locally, until it automatically terminates search. In this example the ship is detected in the first fixation and the person is detected in the next one. The agent then keeps selecting whether to fixate or terminate search (here it seems reasonable to terminate). Image obtained from [1]. Middle: Active human pose estimation, here 3d reconstruction via active triangulation. Given a set of several viewpoints, an agent sequentially inspects different views where it estimates 2d or 3d poses of all people in the scene, until it terminates viewpoint selection. In this example the person is first observed from the side-back, then from the side-front, and the body parts that are observed from both views are reconstructed (blue). The agent should then select the next viewpoint(s) so as to also reconstruct the head and legs. Images obtained from [2]. Right: Embodied visual active learning, where an agent is tasked to actively explore a 3d environment and selectively request annotations for a limited number of informative views to improve its visual perception. We study the task in the context of semantic segmentation, where the agent may propagate labels in its local neighborhood to further bootstrap from its limited annotated training data. The images corresponding to two different viewpoints were obtained from [3].

how an embodied agent operating in a novel 3d environment can efficiently train or refine its perception, assuming it receives only a limited amount of annotated data. This annotated data is adaptively queried by the agent as it actively explores the environment, and the visual perception system is continually refined as the agent obtains more data.

Common for all setups considered in this thesis is that the proposed active visual perception agents are informed by the visual perception models which they are trying to improve (either it in terms of active sensor positioning or perception refinement). These agents are thus in principle able to adapt to any inherent limits or weaknesses of their underlying perception models. In each task we show through thorough experimental evaluation that active visual perception methods trained with reinforcement learning match or outperform hand-engineered or exhaustive counterparts in task accuracy, often while only negligibly increasing – and sometimes even lowering – computational costs.

This introductory chapter is organized as follows. Section 2 introduces the basic concepts from computer vision and machine learning that are needed to understand the scientific papers in the latter part of the thesis. Specifically, this section mainly revolves around what images are and how they are captured with a pinhole camera, as well as the basics of deep learning and reinforcement learning. In Section 3 we explain the different visual perception tasks studied in the thesis (object detection, human pose estimation, and semantic segmentation). Finally, in Section 4 we summarize the main contributions of this thesis, state limits and assumptions, suggest ideas for future work, and provide an overview of the scientific papers that constitute the latter part of this thesis.

2 Computer Vision and Machine Learning Concepts

This section contains a summary of key computer vision and machine learning concepts that are relevant for understanding the scientific papers in the latter part of this thesis.

2.1 Images and Cameras

Computer vision revolves around the processing, analysis and understanding of images and videos. A grayscale image is an $H \times W$ matrix, where the (i, j):th entry contains the image intensity at the (i, j):th pixel of the image, and H and W denote respectively the image height and width. A common set of values for the intensities are the integers $\{0, 1, \dots, 255\}$, where 0 and 255 correspond to black and white, respectively. An RGB image is an $H \times W \times 3$ tensor, where the three channels encode respectively the intensities of red, green and blue color. Any color that is visible for the human eye can be described by specifying the intensities of these three colors. Finally, an RGBD image is an $H \times W \times 4$ tensor, where the first three channels correspond to an RGB image, and the fourth channel encodes depth (distances to all that is observed in 3d space, see Figure 1.2) at each pixel. RGBD images are less commonly used; in this thesis, the concept of image depth is used only in Paper V. We next explain the basics of the *pinhole camera model*, which is the most common model of how an observation of a 3d scene is transformed into an image.

2.1.1 The Pinhole Camera Model

Since the pinhole camera model is assumed throughout this thesis, we now explain the key characteristics of such a camera. See Figure 1.2 for an overview. For simplicity, we consider an idealized model and omit explaining common imperfections such as radial distortion from camera lenses (refer e.g. to [4] for a more complete description).

Coordinate systems. To describe the configuration of a camera in 3d space, one defines the rotation and translation of the camera center $\boldsymbol{C}$ relative to a given global coordinate system. Assume that the camera coordinate system is obtained by first rotating the global coordinate system with the matrix $\boldsymbol{R}$ and then translating with the vector $\boldsymbol{s}$, and let the point $\boldsymbol{X}$ have coordinates (u, v, w) in the global coordinate system. Then the coordinates of $\boldsymbol{X}$ in

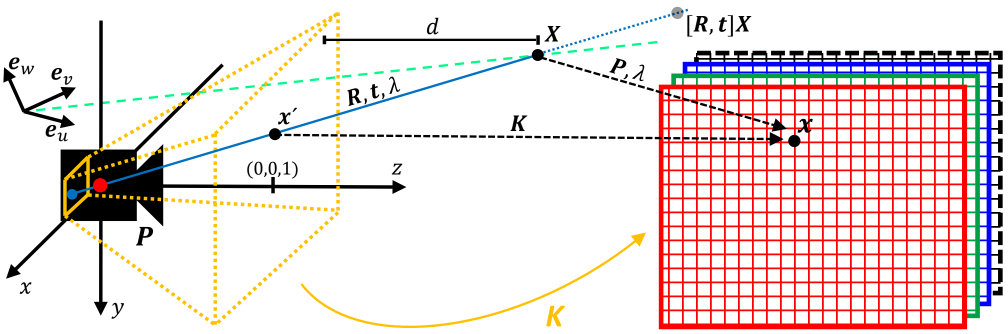


Figure 1.2: Basics of the pinhole camera model. The unit axes $\boldsymbol{e}_u$, $\boldsymbol{e}_v$ and $\boldsymbol{e}_w$ of the global coordinate system are shown on the left. The x-, y- and z-axis of the camera coordinate system are also shown; unless otherwise specified coordinates refer to this system. A camera in $(0, 0, 0)$ points along the z-axis and projects observations onto the image plane, which is parallel to the xy-plane and shown on the left side of the camera. A point $\boldsymbol{X}$ with global coordinates (u, v, w) is projected along the blue line onto the image plane. The image plane is typically thought of as being positioned at $z = 1$ instead, and we will do so as well. Thus the projection point to the left corresponds to $\boldsymbol{x}' = (x', y', 1)$ in the image plane. By augmenting $\boldsymbol{X}$ into $\boldsymbol{X} = (u, v, w, 1)$, the projection into $\boldsymbol{x}'$ can be expressed as a multiplication with $[\boldsymbol{R}, \boldsymbol{t}]$, where $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ and $-\boldsymbol{R}^\top \boldsymbol{t} \in \mathbb{R}^{3\times1}$ are respectively the rotation and translation of the camera coordinate system relative to the global coordinate system. Specifically, given $\boldsymbol{R}$ and $\boldsymbol{t}$, there exists $\lambda \neq 0$ such that $\lambda\boldsymbol{x}' = [\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$. This is because $[\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$ results in a change of coordinates from the global to the camera coordinate system, and $[\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$ together with the camera center span a straight line in this system. Thus, scaling $[\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$ with a particular λ results in $\boldsymbol{x}'$ (without the change of coordinates, such a scaling yields a point on the dashed turquoise line). The camera is described by the camera matrix $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}, \boldsymbol{t}]$, where $\boldsymbol{K}$ is used to transform the image plane into an image with $H \times W$ pixels, shown as the grid on the right. The pixel $\boldsymbol{x}$ corresponding to the image point $\boldsymbol{x}'$ is given by $\boldsymbol{x} = \boldsymbol{K}\boldsymbol{x}'$. Images typically have either three (RGB) or four (RGBD) channels. The depth d of the point $\boldsymbol{X}$ is visualized in the figure; it is the shortest distance from $\boldsymbol{X}$ to the image plane.

the camera coordinate system are given by $\boldsymbol{X}^{\text{cam}} = \boldsymbol{R}(\boldsymbol{X} - \boldsymbol{s})$. Now override notation[1] and set $\boldsymbol{X} = (u, v, w, 1)$. Then the relation between $\boldsymbol{X}^{\text{cam}}$ and $\boldsymbol{X}$ can be expressed as $\boldsymbol{X}^{\text{cam}} = [\boldsymbol{R}, -\boldsymbol{R}\boldsymbol{s}]\boldsymbol{X}$. Introducing $\boldsymbol{t} = -\boldsymbol{R}\boldsymbol{s}$ we thus have $\boldsymbol{X}^{\text{cam}} = [\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$, which is the standard way of expressing the relation between $\boldsymbol{X}^{\text{cam}}$ and $\boldsymbol{X}$.

From the image plane to images. The main characteristic of a pinhole camera is that 3d points are projected onto the image plane of the camera along rays that intersect in the camera center $\boldsymbol{C}$. This implies that objects further away appear smaller in the image, just as they would for the human eye. While the physical image plane resides behind the camera center inside the camera, for practical reasons it is instead imagined as residing in front of the camera, centered at $(0, 0, 1)$ in the camera coordinate system. This is possible because

[1]This is a simplified description. More formally, the pinhole camera model is framed in the context of projective geometry (see e.g. [4]), a topic which we omit in this brief overview.

any point along the straight line between $\boldsymbol{C}$ and $\boldsymbol{X}^{\text{cam}}$ is equivalent in the sense that the camera cannot differentiate between them (they all project to the same point in the physical image plane). Thus, given the rotation matrix $\boldsymbol{R}$ and translation $\boldsymbol{t}$ of the camera, a given point $\boldsymbol{X}$ as seen on the image plane is given by first computing $\boldsymbol{X}^{\text{cam}} = [\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$, then dividing the result by the third coordinate of $\boldsymbol{X}^{\text{cam}}$. The resulting point $\boldsymbol{x}'$ has z-coordinate 1 and hence lies on the image plane. Thus the following relation holds between $\boldsymbol{x}'$ and $\boldsymbol{X}$: There exists $\lambda \neq 0$ such that $\lambda \boldsymbol{x}' = [\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$, where it specifically holds that λ is the reciprocal of the third coordinate of $\boldsymbol{X}^{\text{cam}} = [\boldsymbol{R}, \boldsymbol{t}]\boldsymbol{X}$.

The final component of the basic pinhole camera model is the 3×3 matrix $\boldsymbol{K}$ which describes the inner parameters of the camera. At a high level, $\boldsymbol{K}$ is used to go from the image plane to actual images consisting of pixels. Specifically,

$$\boldsymbol{K} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.1}$$

where f_x and f_y define the focal lengths (typically $f_x = f_y$), s is the skew (typically $s = 0$) and (c_x, c_y) is the focal point. The focal lengths define the scale relationship between image coordinates and pixels, while the focal point essentially dictates where the origin of the image plane ends up in the actual image (it dictates precisely this when $s = 0$).

The relation between a point $\boldsymbol{x}'$ on the image plane and the location $\boldsymbol{x}$ of the corresponding pixel in an image is given by $\boldsymbol{x} = \boldsymbol{K}\boldsymbol{x}'$. Given $\boldsymbol{K}$, $\boldsymbol{R}$ and $\boldsymbol{t}$, the camera is fully described by its camera matrix $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}, \boldsymbol{t}]$. A direct relation between the 3d point $\boldsymbol{X}$ and pixel location $\boldsymbol{x}$ is thus $\lambda \boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}$. In practice, since $\boldsymbol{x} \in \mathbb{R}^3$ while the pixels are in $\{1, \dots, H\} \times \{1, \dots, W\}$, the pixel associated to $\boldsymbol{x}$ is given by first computing $\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}$, then $\hat{\boldsymbol{x}} = \boldsymbol{x}[1\!:\!2]/\boldsymbol{x}[3]$, and finally rounding the entries of $\hat{\boldsymbol{x}}$ to their closest integers.

2.1.2 Image Preprocessing

The visual perception systems used to understand and analyze images typically benefit from slightly preprocessing[2] the images beforehand. One of the most common approaches is a per-image $[-1, 1]$-normalization, which means that the intensity in each image voxel (i, j, c) is transformed from the interval $\{0, \dots, 255\}$ to the $[-1, 1]$-range. Specifically, assume $u \in \{0, \dots, 255\}$. Then $\tilde{u} \in [-1, 1]$ is computed as

$$\tilde{u} = \frac{2u - 255}{255}. \tag{1.2}$$

Another intensity normalization approach involves a whole set of images. It works by computing the average intensities of the red, green and blue channels, and their standard deviations. Before feeding an image to the perception system the channel means are subtracted

[2]We here do not consider preprocessing for enhancing the visual quality of images (e.g. denoising) and assume instead that such steps have already been performed if deemed beneficial or necessary.

and then divided by the respective standard deviations. Thus for the red color channel, the following operation is performed:

$$\tilde{\boldsymbol{I}}_{i,j,r} = \frac{\boldsymbol{I}_{i,j,r} - r_{\text{mean}}}{r_{\text{std}}}, \tag{1.3}$$

where i, j, r refer to the (i, j):th pixel of the red color channel, r_{mean} is the average intensity of red, and r_{std} is the corresponding standard deviation. Analogues of (1.3) are performed for the blue and green channels as well.

2.2 Training, Testing and Validation Sets

In this section we briefly mention the important machine learning concepts of training, testing and validation sets, respectively denoted $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{test}}$ and $\mathcal{D}^{\text{val}}$. Unless otherwise specified $\mathcal{D}$ refers to $\mathcal{D}^{\text{train}}$ throughout this introductory chapter of the thesis.

A parametric machine learning model has a set of parameters which are to be learnt from data. This is typically done by specifying a loss function which describes what the goal or task of the model is (see more in Section 2.4.3). This loss is then minimized on a training dataset $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^D, y^D)\}$, where $\boldsymbol{x}$ is the i:th input (e.g. an image or a feature vector), and y^i is the i:th target (essentially a description of that $\boldsymbol{x}^i$ depicts, e.g. a class label). The loss minimization process is called model training. The purpose of training is typically not only to yield a model which is accurate on the training set, but one which performs well also on unseen data – the model is then said to generalize well. Therefore, to assess machine learning models they are commonly evaluated on one or several test sets $\mathcal{D}^{\text{test}}$, which contain examples (i.e. inputs $\boldsymbol{x}^i$ and corresponding targets y^i) that the model has not seen before.

The validation set $\mathcal{D}^{\text{val}}$ is used during model development, mainly to tune the hyperparameters of the model. Hyperparameters relate to those settings of the model or model training which are not learnt by minimizing a loss function as for standard parameters. Instead one typically tries several configurations of the hyperparameters, trains the model for each, and evaluates which configuration yields the best performance on $\mathcal{D}^{\text{val}}$. Moreover, it is common to use the validation set to ensure that the model does not overfit to the training data (an overfit model generalizes poorly to unseen data). This is done by occasionally evaluating the model on the validation set during training, and early stopping the training procedure once (if) the model performance begins to deteriorate on the validation set.

2.3 Linear Classification with Support Vector Machines

Before diving into nonlinear classification and prediction based on deep learning in Section 2.4, we here briefly describe the *support vector machine* (SVM) [5], a linear classifier that is used in Paper I. As we use the SVM for binary classification we explain that case in this section. Therefore consider a dataset $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^D, y^D)\}$, where $\boldsymbol{x}^i \in \mathbb{R}^m$ is the i:th feature vector and $y^i \in \{-1, +1\}$ is the i:th class label. For example, $\mathcal{D}$ could

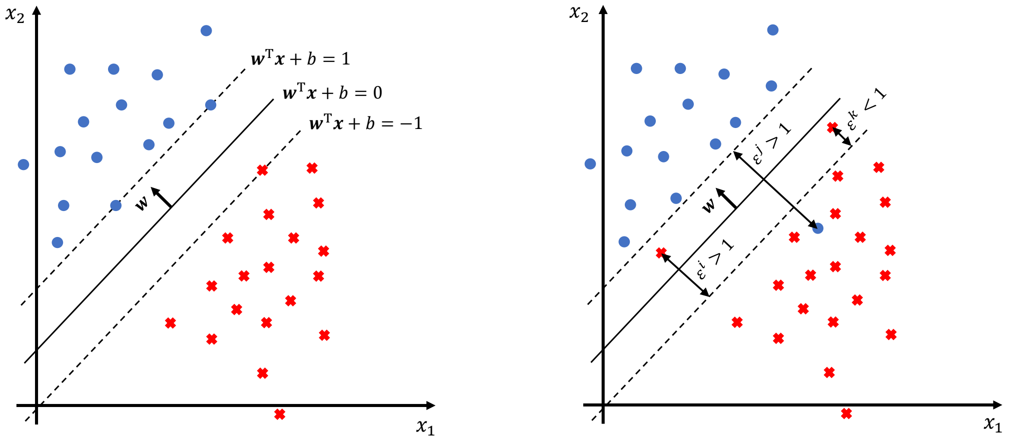


Figure 1.3: Left: Hard-margin SVM for binary linear classification. The two classes (blue circles and red crosses, respectively) are linearly separable. The line $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ separates the classes with maximum margin (the margin is half the distance between the dashed lines). Right: Soft-margin SVM for binary linear classification. This form of SVM allows the classes to be linearly inseparable by introducing a slack variable $\varepsilon^i \geq 0$ for each data point $(\boldsymbol{x}^i, y^i)$. The optimization problem (see (1.6)) then balances between separating the classes and avoiding excessive values for the slack variables. In the example shown, all blue circles above or on the line $\boldsymbol{w}^\top \boldsymbol{x} + b = 1$ and all red crosses below or on the line $\boldsymbol{w}^\top \boldsymbol{x} + b = -1$ have slack variables equal to 0. We also show bounds on the slack variables which are associated with the three data points that do not fall within the respective margins. Note that one of these is still correctly classified, while two are not.

be a dataset consisting of D images of faces and non-faces, including a label for each image, with $y^i = +1$ and $y^i = -1$ corresponding to a face and non-face, respectively.

To the left in Figure 1.3 we show an example dataset $\mathcal{D}$ with feature vectors in $\mathbb{R}^2$. Each blue point (circles) represents a data point $(\boldsymbol{x}^i, y^i)$ with $y^i = +1$, and each red point (crosses) represents a data point $(\boldsymbol{x}^j, y^j)$ with $y^j = -1$. As is seen in the figure, a straight line $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ which perfectly separates the data can be drawn; we therefore say that the data is linearly separable. Every linearly separable dataset has a linear classifier with maximum margin. This is illustrated in the figure, where the line $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ is surrounded on both sides by the two dashed lines $\boldsymbol{w}^\top \boldsymbol{x} + b = \pm 1$. The margin is half the distance between the dashed lines, and the line $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ in the figure is an SVM for binary linear classification. A point is classified as belonging to the class labeled $+1$ if $\boldsymbol{w}^\top \boldsymbol{x} + b > 0$ and as belonging to the class labeled -1 otherwise. Points that lie on any of the dashed lines are called support vectors. Let us now consider how to find the weights $\boldsymbol{w} \in \mathbb{R}^m$ and bias $b \in \mathbb{R}$ that maximizes the margin.

To figure out what optimization problem we should solve, first note that according to the projection formula the margin which we want to maximize is given by

$$\frac{1}{2} \left\| \frac{\boldsymbol{w}^\top (\boldsymbol{x}^+ - \boldsymbol{x}^-)}{\|\boldsymbol{w}\|^2} \boldsymbol{w} \right\|, \tag{1.4}$$

where $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are support vectors with labels $+1$ and -1, respectively. Hence they satisfy $\boldsymbol{w}^\top \boldsymbol{x}^+ + b = 1$ and $\boldsymbol{w}^\top \boldsymbol{x}^- + b = -1$, respectively, which implies that $\boldsymbol{w}^\top(\boldsymbol{x}^+ - \boldsymbol{x}^-) = \boldsymbol{w}^\top \boldsymbol{x}^+ + b - (\boldsymbol{w}^\top \boldsymbol{x}^- + b) = 1 - (-1) = 2$. Thus (1.4) simplifies to $1/\|\boldsymbol{w}\|$, which is our maximization objective. Our constrains are that $(\boldsymbol{w}^\top \boldsymbol{x}^i + b)y^i \geq 1$ should hold for each $i \in \{1, \dots, D\}$. In summary, written as a constrained *minimization* problem, we have the following:

$$\begin{aligned} \min_{\boldsymbol{w}, b} \qquad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{subject to} \qquad & (\boldsymbol{w}^\top \boldsymbol{x}^i + b)y^i \geq 1 \text{ for each } i \in \{1, \dots, D\}. \end{aligned} \tag{1.5}$$

This is a quadratic program with linear inequality constraints and hence a global optimum can be found (see e.g. [6]).

Soft-margin support vector machines. To the right in Figure 1.3 we show a case where a linear classifier cannot perfectly separate the data. This is a common situation in practice. To handle such a situation (and/or to reduce overfitting), we can slightly modify the optimization program (1.5) into:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\varepsilon}} \qquad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{D} \varepsilon^i \\ \text{subject to} \qquad & (\boldsymbol{w}^\top \boldsymbol{x}^i + b)y^i \geq 1 - \varepsilon^i \text{ and } \varepsilon^i \geq 0 \text{ for each } i \in \{1, \dots, D\}, \end{aligned} \tag{1.6}$$

where $C \geq 0$ is a hyperparameter and ε^i is called the i:th slack variable. Note that we have added a third optimization variable $\boldsymbol{\varepsilon} = [\varepsilon^1, \dots, \varepsilon^D]^\top$ in (1.6), and that $C = 0$ makes (1.6) equivalent to (1.5). Like (1.5), the problem (1.6) is a quadratic program with linear inequality constraints and thus a global optimum can be found in this case as well.

2.4 Basics of Deep Learning

All the visual perception systems we study and develop in this thesis are to a significant extent based on *deep neural networks*, the parameters of which are tuned using deep learning. In this section we give an overview of the deep learning concepts and algorithms that are most important for the papers in the thesis – for a more detailed overview, see e.g. [7].

A deep neural network can be seen as a sequence of layers which sequentially process inputs and produce outputs. Each element of an input tensor (e.g. vector or matrix) is called a neuron. Neural networks can be broadly categorized as either feedforward or recurrent. The k:th layer in an L-layer feedforward network is typically of the form

$$\boldsymbol{x}^k = f^k\left(g^k\left(\boldsymbol{x}^{k-1}; \boldsymbol{W}^k, \boldsymbol{b}^k\right)\right), \tag{1.7}$$

where $\boldsymbol{x}^{k-1}$ and $\boldsymbol{x}^k$ are respectively the input and output of the layer, g^k is a linear operation with learnable weights $\boldsymbol{W}^k$ and biases $\boldsymbol{b}^k$, and f^k is the layer's *activation function*, which is typically nonlinear (two subsequent linear layers without a nonlinearity between them can be rewritten as a single linear layer). The input and output of the L-layer network are $\boldsymbol{x}^0$ and $\boldsymbol{x}^L$, respectively. Recall that we in Section 2.3 used $\boldsymbol{x}^i$ and $\boldsymbol{y}^i$ to denote the i:th training sample, while (1.7) overrides that notation so that superscripts instead point to specific layers of a neural network. However, often when we refer to a layer we do not need to refer to the specific layer index k and instead let $\boldsymbol{y} = \boldsymbol{x}^k$, $\boldsymbol{x} = \boldsymbol{x}^{k-1}$, $\boldsymbol{W} = \boldsymbol{W}^k$, $\boldsymbol{b} = \boldsymbol{b}^k$, $g = g^k$ and $f = f^k$.

A recurrent layer differs from a feedforward one in that its present computation depends on computations performed at earlier time steps. Specifically, in its most basic form a recurrent layer is given by

$$\boldsymbol{y}_t = f\left(g\left(\boldsymbol{x}_t, \boldsymbol{y}_{t-1}; \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{W}_R, \boldsymbol{b}_R\right)\right), \tag{1.8}$$

where t denotes the time step (indexes for example frames in a video) and $\boldsymbol{W}_R$ and $\boldsymbol{b}_R$ are the weights and biases associated with the output $\boldsymbol{y}_{t-1}$ of the previous time step.

The remainder of this subchapter is organized as follows. In Section 2.4.1 and 2.4.2 we describe the main network types, layers and activation functions we have used in this thesis. Section 2.4.3 explains how deep learning is used to train neural networks. Finally, in Section 2.4.4 we briefly describe some of the most common techniques for improving neural network training.

2.4.1 Feedforward Networks

To explain the basics of feedforward networks we will consider them in the context of image classification. Given an image and a set $\mathcal{C}$ of C object categories (such as 'human' or 'zebra'), including a background category which means it is none of the $C-1$ other categories, the task of image classification is to predict which category is depicted in the image. An image may of course contain several objects, but for simplicity we assume each image contains one (or zero, in the case of background) instances of some of the $C-1$ object categories. Each category can be uniquely associated with an integer in the set $\{1, \dots, C\}$; we let C be associated with the background label. The task is thus equivalent to predicting which integer $c \in \{1, \dots, C\}$ the image is associated with, which can be formulated as predicting which integer c is the *most likely* to be associated with the image. In practice this task can be solved by designing a system which outputs a C-dimensional probability vector for each image, where the predicted class is set to the index of the largest value in the vector.

The type of model we now consider for going from an image to such a probability vector is a feedforward network, the most common of which is arguably the *convolutional neural network* (CNN) [8]. In CNNs, the first part of the processing is convolutional, where increasingly sophisticated features are produced layer after layer. Then follows a series of *fully connected* (FC) layers, and finally a *softmax* layer is used to output a probability vector over the various object categories. An overview of such a network is given in Figure 1.4.

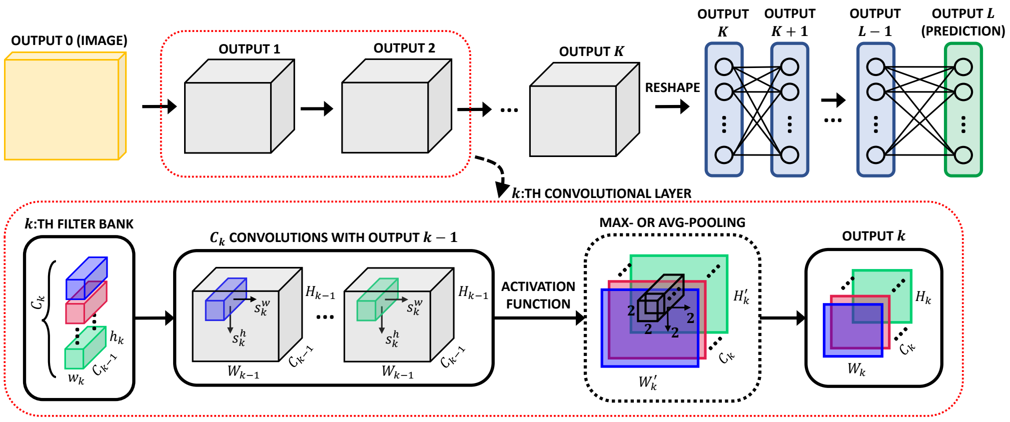


Figure 1.4: Overview of image classification with a basic convolutional neural network. Top: An image is processed by L layers to predict a distribution over object categories. The image is first processed by K convolutional layers. The K:th output is then column-stacked to a vector, which is subsequently processed by $L-K-1$ fully connected (FC) layers. Each layer typically concludes with an element-wise nonlinear activation function. The L:th layer is a softmax function which produces a probability distribution over the object categories. Bottom: A convolutional layer, the specification of which consists of the number of filters C_k, their widths w_k and heights h_k, the horizontal and vertical strides s_k^w and s_k^h, and what padding to use. Each of the C_k filters have the same depth C_{k-1} as the input of the layer. The C_k independent convolutions with the input volume are then applied; each produces a channel of the output (indicated with the different colors). Sometimes other operations are performed prior to obtaining the k:th output, the most common of which are max- and average-pooling. A typical such operation uses a 2×2 sliding window with stride 2 in both directions so as to downsample the size by a factor of 2. Technically, the pooling operation is itself a layer, but here it is for convenience shown inside the k:th convolutional layer.

Common for convolutional and FC layers is the use of activation functions. These are typically nonlinear functions and are applied element-wise on a data structure. We will often use *act* denote a generic activation function. The most common ones are the sigmoid, hyperbolic tangent, and rectified linear unit, which are respectively given by

$$\sigma(x) = \frac{1}{1+\exp(-x)}, \tag{1.9}$$

$$\tanh(x) = \frac{\exp(2x)-1}{\exp(2x)+1}, \tag{1.10}$$

$$\mathrm{ReLU}(x) = \max(0, x). \tag{1.11}$$

Fully connected layers. A fully connected (FC) layer has the form

$$\boldsymbol{y} = \mathrm{act}\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right), \tag{1.12}$$

where $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$ are respectively the input and output of the layer, $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ is the weight matrix, and $\boldsymbol{b} \in \mathbb{R}^n$ is the bias. Figure 1.4 shows an example FC layer between

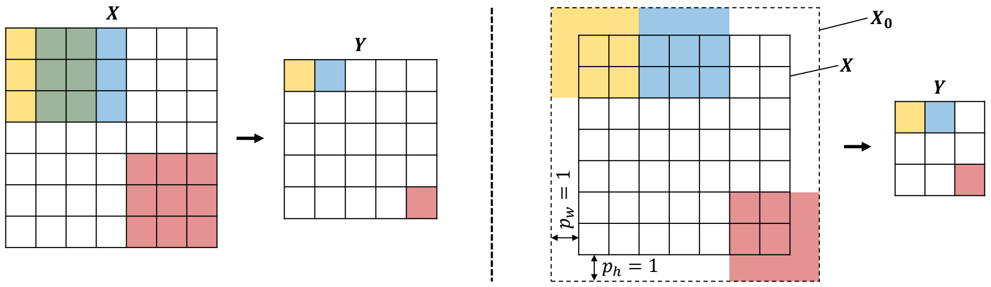


Figure 1.5: Illustrations of how a single 3×3 filter $\boldsymbol{W}_c$ of a convolutional layer operates on a $7 \times 7 \times C_x$ input $\boldsymbol{X}$ to generate the c:th channel $\boldsymbol{Y}_c$ of the output $\boldsymbol{Y}$. Only the spatial dimensions of $\boldsymbol{X}$ are shown; refer to Figure 1.4 to see how the operations look in three dimensions. Activation functions and biases are not shown to avoid visual clutter. Both examples illustrate how the first (yellow), second (blue) and last (red) element of $\boldsymbol{Y}$ are computed given $\boldsymbol{X}$. Left: No horizontal nor vertical padding ($p_h = p_w = 0$). The strides are equal with $s_h = s_w = 1$. According to (1.13), the output size is $(7 - 3 + 2 \cdot 0)/1 + 1 = 5$ in both directions. Right: Equal padding $p_h = p_w = 1$, and thus the convolution operates on the zero-padded version $\boldsymbol{X}_0$ of $\boldsymbol{X}$. The strides are equal with $s_h = s_w = 3$. According to (1.13), the output size is $(7 - 3 + 2 \cdot 1)/3 + 1 = 3$ in both directions. Note that the max- and average-pooling operations look conceptually similar to these visualizations.

outputs K and $K + 1$, and these correspond respectively to $\boldsymbol{x}$ and $\boldsymbol{y}$ in (1.12). FC layers are also known as dense layers, which refers to the large amount of learnable parameters of this type of layer; there are $(m + 1)n$ parameters in total.

Convolutional layers. While FC layers are typically used as vector-to-vector transformations, convolutional layers operate in the matrix-to-matrix and tensor-to-tensor (also called volume-to-volume) regimes. A schematic explanation is provided in Figure 1.5. With input volume $\boldsymbol{X} \in \mathbb{R}^{H_x \times W_x \times C_x}$ it holds that the spatial dimensions H_y and W_y of the output volume $\boldsymbol{Y} \in \mathbb{R}^{H_y \times W_y \times C_y}$ of a convolutional layer are given by

$$\begin{aligned} H_y &= \left\lfloor \frac{H_x - h + 2p_h}{s_h} \right\rfloor + 1, \\ W_y &= \left\lfloor \frac{W_x - w + 2p_w}{s_w} \right\rfloor + 1, \end{aligned} \tag{1.13}$$

where h and w denote respectively the height and width of the convolutional filter, p_h and p_w denote respectively the amount of vertical and horizontal padding, and s_h and s_w denote respectively the vertical and horizontal stride (cf. Figure 1.5). The (a, b, c):th entry of the output $\boldsymbol{Y}$ is given by

$$\boldsymbol{Y}(a, b, c) = \text{act}\left(b_c + \sum_{i,j,k} \boldsymbol{W}_c(i, j, k) \boldsymbol{X}_0\left(i + s_h(a-1), j + s_w(b-1), k\right) \right), \tag{1.14}$$

where b_c and $\boldsymbol{W}_c \in \mathbb{R}^{h\times w\times C_x}$ are respectively the bias and weight filter for the c:th channel of $\boldsymbol{Y}$, and $\boldsymbol{X}_0 \in \mathbb{R}^{(H_x+2p_h)\times(W_x+2p_w)\times C_x}$ is the zero-padded version of $\boldsymbol{X}$. Specifically, the (a,b,c):th entry of $\boldsymbol{X}_0$ is given by

$$\boldsymbol{X}_0(a,b,c) = \begin{cases} \boldsymbol{X}(a-p_h, b-p_w, c) \text{ if } a \in (p_h, H_x+p_h],\ b \in (p_w, W_x+p_w] \\ 0 \text{ else.} \end{cases} \tag{1.15}$$

A convolutional layer can be compactly expressed as

$$\boldsymbol{Y} = \text{act}\left(\boldsymbol{W} * \boldsymbol{X} + \boldsymbol{B}\right), \tag{1.16}$$

where $\boldsymbol{W}$ contains all weight filters and $\boldsymbol{B}$ all biases. Note that all values in the c:th channel of $\boldsymbol{B}$ are equal to b_c. The total number of learnable parameters in a convolutional layer is $(hwC_x+1)C_y$, which is typically much smaller than in a fully connected layer.

Max- and average-pooling layers. It is common to apply some form of pooling after some of the convolutional layers of a CNN. The most common pooling layers are max- and average-pooling, which like convolutions are applied in a sliding window fashion, although different from convolutional layers they do not have learnable parameters. With notation analogous to that of (1.14), the max- and average-pooling layers are respectively given by

$$\boldsymbol{Y}(a,b,c) = \max_{\substack{1\le i\le h,\\ 1\le j\le w}} \boldsymbol{X}_0\left(i+s_h(a-1), j+s_w(b-1), c\right) \tag{1.17}$$

and

$$\boldsymbol{Y}(a,b,c) = \frac{1}{hw}\sum_{i=1}^{h}\sum_{j=1}^{w} \boldsymbol{X}_0\left(i+s_h(a-1), j+s_w(b-1), c\right), \tag{1.18}$$

where h and w denote respectively the height and width of the pooling kernel. Note that while a convolutional layer operates in all three dimensions of the input volume to produce a single output channel, a pooling layer operates independently on each channel of the input to produce the output (thus $\boldsymbol{Y}$ always has the same number of channels as $\boldsymbol{X}$).

Softmax layer. Recall how in the beginning of this section we mentioned that the output of an image classification system should be a probability vector. The most common way to achieve this is by letting the final layer of the neural network be a softmax layer, which is defined as follows given $\boldsymbol{x} \in \mathbb{R}^m$:

$$\text{softmax}(\boldsymbol{x}) = \frac{\left[\exp(x_1), \ldots, \exp(x_m)\right]^\top}{\sum_{i=1}^m \exp(x_i)}. \tag{1.19}$$

Thus the softmax layer yields an m-dimensional probability vector, since each element is in the range $(0,1)$ and the elements sum to 1. It does not contain learnable parameters.

2.4.2 Recurrent Networks

We now have a basic understanding of feedforward networks, more specifically convolutional neural networks (CNNs). However, there exist several problems which require the processing of not only a single image (frame), but a whole series of connected frames which depend on each other. This is the case in for example video classification, where the task is to predict what a given video depicts (which could be various actions such as 'woman playing a guitar', 'dog running', and so on). In some cases it may be sufficient to only look at a single frame to predict what a video depicts, but often one has to observe a larger sequence to be successful. For example, it may be difficult to distinguish walking from running given only a single image. The feedforward CNN we described in the previous section is not able to meaningfully process such sequential data, but there exist several other model types which can be used, such as recurrent neural networks (RNNs) [9, 10], 3d CNNs [11, 12], or transformers [13, 14]. Some methodologies proposed in this thesis rely on RNNs, so we next give a brief overview of such networks.

Vanilla recurrent layer. Recall from (1.12) that a fully connected layer has the form $\boldsymbol{y} = \mathrm{act}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$. In sequential processing we add a subscript t to denote which time step we are considering. The fully connected layer then becomes $\boldsymbol{y}_t = \mathrm{act}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b})$. A straightforward way to make such a layer recurrent is to let $\boldsymbol{y}_t$ depend also on the output $\boldsymbol{y}_{t-1}$ from the previous time step, i.e. $\boldsymbol{y}_t = \mathrm{act}(\boldsymbol{W}_{xy}\boldsymbol{x}_t + \boldsymbol{b}_{xy} + \boldsymbol{W}_{yy}\boldsymbol{y}_{t-1} + \boldsymbol{b}_{yy})$. Note that the sum of the two biases $\boldsymbol{b}_{xy}$ and $\boldsymbol{b}_{yy}$ can be replaced by a single one, so we instead expresses this layer as

$$\boldsymbol{y}_t = \mathrm{act}(\boldsymbol{W}_{xy}\boldsymbol{x}_t + \boldsymbol{W}_{yy}\boldsymbol{y}_{t-1} + \boldsymbol{b}_y). \tag{1.20}$$

It is good practice to let the activation function output bounded values so that the recurrence does not cause a numerical explosion; a common choice in RNNs is the $\tanh$ activation. While the vanilla recurrent layer (1.20) works for sequential tasks in which sequences consist of a few time steps, there exist more robust variants which are better suited for longer-horizon setups. We next detail the two most popular such variants.

Long short-term memory (LSTM). The LSTM [9] is the most famous and most widely used recurrent module. The equations for an LSTM are provided below:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \tag{1.21}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \tag{1.22}$$

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \tag{1.23}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t, \tag{1.24}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \tag{1.25}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t). \tag{1.26}$$

The $\boldsymbol{c}_t$-variable is the main carrier and accumulator of temporal information and is called the cell state. In each time step it is updated according to (1.24), which combines past information $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$ and new information $\boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$. The weightings of the two are given by the forget and input gates $\boldsymbol{f}$ and $\boldsymbol{i}$, which are respectively given in (1.21) and (1.22). These weights are multiplied element-wise, denoted by the $\odot$ operator. Note that the new information $\tilde{\boldsymbol{c}}_t$ given in (1.23) is computed in a way that looks similar to the vanilla recurrent layer (1.20). Finally, despite its name, the hidden state $\boldsymbol{h}_t$ is the variable which is output from the LSTM and can be fed to subsequent layers of the network. The hidden state is obtained by feeding $\boldsymbol{c}_t$ through a $\tanh$ activation and then weighting the result according to the output gate defined in (1.25). Note that the forget, input and output gates provide weights in the $(0, 1)$-range due to the sigmoids.

Gated recurrent unit (GRU). The GRU [10] is quite similar to the LSTM but has a somewhat simpler form:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_{xz}\boldsymbol{x}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1} + \boldsymbol{b}_z), \tag{1.27}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{xr}\boldsymbol{x}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1} + \boldsymbol{b}_r), \tag{1.28}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h), \tag{1.29}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t. \tag{1.30}$$

Different from the LSTM it does not track two recurrent variables and instead only keeps track of the single recurrent variable $\boldsymbol{h}_t$. Also, there are only two gating variables: $\boldsymbol{z}_t$ is the update gate and determines how much of the old and new information to use, respectively, while $\boldsymbol{r}_t$ is the reset gate and is used to weigh the importance of old information $\boldsymbol{h}_{t-1}$ within the new information $\tilde{\boldsymbol{h}}_t$.

Convolutional recurrent layers. In the above we have provided fully connected formulations of recurrent layers, i.e. the operations are based on dense matrix multiplications. However, it is also possible to use convolutional implementations [15], which may be more suitable for spatio-temporal processing. For example, the convolutional GRU (conv-GRU), is given by the following equations:

$$\boldsymbol{Z}_t = \sigma(\boldsymbol{W}_{xz} * \boldsymbol{X}_t + \boldsymbol{W}_{hz} * \boldsymbol{H}_{t-1} + \boldsymbol{B}_z), \tag{1.31}$$

$$\boldsymbol{R}_t = \sigma(\boldsymbol{W}_{xr} * \boldsymbol{X}_t + \boldsymbol{W}_{hr} * \boldsymbol{H}_{t-1} + \boldsymbol{B}_r), \tag{1.32}$$

$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{W}_{xh} * \boldsymbol{X}_t + \boldsymbol{W}_{hh} * (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}) + \boldsymbol{B}_h), \tag{1.33}$$

$$\boldsymbol{H}_t = (1 - \boldsymbol{Z}_t) \odot \boldsymbol{H}_{t-1} + \boldsymbol{Z}_t \odot \tilde{\boldsymbol{H}}_t. \tag{1.34}$$

2.4.3 Training Neural Networks

When a neural network is constructed its parameters are typically initialized in some heuristic manner. For example, one can set each initial weight and bias to an independent sample from the $\mathcal{N}(0,1)$-distribution (more sophisticated initialization techniques exist and are used in this thesis, such as Xavier initialization [16]). In this section we briefly explain, again in the context of image classification, the key steps and components involved when training a neural network so that it may perform well on a given task. We focus the presentation on how to train feedforward networks – the training of recurrent networks is performed in essentially the same way, but requires some additional considerations (see e.g. Chapter 10.2.2 in [7]). In the image classification setting we are given a training dataset $\mathcal{D} = \{(\boldsymbol{X}^1, \boldsymbol{y}^1), \dots, (\boldsymbol{X}^D, \boldsymbol{y}^D)\}$, where $\boldsymbol{X}^i \in \mathbb{R}^{H \times W \times 3}$ is the i:th image and $\boldsymbol{y}^i \in \{0,1\}^C$ is the ground truth onehot vector corresponding to $\boldsymbol{X}^i$ (recall that $C = |\mathcal{C}|$ denotes the number of object categories, including background). An important concept as we proceed is that of minibatches, or batches for short. A random minibatch $\mathcal{B} \subset \mathcal{D}$ of size $|\mathcal{B}| = B$ is given by $\mathcal{B} = \{(\boldsymbol{X}^{i_1}, \boldsymbol{y}^{i_1}), \dots, (\boldsymbol{X}^{i_B}, \boldsymbol{y}^{i_B})\}$, where the index set $\{i_1, \dots, i_B\}$ is a uniform random subset of the full index set $\{1, \dots, D\}$.

Loss functions. Given a basic understanding of the main building blocks of feedforward CNNs (cf. Section 2.4.1), let us now consider how one can tune their parameters to make them accurate at a given task. With the running example of image classification in mind, we want to be able to adjust the various parameters of the network (i.e. the weights and biases of the convolutional and fully connected layers) so that the final softmax layer produces reasonable probability distributions. For example, given an image of a dog, we want the CNN to produce a vector which has high probability for the entry corresponding to 'dog', and low for the others.

To achieve this we first need to define an appropriate *loss function* (or cost function) which aligns with the desired behavior of the network. The foundational loss function for image classification is the cross-entropy. Letting $\boldsymbol{\theta}$ denote all learnable parameters of the neural network, this loss is given by

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = -\frac{1}{D} \sum_{i=1}^{D} \sum_{j=1}^{C} \left[y_j^i \log \hat{y}_j^i(\boldsymbol{\theta}) + (1 - y_j^i) \log \left(1 - \hat{y}_j^i(\boldsymbol{\theta})\right) \right]. \tag{1.35}$$

Here y_j^i is the j:th component of the i:th ground truth vector $\boldsymbol{y}^i$ in $\mathcal{D}$, and $\hat{y}_j^i(\boldsymbol{\theta})$ is the j:th component of the i:th probability vector $\hat{\boldsymbol{y}}^i(\boldsymbol{\theta})$ produced by the neural network based on input $\boldsymbol{X}^i$ in $\mathcal{D}$. The dependence of $\hat{\boldsymbol{y}}^i$ on $\boldsymbol{\theta}$ will henceforth not be explicitly expressed to avoid clutter. Since in the image classification setup each $\boldsymbol{y}^i$ is a onehot vector – that is, a single element is equal to one and the rest are equal to zero – (1.35) simplifies to

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = -\frac{1}{D} \sum_{i=1}^{D} \log(\hat{y}_{c_i}^i), \tag{1.36}$$

where $\hat{y}_{c_i}^i$ is the predicted probability that $\boldsymbol{X}^i$ depicts category c_i, when c_i is the correct category. Thus minimizing the cross-entropy loss (1.36) is equivalent to maximizing the average of the correct-class probability estimates, so this loss aligns well with the desired behavior of the network.

Another common loss function is the L2-loss, or mean squared error, which is given by

$$\mathcal{L}\left(\boldsymbol{\theta};\mathcal{D}\right) = -\frac{1}{2D}\sum_{i=1}^{D}\left(\boldsymbol{y}^i - \hat{\boldsymbol{y}}^i\right)^\top\left(\boldsymbol{y}^i - \hat{\boldsymbol{y}}^i\right) = -\frac{1}{2D}\sum_{i=1}^{D}\sum_{j=1}^{n}\left(y_j^i - \hat{y}_j^i\right)^2. \qquad (1.37)$$

The L2-loss is not suitable when the task is to predict a probability vector from a given input, but is instead commonly used in regression tasks, where $\boldsymbol{y}^i, \hat{\boldsymbol{y}}^i \in \mathbb{R}^n$.

When minimizing a neural network loss one typically resorts to a first-order optimization method based on some form of gradient descent. An example of a gradient descent-based update rule for $\boldsymbol{\theta}$ is $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{D})$, where $\alpha > 0$ is the learning rate. Other step directions than $-\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{D})$ are possible (and used in this thesis), some of which are described in Section 2.4.4. For more in-depth overviews of first-order optimization methods and parameter update rules, refer e.g. to [7, 17].

Note that the cross-entropy and L2-losses over a dataset $\mathcal{D}$ can be written as averages of losses of individual data points, i.e.

$$\mathcal{L}\left(\boldsymbol{\theta};\mathcal{D}\right) = \frac{1}{D}\sum_{i=1}^{D}\mathcal{L}\left(\boldsymbol{\theta};d^i\right), \qquad (1.38)$$

where $d^i = (\boldsymbol{X}^i, \boldsymbol{y}^i)$ is the i:th element of $\mathcal{D}$. The property (1.38), which in general holds for loss functions in deep learning, together with the linearity of the gradient give that

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\theta};\mathcal{D}\right) = \frac{1}{D}\sum_{i=1}^{D}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\theta};d^i\right). \qquad (1.39)$$

Thus the gradient of the full loss is given by simply averaging the gradients of the losses over the individual data points d^i. However, instead of computing the full average one typically resorts to *stochastic gradient descent* (SGD) which approximates this average over a random minibatch $\mathcal{B} \subset \mathcal{D}$ of size $B < D$ as

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\theta};\mathcal{D}\right) = \frac{1}{D}\sum_{i=1}^{D}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\theta};d^i\right) \approx \frac{1}{B}\sum_{i=1}^{B}\nabla_{\boldsymbol{\theta}}\mathcal{L}\left(\boldsymbol{\theta};b^i\right), \qquad (1.40)$$

where b^i is the i:th element of $\mathcal{B}$. The approximation (1.40) becomes more accurate as B increases towards D, but typically $B \ll D$. In practice one typically partitions $\mathcal{D}$ as $\mathcal{D} = \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_M$ and performs parameter updates based on each $\mathcal{B}_i$; this is called an epoch of $\mathcal{D}$. After an epoch a new random partition is computed and the process repeats until training finishes. The number of training iterations is often set based on early stopping on a separate validation set $\mathcal{D}^{\text{val}}$ (cf. Section 2.2).

There are two main reasons why SGD is preferred over using full gradient steps in the context of deep learning. First, SGD requires significantly less computation per parameter update. In contrast, for large D each update based on a full gradient step (1.39) is associated with a heavy computational footprint. Second, full gradient steps cannot be used in cases where a dataset $\mathcal{D}$ does not exist a priori, i.e. where data is continually received in an online setting. This is the case for example in reinforcement learning (see Section 2.5), a methodology which is used extensively in the papers of this thesis.

Backpropagation. So far we have covered the basics of neural networks, their inputs and outputs, and provided some of the basic components used to train such networks, i.e. loss functions and stochastic gradient descent. We will now look at how to efficiently compute an individual gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{X}^i, \boldsymbol{y}^i)$, which we will write simply as $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$ to avoid clutter with superscripts. To see why it is not obvious how to efficiently compute this gradient we consider the cross-entropy loss in (1.36) and expand it over all the layers:

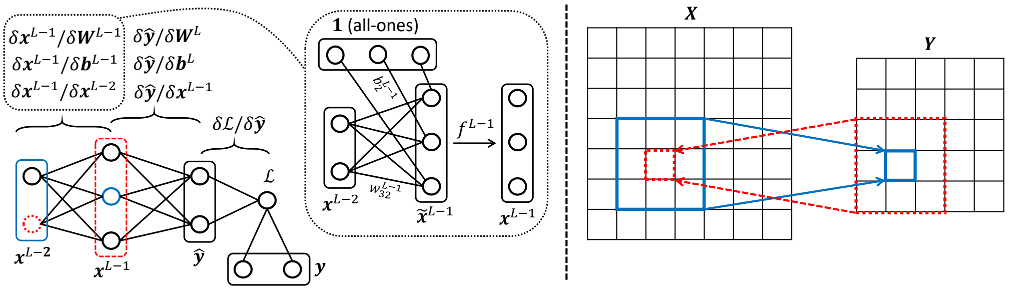
$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = -\nabla_{\boldsymbol{\theta}} \log(\hat{y}_c) = -\nabla_{\boldsymbol{\theta}} \log([f^L \circ f^{L-1} \circ \cdots \circ f^1(\boldsymbol{X})]_c), \qquad (1.41)$$

where each f^k corresponds to $f^k \circ g^k$ in (1.7), i.e. we override notation to avoid having to explicitly reference for each layer both the activation function f and the linear operator g. Each layer f^k is as usual parametrized by its weights and biases $\boldsymbol{W}^k$ and $\boldsymbol{b}^k$, respectively. Finally, $[f^L \circ f^{L-1} \circ \cdots \circ f^1(\boldsymbol{X})]_c$ denotes the c:th component of the output of the network, where c is the index of the object category depicted in the image $\boldsymbol{X}$.

Given the expanded expression (1.41), we note that we need to find the gradient of a composition of several functions. The resulting algorithm for computing gradients such as (1.41) is called *backpropagation* [18]. It begins by computing the gradients at the loss layer, then gradually propagates gradients backwards (hence the name of the algorithm) all the way back to the input layer using the chain rule. Backpropagation is conceptually illustrated in Figure 1.6 for a fully connected and convolutional layer, respectively. Details of the algorithm are given e.g. in [7], where *backpropagation through time* (BPTT), a variant that is used for recurrent neural networks, is also explained.

Automatic differentiation. Modern deep learning libraries (such as Tensorflow [19] and Caffe [20], the two libraries used in this thesis) include automatic differentiation functionalities. This means that the one implementing a neural network only has to specify the structure of the network (i.e. what its layers, inputs and outputs are), the loss function(s), as well as what optimizer to use. The loss minimization is then performed by calling the built-in optimizer, which automatically performs the backpropagation steps.

Automatic differentiation works due to the compositional nature of neural networks – by specifying for each layer type its forward and backward (gradient) computation, backpropagation can sequentially update all parameters given any composition of such layers. The popular automatic differentiation libraries contain a wide range of common layers,



$\delta x^{L-1}/\delta W^{L-1}$
$\delta x^{L-1}/\delta b^{L-1}$
$\delta x^{L-1}/\delta x^{L-2}$
$\delta \hat{y}/\delta W^{L}$
$\delta \hat{y}/\delta b^{L}$
$\delta \hat{y}/\delta x^{L-1}$
$\delta \mathcal{L}/\delta \hat{y}$
$\mathcal{L}$
x^{L-2}
x^{L-1}
$\hat{y}$
y
1 (all-ones)
b_2^{L-1}
f^{L-1}
w_{32}^{L-1}
x^{L-2}
$\tilde{x}^{L-1}$
x^{L-1}
X
Y

Figure 1.6: Conceptual explanations of backpropagation for fully connected (left) and convolutional (right) layers. In these examples, the solid blue and dashed red neurons indicate respectively computational dependence during the forward and backward passes (the set of neurons on which the computation depends is also indicated with the corresponding color). Note that the nature of this dependence looks symmetrical in both directions; e.g. in the convolutional layer the forward and backward passes are both implemented as convolutions. We now provide more details for the backward pass of the fully connected graph to the left. Beginning at the loss $\mathcal{L}$, partial derivatives are sequentially computed from right to left using the chain rule. Details of what occurs between $\boldsymbol{x}^{L-2}$ and $\boldsymbol{x}^{L-1}$ is shown within the dotted region. Let us now look at how to obtain $\partial\mathcal{L}/\partial\boldsymbol{x}^{L-2}$, $\partial\mathcal{L}/\partial\boldsymbol{W}^{L-1}$ and $\partial\mathcal{L}/\partial\boldsymbol{b}^{L-1}$, by considering one element of each. To avoid clutter, let $\boldsymbol{x} = \boldsymbol{x}^{L-2}$, $\boldsymbol{z} = \boldsymbol{x}^{L-1}$, $\boldsymbol{W} = \boldsymbol{W}^{L-1}$, $\boldsymbol{b} = \boldsymbol{b}^{L-1}$, and $f = f^{L-1}$. Finally, let $\tilde{\boldsymbol{z}}$ denote the input to the activation function f, i.e. $\boldsymbol{z} = f(\tilde{\boldsymbol{z}})$. From the chain rule we have that $\partial\mathcal{L}/\partial x_i = \sum_j(\partial\mathcal{L}/\partial z_j)(\partial z_j/\partial x_i)$, where $\partial\mathcal{L}/\partial z_j$ has already been computed (as it is closer to the loss layer). Meanwhile, $\partial z_j/\partial x_i = w_{ji}f'(\tilde{z}_j)$. Thus $\partial\mathcal{L}/\partial x_i = \sum_j(\partial\mathcal{L}/\partial z_j)w_{ji}f'(\tilde{z}_j)$. Next, $\partial\mathcal{L}/\partial w_{ij} = \sum_k(\partial\mathcal{L}/\partial z_k)(\partial z_k/\partial w_{ij})$, where $\partial z_k/\partial w_{ij} = x_j f'(\tilde{z}_i)$ if $k = i$ and $\partial z_k/\partial w_{ij} = 0$ if $k \neq i$. Thus $\partial\mathcal{L}/\partial w_{ij} = (\partial\mathcal{L}/\partial z_i)x_j f'(\tilde{z}_i)$. Finally, $\partial\mathcal{L}/\partial b_i = \sum_j(\partial\mathcal{L}/\partial z_j)(\partial z_j/\partial b_i)$, where $\partial z_j/\partial b_i = f'(\tilde{z}_i)$ if $j = i$ and $\partial x_j/\partial b_i = 0$ if $j \neq i$. Thus $\partial\mathcal{L}/\partial b_i = (\partial\mathcal{L}/\partial z_i)f'(\tilde{z}_i)$.

including their gradients, as well as options for the users to specify those computations for layers that are not already included in the respective libraries.

2.4.4 Common Techniques for Improving Neural Network Training

A typical problem when naively training neural networks is that they tend to overfit to the training data, since the number of learnable parameters is typically very large in relation to the size of the training dataset. In practice this means that the network obtains high accuracy in training but that the accuracy drops severely when testing the network on unseen data (i.e. it generalizes poorly). Another issue may be the training stability itself. For example, a too large learning rate, possible coupled with inappropriate gradient step directions, may result in highly suboptimal network parameters. Fortunately, several techniques exist to improve the way in which neural networks learn. We here consider a few of the most commonly used such techniques – with a sole focus on those which are used in this thesis – which often enable the trained networks to generalize better to unseen data.

Weight decay. Overfitting is typically associated with with large network weights, so a basic strategy to avoid overfitting, known as weight decay or L2-regularization, is to encourage smaller weights. Specifically, weight decay introduces the augmented loss function

$$\mathcal{L}_\lambda(\boldsymbol{\theta}; \mathcal{D}) = \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) + \lambda \|\tilde{\boldsymbol{\theta}}\|_2, \tag{1.42}$$

where $\lambda \geq 0$ is a hyperparameter (a larger λ implies smaller network weights and hence stronger regularization), and $\tilde{\boldsymbol{\theta}}$ denotes all network parameters except the biases.

Dropout. Another common regularization technique is dropout [21]. It is most commonly applied to fully connected layers and is therefore explained for them. During training, a uniform random subset of the connection weights in a layer are set to zero (both in the forward and backward pass). The subset of weights that are set to zero differs in each minibatch. Intuitively and empirically, using dropout yields more robust connection weights, since the neurons tend to develop less co-dependence. The fraction p of neurons dropped is a hyperparameter, which is commonly set to 0.5.

Residual blocks. We here briefly mention residual blocks [22], which are used in many contemporary state-of-the-art deep neural networks. To explain these, consider a stack of feedforward layers $H(\boldsymbol{x})$, where $\boldsymbol{x}$ denotes the input to the first layer in the stack (the stack may consist of a single layer). Assuming that $H(\boldsymbol{x})$ and $\boldsymbol{x}$ have the same dimension, then $H(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}$ with $F(\boldsymbol{x}) = H(\boldsymbol{x}) - \boldsymbol{x}$. A residual block alters the neural network design by replacing a layer stack $H(\boldsymbol{x})$ with $F(\boldsymbol{x}) + \boldsymbol{x}$. In terms of network implementation, this simply means feeding an (intermediate) input $\boldsymbol{x}$ both to a learnable layer stack F and to an identity mapping, then adding the results. The feeding of $\boldsymbol{x}$ through an identity mapping is called a shortcut (or skip) connection, and the sum $F(\boldsymbol{x}) + \boldsymbol{x}$ is a residual block. A neural network which includes residual blocks is called a residual network. Such networks are often easier to optimize than counterparts without residual blocks, which is especially the case for very deep networks consisting of tens or hundreds of layers.

Momentum. One of the most common modifications to the standard SGD update rule $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$, with $\mathcal{B}$ a minibatch of a dataset $\mathcal{D}$, is:

$$\begin{aligned} \boldsymbol{m} &\leftarrow \beta \boldsymbol{m} + (1 - \beta) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{B}), \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{m}. \end{aligned} \tag{1.43}$$

Here $\beta \in [0, 1)$ is a hyperparameter (setting $\beta = 0$ corresponds to the standard SGD update rule, and a common choice when using momentum is $\beta = 0.9$). The momentum vector $\boldsymbol{m}$ can either be initialized as all-zeros or be set equal to the first gradient. The effect of the momentum update rule (1.43) is that the updates to the parameters $\boldsymbol{\theta}$ are based on an exponential moving average of earlier gradient estimates, as opposed to being based only on the current one. This typically results in less fluctuating updates and faster learning.

Adaptive Moment Estimation (Adam). Another widely used modification to standard SGD is Adam [23]. This update rule is defined as follows:

$$\begin{aligned}
\boldsymbol{m} &\leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{B}), \\
\boldsymbol{v} &\leftarrow \beta_2 \boldsymbol{v} + (1-\beta_2)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{B}) \odot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{B}), \\
\boldsymbol{m}^{\text{corr}} &\leftarrow \frac{\boldsymbol{m}}{1-\beta_1^t}, \\
\boldsymbol{v}^{\text{corr}} &\leftarrow \frac{\boldsymbol{v}}{1-\beta_2^t}, \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha\frac{\boldsymbol{m}^{\text{corr}}}{\sqrt{\boldsymbol{v}^{\text{corr}}}+\varepsilon}.
\end{aligned} \tag{1.44}$$

Here $\boldsymbol{m}$ and $\boldsymbol{v}$ are exponential moving averages of the first (mean) and second (non-centered variance) moments of the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta};\mathcal{B})$, respectively. In addition to the learning rate α there are three hyperparameters to set in Adam: $\beta_1, \beta_2 \in [0,1)$ and $\varepsilon \geq 0$ (typical values are $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$), where ε is added in the denominator of the parameter update to improve numerical stability. Both $\boldsymbol{m}$ and $\boldsymbol{v}$ are corrected in each training iteration (indexed by $t \in \{1,2,\dots\}$) by dividing with $1-\beta_1^t$ and $1-\beta_2^t$, respectively, prior to performing the update of $\boldsymbol{\theta}$. Note that β_1^t and β_2^t here refer respectively to β_1 and β_2 to the power of t. Finally, note that the update rule for $\boldsymbol{\theta}$ is to be interpreted element-wise in (1.44), i.e. the square root, addition of ε and division are all applied on a per-element basis.

Data augmentation. The above mentioned techniques either alter the neural network structure, loss function, or parameter update rule. A complementary option is to modify the training data such that when trained on, the neural network is more likely to generalize better. This procedure is called data augmentation, examples of which include adding to the training set left-right flipped, randomly cropped, and/or color distorted versions of the images in the original training set. Data augmentation is thus a bootstrapping process by which the neural network obtains more data to train on 'for free'.

2.5 Basics of Reinforcement Learning

In this section we give an overview of the reinforcement learning (RL) concepts and algorithms that are fundamental to the papers in the thesis. There are several variants of RL, mainly differentiated by whether the learning is value-based or policy-based. The papers in this thesis rely on the latter, so the key focus will be on explaining the pre-requisites of policy-based RL (but see e.g. [24] for a more comprehensive coverage). In Section 2.5.1 we introduce the main terminology and concepts of RL, while Section 2.5.2 focuses on the formulations that are most relevant for this thesis.

2.5.1 Reinforcement Learning Terminology and Concepts

The image and video classification problems that were used to explain the core deep learning concepts in Section 2.4 are examples of *supervised* and *passive* learning problems. They are supervised in the sense that the training involves a labeled dataset $\mathcal{D}$ of inputs and expected outputs. The expected outputs are often well-defined and can be provided by humans that annotate the data beforehand. The problems are passive in that they involve perception systems which only have to observe inputs and predict associated outputs – the systems cannot affect any parts of the input themselves. Thus the methodologies discussed so far are useful for learning to *perceive* an environment.

To make a system *act* rationally in relation to its perceived observations involves several additional challenges. Consider for example a self-driving car that should navigate from a start to a goal location in an urban scene. Let us for simplicity assume that the car has a single forward mounted RGB camera for observing its surroundings and that it has three discrete actions it can take: drive a step in the left, forward or right direction (where the left and right steps correspond to first rotating the car to the left or right and then moving forward in the new direction). It does not seem suitable to frame the training of this car in a fully supervised framework, as it would rarely make sense or be practical to label each possible RGB image that the car can see with a 'correct' action to take. Nor is the problem passive – it is active in the sense that the actions taken by the car in the current step affect what it sees in the next step. We will later consider how these types of problems can be tackled with reinforcement learning, which is one of the main approaches for training agents to act rationally in a given environment. First we must however introduce some fundamental reinforcement learning terminology.

States, actions and environment dynamics. The core framework in which reinforcement learning (RL) is studied is called a *Markov decision process* (MDP), which provides a practical formalism for describing the interaction process of an agent within an environment. Another key concept is the *policy distribution* which dictates how an agent acts in an environment. The goal of RL is to learn a policy that successfully solves a given task (or set of tasks) in an environment.

As the name suggests, in MDPs one assumes that the decision making and state transitions have the Markov property, i.e. that the immediate future depends only on the current state and action. Given a policy π, the evolution of an MDP can be illustrated as follows:

$$s_0 \xrightarrow{\pi(*|s_0)} a_0 \xrightarrow{p(*,*|s_0,a_0)} r_1, s_1 \xrightarrow{\pi(*|s_1)} a_1 \xrightarrow{p(*,*|s_1,a_1)} \dots \xrightarrow{p(*,*|s_{T-1},a_{T-1})} r_T, s_T \tag{1.45}$$

Here s_0 is an initial state which is sampled at the beginning of the process from an initial state distribution ρ_0; π is the given policy distribution which samples an action a_t given a state s_t; p is a transition probability distribution (also called the environment dynamics), which samples a next state s_{t+1} and reward r_{t+1} given that action a_t is taken in state s_t;

and T is the length of the interaction trajectory. In general T may be infinite, but for the purposes of this thesis we can assume T is finite. We let

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, a_{T-1}, r_T, s_T) \tag{1.46}$$

denote the trajectory induced by starting in s_0 and then following the policy π and transition distribution p. Such a trajectory is also called an episode.

Formally, a Markov decision process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, p, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R} \subset \mathbb{R}$ is a set of rewards, $p\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, \infty)$ is a transition probability distribution, $\rho_0\colon \mathcal{S} \to [0, \infty)$ is an initial state distribution[3] and $\gamma \in [0, 1]$ is a discount factor. As illustrated in (1.45), taking action a_{t-1} in state s_{t-1} yields an associated immediate reward r_t by sampling from p (it also yields a next state s_t). In many cases the reward r_t is deterministic given s_{t-1}, a_{t-1} and s_t. In such cases it is more common to refer to a *reward function*, e.g. $r\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$. Note that it is not only the immediate reward r_t that reflects the quality of action a_{t-1}, as a_{t-1} indirectly affects *all* future states and actions, and thus rewards. Hence a more central concept is the discounted cumulative reward R_t that follows after a_{t-1}, which is given by:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t. \tag{1.47}$$

It is common to let $\gamma \in [0, 1)$, which means that future rewards have a relatively smaller impact on R_t than more recent ones.

Two additional important concepts within MDPs are the *state value function* V^π and the *state-action value function* Q^π, which are defined recursively as follows for discrete state-action-reward spaces:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{r',s'} p(r', s'|s, a) \left[r' + \gamma V^\pi(s') \right], \tag{1.48}$$

$$Q^\pi(s, a) = \sum_{r',s'} p(r', s'|s, a) \left[r' + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right]. \tag{1.49}$$

The state value function V^π is the expected cumulative reward from state s onwards, assuming one acts according to the policy π. The state-action value function Q^π is seemingly very similar to V^π; the only difference is that Q^π also has the action a taken in state s as an additional input. Specifically, Q^π is the expected cumulative reward from state s onwards, assuming that the first action taken is a and that one thereafter acts according to the policy π. In Section 2.5.2 we will explain how the discounted cumulative reward (1.47), value function (1.48) and/or state-value function (1.49) can be used to learn a policy π that operates desirably in an environment, but let us first return to the example of the

[3]If $\mathcal{S}$ is finite, the range of ρ_0 is $[0, 1]$. If $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are finite, the range of p is $[0, 1]$.

self-driving car to see why a deep neural network can be useful for representing the policy π.

Deep networks as policy distributions. Recall that we are considering a self-driving car which has a single forward mounted RGB camera for observing its surroundings and that it has three discrete actions: drive a step in the left, forward or right direction. Let the immediate reward r_t be deterministic and such that $r_t = 0.1$ if the car moves closer to the goal, $r_t = 10$ if it reaches the goal, and $r_t = -0.1$ otherwise. Finally, let $\gamma = 0.95$. The defined reward and γ imply that the agent receives a higher cumulative reward if it quickly moves from the start to the goal location. It would in principle be possible to omit the $r_t = \pm 0.1$ rewards, since the discount factor $\gamma < 1$ still ensures that the highest cumulative reward is maximized by reaching the goal as fast as possible. In practice it is however common to use a denser reward (in this case keeping the $r_t = \pm 0.1$ rewards), since a sparse reward often makes the learning problem harder.

Given the states, actions and rewards as defined above, it follows that $\mathcal{S}$ is the set of all possible images that the forward mounted camera can observe, $\mathcal{A} = \{\text{left}, \text{forward}, \text{right}\}$, and $\mathcal{R} = \{-0.1, 0.1, 10\}$. Since each state $s \in \mathcal{S}$ is high-dimensional (an image), it makes sense to let the driving policy π be a deep neural network parametrized by $\boldsymbol{\theta}$. The dependence of π on $\boldsymbol{\theta}$ is denoted $\pi_{\boldsymbol{\theta}}$. Similar to how we in Section 2.4 used a softmax layer at the end of the network to produce a distribution over object categories, we can now apply a softmax to produce an action distribution over the three movement actions.

We will next explain how one can learn the parameters $\boldsymbol{\theta}$ of a deep policy network $\pi_{\boldsymbol{\theta}}$ so as to make the policy successful at a given task. This learning process is called policy-based reinforcement learning, since the parameters of the policy are directly updated based on the rewards obtained during the interaction with an environment. A high-level comparison between supervised learning and policy-based reinforcement learning is given in Figure 1.7.

2.5.2 Policy Gradients

Let $\pi_{\boldsymbol{\theta}}$ denote a deep policy network. Our goal in this section is to establish a framework that allows us to tune the parameters $\boldsymbol{\theta}$ of the network such that the policy becomes successful at solving a given task in an environment (where the level of success is measured using a reward function). In doing this, we want to rely as much as possible on the already existing methodology that we described for training neural networks, mainly backpropagation.

If we are given a dataset $\mathcal{D} = \{(s_1, a_1), \dots, (s_N, a_N)\}$ of state-action pairs, where the ground truth action (i.e. the action that is considered most suitable) is given in each pair, then we can train the policy $\pi_{\boldsymbol{\theta}}$ in a standard supervised manner as explained in Section 2.4.3. This is called behavioral cloning. Providing such a dataset $\mathcal{D}$ is possible sometimes, but often it is difficult or impossible to know which is the correct action to take (or there may be multiple candidate actions that appear optimal), so providing a sufficiently large-scale dataset is often very difficult. Because of this, behavioral cloning typically suffers from compounding errors, which is a test time phenomenon by which the agent becomes

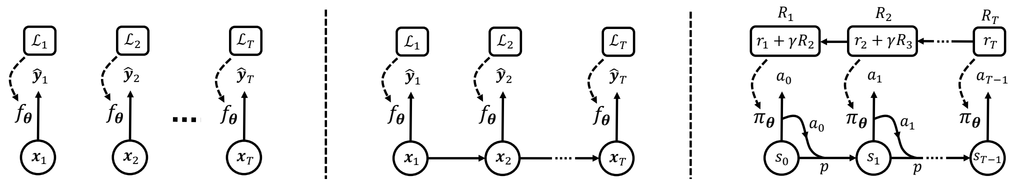


Figure 1.7: Comparisons between non-temporally dependent supervised learning (e.g. image classification), temporally dependent supervised learning (e.g. video classification), and policy-based reinforcement learning. Left and middle: A neural network $f_{\boldsymbol{\theta}}$ with input $\boldsymbol{x}_t$ outputs a prediction $\hat{\boldsymbol{y}}_t = f_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$ with associated loss $\mathcal{L}_t$, which is used to update the parameters $\boldsymbol{\theta}$ via backpropagation. Note that an output prediction $\hat{\boldsymbol{y}}_t$ does not affect the subsequent frame $\boldsymbol{x}_{t+1}$, and the losses can be independently computed. Right: Policy-based reinforcement learning (RL). In RL, the current state s_t depends on both the previous state s_{t-1} and action a_{t-1}. More specifically, s_t is obtained by sampling from the state transition distribution, i.e. $s_t \sim p(*|s_{t-1}, a_{t-1})$. A policy network $\pi_{\boldsymbol{\theta}}$ with input s_t then outputs a distribution from which an action a_t can be sampled as $a_t \sim \pi_{\boldsymbol{\theta}}(*|s_t)$. The concept of a loss function in RL differs significantly from the supervised scenarios in that it is based on a reward function. To obtain the loss at the t:th step, all future rewards must be a available (although there exist methods which can circumvent this requirement). Once the discounted sum of present and future rewards R_t has been computed it is multiplied with $\log \pi_{\boldsymbol{\theta}}$ and the sign is flipped, which provides the loss for updating $\boldsymbol{\theta}$ via backpropagation.

increasingly uncertain about what to do as it experiences states it has not seen during training. This can become a vicious cycle in which the agent traverses further and further away from states it is familiar with, and hence acts increasingly poorly.

Behavioral cloning is perhaps the most simple instantiation of a broader set of techniques which are commonly known as imitation learning. Another line of imitation learning methods (e.g. DAgger [25]) assume that one is given access to an expert, which for each state – not just those that exist in a pre-specified dataset $\mathcal{D}$ – suggests an action to take. Such methods suffer to a much smaller extent from the compounding error issue, since the agent can in principle explore the whole state space and obtain feedback for the actions it tries, and thus the states encountered at test time are more likely to be similar to those seen during training. The exploration can be done for example randomly, or by sampling from the current policy distribution $\pi_{\boldsymbol{\theta}}$.

The issue with this type of approach is that an interactive expert is quite rarely available in practice. Therefore, in policy-based reinforcement learning (RL) one replaces the expert with a reward function, as described earlier. While intuitively a reward function sounds like a reasonable replacement, a new issue arises – the supervised setting is no longer valid, since one does not have access to ground truth actions for each state. However, cumulative rewards can be used as a form of replacement for such ground truths. To arrive at a suitable optimization objective which is based on cumulative rewards, we will first consider what is the density of a given trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, \ldots, a_{T-1}, r_T, s_T)$, cf. (1.46).

This density is given by

$$\rho_0(s_0)\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}}(a_t|s_t)p(r_{t+1},s_{t+1}|s_t,a_t) =: q_{\boldsymbol{\theta}}(\tau). \tag{1.50}$$

Given (1.50) we now define our optimization objective to be the expected cumulative reward when the initial state s_0 is sampled from ρ_0 and assuming one thereafter follows the policy $\pi_{\boldsymbol{\theta}}$ and transition dynamics p, i.e.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{q_{\boldsymbol{\theta}}}\left\{R_0\right\} = \mathbb{E}_{q_{\boldsymbol{\theta}}}\left\{\sum_{t=0}^{T-1}\gamma^t r_{t+1}\right\}. \tag{1.51}$$

Note that maximizing (1.51) only involves tuning the policy parameters $\boldsymbol{\theta}$; one has no control over ρ_0 nor p. To maximize (1.51) we need to find an appropriate expression for $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$, which requires a few tricks.

From an algorithmic standpoint we cannot compute gradients of exact expectations. Therefore we will first perform some manipulations to (1.51) in order to move the gradient operator $\nabla_{\boldsymbol{\theta}}$ inside the expectation, which will allow us to compute sample-based estimates of the gradient. For this we use the log-derivative trick:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{g_{\boldsymbol{\theta}}}\{h(x)\} &= \nabla_{\boldsymbol{\theta}}\int g_{\boldsymbol{\theta}}(x)h(x)\mathrm{d}x = \int \frac{g_{\boldsymbol{\theta}}(x)}{g_{\boldsymbol{\theta}}(x)}\nabla_{\boldsymbol{\theta}}g_{\boldsymbol{\theta}}(x)h(x)\mathrm{d}x \\ &= \int g_{\boldsymbol{\theta}}(x)\nabla_{\boldsymbol{\theta}}\log g_{\boldsymbol{\theta}}(x)h(x)\mathrm{d}x = \mathbb{E}_{g_{\boldsymbol{\theta}}}\{\nabla_{\boldsymbol{\theta}}\log g_{\boldsymbol{\theta}}(x)h(x)\}.\end{aligned} \tag{1.52}$$

Note that in the above we have assumed that $g_{\boldsymbol{\theta}}$ is differentiable whenever it is non-zero. From (1.52) we conclude that

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\theta}}}\left\{R_0\right\} = \mathbb{E}_{q_{\boldsymbol{\theta}}}\left\{\nabla_{\boldsymbol{\theta}}\log q_{\boldsymbol{\theta}}(\tau)R_0\right\}. \tag{1.53}$$

Next, from (1.50) it follows that

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\log q_{\boldsymbol{\theta}}(\tau) &= \nabla_{\boldsymbol{\theta}}\log\left(\rho(s_0)\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}}(a_t|s_t)p(r_{t+1},s_{t+1}|s_t,a_t)\right) \\ &= \nabla_{\boldsymbol{\theta}}\left(\log\rho_0(s_0) + \sum_{t=0}^{T-1}\log\pi_{\boldsymbol{\theta}}(a_t|s_t) + \sum_{t=0}^{T-1}\log p(r_{t+1},s_{t+1}|s_t,a_t)\right) \\ &= \sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t|s_t),\end{aligned} \tag{1.54}$$

where the last equality follows from the fact that neither ρ_0 nor p depend on the parameters $\boldsymbol{\theta}$. Because of this it is common to trade rigor for convenience and write $\mathbb{E}_{q_{\boldsymbol{\theta}}}$ as $\mathbb{E}_{\pi_{\boldsymbol{\theta}}}$ in this context, which we will henceforth do as well. Combining (1.51), (1.53) and (1.54), we get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \left(\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right) \left(\sum_{t=0}^{T-1} \gamma^t r_{t+1} \right) \right\}, \tag{1.55}$$

which is equivalent to

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \left(\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \right) \right\}. \tag{1.56}$$

The inner summation index in (1.56) starts at $t' = 0$, while the policy $\pi_{\boldsymbol{\theta}}$ operating from time step t has no effect on rewards for $t' < t$. Therefore it is more common to use the following gradient expression:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'+1} \right) \right\}. \tag{1.57}$$

Now we are finally ready to express a sample-based estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'+1}^i \right), \tag{1.58}$$

where M denotes the number of trajectories used to estimate the exact gradient (1.57).

As already mentioned in Section 2.4.3, modern deep learning libraries do not require gradient specifications, and in particular the gradient expression (1.58) does not have to be provided in such libraries. Instead one determines the minibatch size $B < M$, runs B trajectories following $\pi_{\boldsymbol{\theta}}$ with the current parameters $\boldsymbol{\theta}$, stores each $\langle s_t^i, a_t^i, R_{t+1}^i \rangle$-tuple (where R_{t+1}^i is computed at the end of the i:th trajectory, as it is not available before), and finally the loss based on the current minibatch is given by

$$\mathcal{L}(\boldsymbol{\theta}) = -J(\boldsymbol{\theta}) \approx -\frac{1}{B} \sum_{i=1}^{B} \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i) R_{t+1}^i. \tag{1.59}$$

The above is a high-level description of REINFORCE [26], which is the main reinforcement learning method used in the thesis.

Improvements upon standard policy gradients. The above derivation led us to the REINFORCE method for estimating the policy gradient. It is intuitive and easy to implement, but it has some drawbacks such as suffering from high variance (the variance typically increases as the lengths of the agent-environment interaction trajectories increase). Another issue is that it may lead to training instability, if large parameter updates that lead to poor subsequent exploration are performed. This can happen because REINFORCE is an on-policy algorithm, which means that the policy we want to optimize is also used for collecting training data. Thus some configurations of the policy parameters may imply that the policy collects poor training data. We however find that REINFORCE works well in most of the settings considered in this thesis (Paper I - IV), which is mainly because the average lengths of the trajectories are relatively short (typically less than about 15 actions).

However, due to the aforementioned issues, REINFORCE is often unsuitable for tasks with long episodes, such as in Paper V where episodes consisting of more than 250 actions are considered. Fortunately, there exist a myriad of other approaches for estimating the policy gradient, many of which result in estimates of lower variance than REINFORCE. In fact, the policy gradient theorem (see [24]) states that

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right\}, \tag{1.60}$$

where $Q^{\pi_{\boldsymbol{\theta}}}$ is the state-action value function defined in (1.49). Furthermore, it can be shown that an action-independent baseline $B(s_t)$ can be subtracted in (1.60). A common choice for the baseline is $B(s_t) = V^{\pi_{\boldsymbol{\theta}}}(s_t)$, where $V^{\pi_{\boldsymbol{\theta}}}$ is the state value function defined in (1.48). One then introduces the advantage function $A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) = Q^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) - V^{\pi_{\boldsymbol{\theta}}}(a_t)$. As the name suggests, it can be interpreted as a measure of how advantageous (or disadvantageous) the action a_t is relative to the expected action from $\pi_{\boldsymbol{\theta}}$ in state s_t. Thus it holds that the following is also an expression for the policy gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) A^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) \right\}. \tag{1.61}$$

While the variance of (1.61) is lower than that of (1.60), there is in practice a higher bias in (1.61) since one has to estimate $V^{\pi_{\boldsymbol{\theta}}}(s_t)$, i.e. $V_{\boldsymbol{\varphi}}(s_t) \approx V^{\pi_{\boldsymbol{\theta}}}(s_t)$. It is common to use a deep neural network for $V_{\boldsymbol{\varphi}}$, whose parameters are often partially shared with $\pi_{\boldsymbol{\theta}}$.

In Paper V we resort to a popular method called *proximal policy optimization* (PPO) [27]. At a high level, the algorithm revolves around a stable, lower-variance approximation of (1.61). The first step is to replace the policy gradient expression (1.61) with[4]

$$J(\boldsymbol{\theta}) \approx \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)} A_{\boldsymbol{\varphi}}(s_t, a_t) \right\} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \eta_{\boldsymbol{\theta}}^t A_{\boldsymbol{\varphi}}^t \right\}, \tag{1.62}$$

[4]We omit the gradient in the expression, since automatic differentiation libraries only require the specification of the objective, not its gradient. Furthermore, we write $A_{\boldsymbol{\varphi}}$ to indicate that we approximate $A^{\pi_{\boldsymbol{\theta}}}$. For details and derivation, see [28].

where $\boldsymbol{\theta}_{\text{old}}$ are the parameters prior to the update, $\eta_{\boldsymbol{\theta}}^t = \pi_{\boldsymbol{\theta}}(a_t|s_t)/\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)$, and $A_{\boldsymbol{\varphi}}^t = A_{\boldsymbol{\varphi}}(s_t, a_t)$. Next, (1.62) is replaced with a clipped surrogate objective:

$$J^{\text{clip}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \min\left(\eta_{\boldsymbol{\theta}}^t A_{\boldsymbol{\theta}}^t, \text{clip}\left(\eta_{\boldsymbol{\theta}}^t, 1-\varepsilon, 1+\varepsilon\right) A_{\boldsymbol{\varphi}}^t\right) \right\}, \tag{1.63}$$

where $\varepsilon > 0$ is a hyperparameter, and $\text{clip}(a, b, c) = \min(\max(a, b), c)$. Thus the clip-function bounds the first argument between the latter two arguments (assuming $b \leq c$). At a high level, due to the outer min-operation and the inner clip-function, the expression (1.63) ensures that the parameter updates are not too drastic, which makes the learning process more stable.

3 Visual Perception Tasks Studied in this Thesis

In the thesis we develop active methods for three different visual perception tasks; see Figure 1.1 for an overview. This section contains a high-level description of each task.

3.1 Object Detection

Object detection is a core computer vision task which is intuitively simple to understand (but convoluted to evaluate, as we will see in Section 3.1.1). See Figure 1.8 for a few examples of object detection. Given a set $\mathcal{C}$ of C object categories (e.g. 'cat', 'chair', 'person'), including a background category, the goal is to both localize and classify in a given image each object instance among the object categories. Here localizing an object refers to drawing a tight bounding box that contains the object (no boxes should be drawn around things that are considered background), and classification refers to predicting for the content of the bounding box a label among those in the set $\mathcal{C}$.

For an object instance to be considered correctly detected, two things must hold simultaneously: i) it must be correctly classified, and ii) the predicted bounding box must overlap sufficiently with the ground truth bounding box. As for ii), this overlap is measured by the *intersection-over-union* (IoU) between the predicted bounding box and the ground truth bounding box. As the name suggests, the IoU is defined as the ratio between the area of the intersection and union of the two bounding boxes, respectively; see the two left-most columns of Figure 1.9 for a conceptual explanation. The IoU ranges from 0 to 1 and a common threshold for what is considered sufficient overlap is 0.5. Refer e.g. to the surveys [29, 30, 31] for more in-depth coverages of the object detection task, including explanations of several standard methodologies.

3.1.1 Evaluation Metrics

The standard evaluation protocol for object detectors revolves around computing on a test set of images the *average precision* (AP) for each non-background class *independently*, then

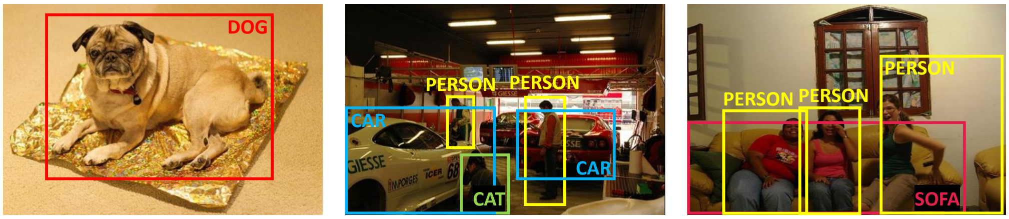


Figure 1.8: Three examples of object detection. Left: The image contains only one object instance (the blanket is considered background here), which is clearly and fully visible. The bounding box quite tightly encloses the object and the predicted label 'dog' is correct, so this is a successful example. Middle: Three persons and three cars in a garage. All six objects are viewed in fairly poor lightning, from a challenging perspective, and several of the objects are only partially visible, so this is quite a difficult example. Two cars (blue) and two persons (yellow) are correctly detected. One of the persons (green) is correctly localized, but the label 'cat' is incorrect. Finally, one of the cars is not detected at all. Right: Three persons and two sofas (one of which is barely visible). This example is somewhat challenging since the lightning is quite poor. Two persons and one sofa are correctly detected. The third person obtains a correct label, but the bounding box is too large, so it is an incorrect detection. Finally, the barely visible sofa is not detected at all, but such difficult object instances are often ignored when evaluating object detectors. The images were obtained from [1], while the rectangles and label text were drawn by the author of this thesis.

averaging the individual AP-scores to obtain the mean average precision (mAP). In this section we will therefore consider how the AP for a given object category with index $c \in \{1, \ldots, C-1\}$ is computed (we assume that the last index C corresponds to background).

Algorithm 1 contains details of how the AP is computed and Figure 1.9 shows a few conceptual examples on single images. One begins by extracting for each detection in each image the c:th probability score of the C-dimensional probability vector associated with the detection. This probability score is called the confidence of the detection. Given the confidences it is common to perform two preprocessing steps before the AP-computation (i.e. before line 1 of Algorithm 1). Note that these two steps are not part of the evaluation protocol, so they should be included in the overall detection system.

The first preprocessing step is called non-maximum suppression (NMS) and is used to filter among highly overlapping detection bounding boxes.[5] Specifically, one iterates over each detection in descending order of confidence, and each other detection which overlaps sufficiently (typically IoU at least 0.5) is discarded. If NMS is not performed there is typically an increased risk that multiple detections overlap sufficiently with the same ground truth instance, which reduces AP. After NMS, one also discards detections for which the confidence is below a threshold (this is the second preprocessing step). The reason for this is that each object instance of a category indexed by anything other than c is considered background during the AP-computation of the c:th category, and thus one

[5]Typical object detectors propose several bounding boxes at various locations and of different scales and aspect ratios (e.g. using a sliding windows approach), each of which is classified as an object or background.

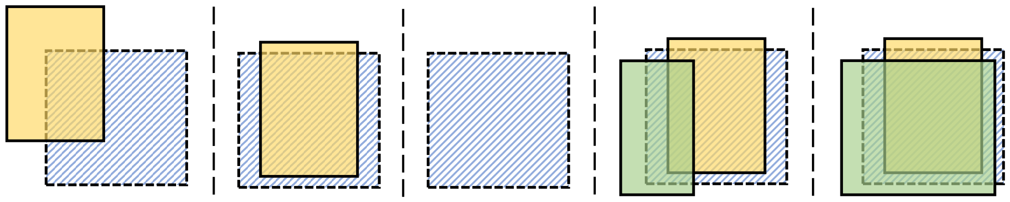


Figure 1.9: Conceptual explanations of intersection-over-union (IoU), precision, recall, and average precision (AP), assuming a single ground truth (GT) instance in the dataset. We also assume that the IoU-threshold that determines if a GT is sufficiently overlapped is set to 0.5. Dashed and solid rectangle borders indicate GT and detection boxes, respectively. See Algorithm 1 for details of how AP is computed. Column 1: IoU < 0.5, so the detection is a false positive (FP). The extended recall and precision envelope arrays are both equal to $[0, 0]$, so AP $= 0$. Column 2: IoU ≥ 0.5, so the detection is a true positive (TP). The extended recall and precision envelope arrays are equal to $[0, 1]$ and $[1, 1]$, respectively, so AP $= 1$. Column 3: There are no detections, so AP $= 0$. Column 4: There are two detections, but only one has IoU ≥ 0.5 with the GT, so we have one TP and one FP. Assuming that the TP has the highest confidence, the extended recall and precision envelope arrays are $[0, 1, 1]$ and $[1, 1, 0.5]$, respectively, and hence AP $= (1 - 0) \cdot 1 + (1 - 1) \cdot 0.5 = 1$. If instead the FP has the highest confidence, the extended recall and precision envelope arrays are $[0, 0, 1]$ and $[0.5, 0.5, 0.5]$, respectively, and hence AP $= 0.5$. Column 5: There are two detections, both with IoU ≥ 0.5 with the GT. However, only the one with highest confidence is a TP; the other one is an FP. The extended recall and precision envelope arrays are $[0, 1, 1]$ and $[1, 1, 0.5]$, respectively, so AP $= 1$. This surprising result is mainly related to the simplified setting considered here – in real scenarios with many images, detections and ground truths, duplicate detections typically reduce AP.

wants to discard detections of other object categories (providing detections of background reduces AP). The collection of the M detections that remain after the preprocessing steps is subsequently sorted in descending order of confidence.

The main procedure of the AP-computation is then performed (see lines 4 - 16 in Algorithm 1). This process determines for each detection if it is a true positive (TP) or false positive (FP). To do this for a given detection, one computes the IoU between the detection and all ground truth bounding boxes (corresponding to the c:th category) in the image associated with the detection. If the image contains at least one ground truth with which the IoU exceeds the IoU-threshold, then the ground truth with which the IoU is highest is inspected. If this ground truth has not yet been detected (i.e. not sufficiently overlapped) by a detection with higher confidence, then the current detection is considered a TP and otherwise it is an FP (thus each ground truth is only allowed to be detected once). If on the other hand there is no ground truth with which the IoU exceeds the IoU-threshold, or if the image has no ground truth at all, then the detection is considered an FP.

Once the above procedure is complete we know for each detection if it is a TP or FP. This knowledge is represented by two binary arrays of length M. The i:th detection is a TP if the i:th entry of the TP-array is equal to 1 and otherwise it is an FP. Given these arrays, one computes the recall and precision arrays according to lines 17 - 19 of Algorithm 1.

Algorithm 1 Procedural code for computing average precision (AP) for a given object category in the object detection task (array indexing begins at 1).

```
1: Input $[0,1]$-bounded IoU threshold iou_thresh, list GTs of all $K$ ground truth
   bounding boxes in the dataset, and list BBs of all $M$ detection bounding boxes sorted
   in descending order of confidence for the given object category.
2: Initialize false and true positive arrays fps and tps of length $M$ as all-zeros.
3: Initialize boolean array GT_dets of length $K$ as all-zeros.
4: for $d = 1, 2, \ldots, M$ do
5:     Compute IoUs between BBs[d] and all ground truth bounding boxes in the im-
       age associated with BBs[d]. Let iou_max denote the maximum such IoU, and
       iou_idx be the argmax (indexing relative to the whole of GTs). If the image
       associated with BBs[d] has no ground truths, set iou_max = -1.
6:     if iou_max $\geq$ iou_thresh then
7:         if GT_dets[iou_idx] == 0 then
8:             tps[d] = 1
9:             GT_dets[iou_idx] = 1
10:        else
11:            fps[d] = 1
12:        end if
13:    else
14:        fps[d] = 1
15:    end if
16: end for
17: Cumulatively sum up fps and tps: fps = cumsum(fps), tps = cumsum(tps).
18: Compute extended recall array rec = [0, tps / K].
19: Compute extended precision array prec = [0, tps / (tps + fps)].
20: Initialize precision envelope array prec_env = prec.
21: for $i = M + 1, M, \ldots, 2$ do
22:     prec_env[i - 1] = max(prec_env[i - 1], prec_env[i])
23: end for
24: Initialize AP = 0.
25: for $i = 2, 3, \ldots, M + 1$ do
26:     AP = AP + (rec[i] - rec[i - 1]) * prec_env[i]
27: end for
28: return AP
```

Recall measures the fraction of ground truths that are detected, while precision measures the fraction of detections that are true positives.

Intuitively, as more detections are provided, more ground truths are detected on average (assuming the detections are at different locations and of different scales and aspect

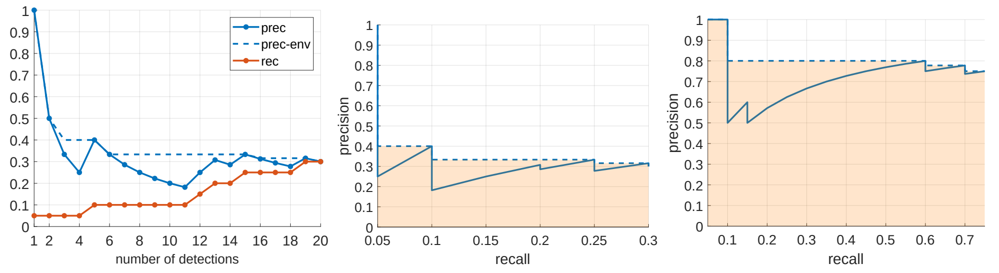


Figure 1.10: Left: Precision and recall versus number of detections in an artificial setting. The true positive (TP) array of length 20, corresponding to 20 imagined detection boxes, was manually generated by setting 6 of the 20 values to 1 and the rest to 0. The false positive (FP) array thus consists of 14 ones and 6 zeros. Given the TP- and FP-arrays, lines 17 - 23 of Algorithm 1 were executed, assuming the dataset has 20 ground truth (GT) instances of the considered class. The first entry of the respective arrays were discarded, as they correspond to the extensions on lines 18 - 19. The non-decreasing red recall curve begins at 0.05, corresponding to 1 of 20 GTs being detected by the first detection, and reaches a maximum value of 0.3 (6 of 20 GTs detected in total). Similarly, the solid blue precision curve begins at 1 and decreases until the fifth detection (since the second, third and fourth detections are FPs). At 20 detections the precision is 0.3, since $6/(6+14) = 0.3$. Finally, the dashed curve is the precision envelope, which is a tight non-increasing version of the precision curve and is used to compute AP (see middle figure). Middle: Precision plotted against recall for the setup described in the left figure. The AP is given by the shaded area under the precision envelope (dashed curve). In this case the AP is 0.14. Right: Precision plotted against recall in a case where 15 of 20 entries of the TP-array are equal to 1. As expected, AP increases significantly as the number of TPs increases relative to the number of FPs. The AP is 0.62 in this example.

ratios within the images) and thus the values of the recall array are non-decreasing. Conversely, the entries of the precision array are typically (but not everywhere) non-increasing. Specifically, each TP increases precision and each FP decreases it. An intuitive motivation for why the precision array typically has a non-increasing trend is that for a well-performing object detector, the detections with higher confidences are more likely to correspond to true positives, compared to detections with lower confidences. Now recall that once a ground truth is marked as detected, any detections of it which have lower or equal confidence are considered false positives, and thus when the number of detections increases, the risk of including false positives also increases.

After the recall and precision arrays have been obtained one computes the precision envelope array, which is the tightest non-increasing array whose entries are equal to or higher than the corresponding entries in the original precision array. Finally, the AP is computed according to lines 24 - 27 of Algorithm 1. Figure 1.10 shows examples of precision, recall and average precision. The AP is given by the area under the precision envelope in a precision-recall plot (i.e. the areas of the shaded regions in the middle and right columns of Figure 1.10).

In summary, AP can be interpreted as an overall assessment of how a detector performs under different requirements on the recall of ground truth instances. Again, the stricter the requirement regarding recall is, the lower the precision typically gets, and vice versa. This is why one computes an average at different recall levels. It should be noted that AP and mAP (the mean of the AP-values over the $C - 1$ non-background categories) are typically multiplied with a factor 100 in the end, so that the values are reported in the range $[0, 100]$.

3.2 Human Pose Estimation

Human pose estimation refers to recognizing the 2d or 3d articulated poses of people given an image, or a set of images captured from multiple viewpoints, of the people. This means predicting for each person the 2d or 3d locations of a pre-defined set of body joints (e.g. head, neck, shoulders, knees, ankles), as well as grouping together which joints belong to the same person. Body joints are sometimes referred to as parts or keypoints. The grouping of joints refers to assigning to each estimated joint a unique identifier – this can be thought of as 'person 1', 'person 2', and so on. Based on all joints with the same identifier, connections are drawn according to a pre-defined set set of rules (e.g. connecting a head to a neck, or a knee to an ankle); see the blue lines between joints in Figure 1.11. A connection between two joints is called a limb and the full set of limbs is often referred to as a (pose) skeleton. Note that an incorrect joint identifier may induce a highly inaccurate pose skeleton, as conceptually illustrated in the bottom-left of Figure 1.11.

We next briefly describe the single-view 2d and 3d pose estimation tasks and thereafter explain the corresponding multi-view settings (which are studied in Paper III - IV of this thesis). For more extensive overviews, see for example the recent surveys [32, 33, 34]. The survey [34] specifically covers the 3d human pose estimation task.

3.2.1 Estimating Human Poses from a Single Image

In this section we consider the case where we are given a single image of one or several people for which we wish to estimate the 2d or 3d poses, see Figure 1.11. In 2d pose estimation the task is to localize in the image each *visible*[6] joint among a given set of joints. Meanwhile, 3d pose estimation requires the localization in 3d space of *every* joint, even those that are not visible in the image. The 3d space in which the poses are to be reconstructed is induced by the camera matrix $\boldsymbol{P}$ (cf. Section 2.1.1) corresponding to the image. More specifically, once the 3d human poses have been estimated in the camera coordinate system, they can be rotated and translated to the global coordinate system in which they can be compared with ground truth poses (see Section 3.2.3). Note that, as mentioned above, both the 2d and 3d human pose estimation tasks also require correct groupings of joints into pose skeletons (this step is trivial if one assumes that the image contains a single person, however).

The 3d pose estimation task is in general more difficult than the 2d counterpart for a few key reasons, one of which is that the full 3d pose is to be estimated even when the person

[6]A joint is considered visible also if it is on the other side of the body and no other body parts occlude the joint, e.g. a neck is considered visible even if only the throat is observed from the front.

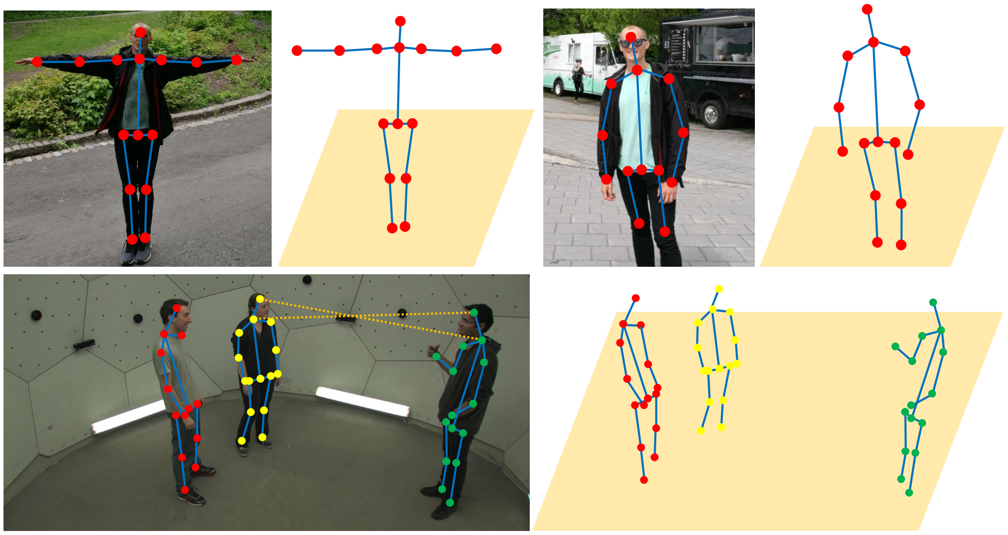


Figure 1.11: Top: Single-view, single-person examples of 2d and 3d human pose estimation. Columns 1 and 3: In 2d pose estimation the task is to localize each *visible* joint. Non-visibility of joints can result from obstacles, people (others or self-induced), or if a person is not entirely visible in the image. Column 2 and 4: In 3d pose estimation the task is to localize in 3d space *every* joint, even those that are non-visible in the image. The ground planes do not have to be estimated, but are added here for visualization purposes. Columns 2 and 4 correspond to the images in columns 1 and 3, respectively. Bottom: Single-view, multi-person example of 2d and 3d human pose estimation. Note the self-occlusion of one the arms of the man to the left (the occluded joints are nevertheless estimated in 3d). In the multi-person setting there exist several possible joint pairs, so this task further involves grouping joints into correct pairs. This has been indicated by using three different colors for the joints, one for each person. For illustration, a dashed line has also been drawn between the head of the woman in the middle and the neck of the man to the right, and vice versa. This shows how the pose estimate may become highly inaccurate if any joint grouping fails. The images on the top row depict the author of this thesis, while the image in the bottom row was obtained from [2]. All drawings were made by the author of this thesis.

is partially non-visible in the image. Another reason is that 3d estimation is inherently ill-posed in that it suffers from depth ambiguities – given a single image captured with a pinhole camera (cf. Section 2.1.1), there are infinitely many 3d configurations which result in the same 2d projection, since all points on the same ray from the camera center project to the same point in the image plane. For this reason the 3d estimation task is sometimes formulated in a simpler way, where providing pose estimates up to translation is sufficient (during training and evaluation, the pose estimates are simply translated so that they are centered around their respective ground truths). It should be noted that even if the translation of the 3d body pose is accurate, there may still be multiple 3d configurations which result in the same 2d projection. A further issue with 3d human pose estimation is that it is much harder to acquire ground truth annotations than in the 2d case, especially

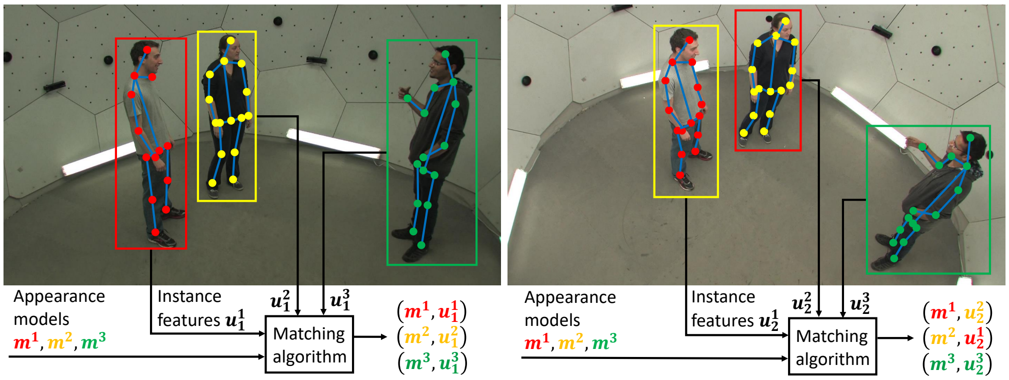


Figure 1.12: Accurately matching people across viewpoints is crucial for successful multi-view 3d human pose estimation. In this thesis, deep instance features $\boldsymbol{u}_i^1, \ldots, \boldsymbol{u}_i^{n_i}$ are computed for each of the n_i detected persons in the i:th view. Each $\boldsymbol{u}_i^j$ is then compared to a set of given appearance models $\boldsymbol{m}^1, \ldots, \boldsymbol{m}^l$ for the l persons in the scene. Note that l may differ from n_i, as some persons may be non-visible in a view, or there may be additional incorrect detections. Next, the L2-distance is computed between each pair, which yields a cost matrix that specifies each pairwise assignment cost. Given this matrix, the Hungarian algorithm is used to produce the assignments. In the left view all three people are correctly matched to their respective appearance models. In the right view only one person is correctly matched to its appearance model. Hence two of the resulting 3d poses will likely become inaccurate, while the third person may obtain an accurate 3d pose estimate. Images obtained from [2]. All drawings were made by the author of this thesis.

in the wild. For this reason, many of the most popular datasets currently used are obtained from in-lab motion capture systems such as [35, 2].

3.2.2 Estimating Human Poses from Multiple Images

Given access to several cameras with known camera matrices – or a moving camera observer such as a drone – it is possible to combine 2d or 3d pose estimates from several viewpoints, either to improve upon individual 3d estimates (Paper IV), or to transform 2d estimates to 3d estimates via triangulation (Paper III). In both cases, being able to accurately match people across viewpoints is crucial to enable the integration of pose estimates from different viewpoints; see Figure 1.12.

The triangulation problem is explained in Figure 1.13. In this thesis we use a straightforward linear triangulation method (see details on p. 312 in [4]), which for each pair of images containing a given joint can estimate the corresponding 3d joint when the camera matrices are known. We then median-average over the 3d points observed from each pair to produce the final 3d pose estimates. An alternative would be to jointly solve an over-determined linear system, but such an approach is more sensitive to outliers.

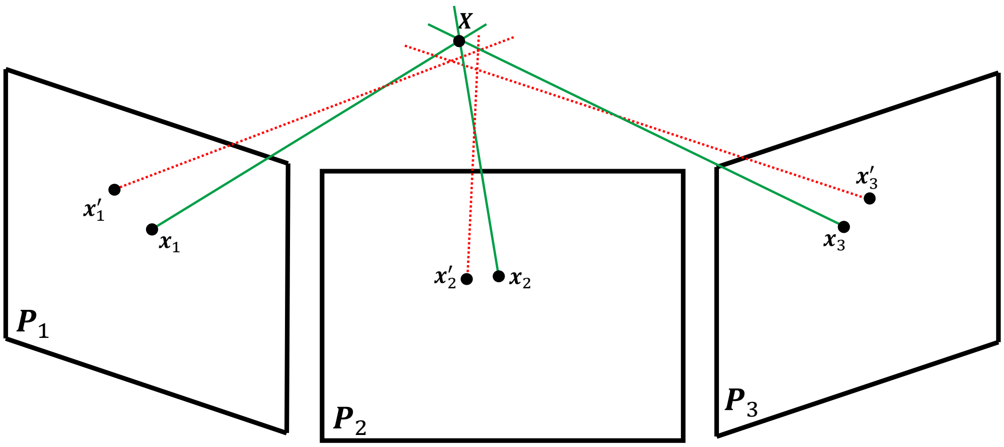


Figure 1.13: Conceptual illustration of the triangulation problem. Recall from Figure 1.2 that in an ideal setting, given a point in an image taken with a known camera matrix, the line on which the corresponding 3d point lies can be recovered. Thus, in an ideal setting, given a pair of image points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in two different images captured with known camera matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, it is possible to find the corresponding 3d point $\boldsymbol{X}$ as the intersection between two lines in 3d space (green lines). Note that a corresponding point $\boldsymbol{x}_3$ in a third image captured with the known camera matrix $\boldsymbol{P}_3$ would also generate such a line that intersects the others in $\boldsymbol{X}$. In practice however, due to noise, distortions etcetera, the point $\boldsymbol{X}$ is not captured in the images as $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$, but instead as the displaced projections $\boldsymbol{x}_1'$, $\boldsymbol{x}_2'$ and $\boldsymbol{x}_3'$. The associated 3d lines are unlikely to intersect (red dotted lines). Triangulation refers to the process of recovering an estimate $\hat{\boldsymbol{X}}$ of the true 3d point $\boldsymbol{X}$, given image points $\boldsymbol{x}_1', \ldots, \boldsymbol{x}_m'$ captured with known camera matrices $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_m$ $(m \geq 2)$.

3.2.3 Evaluation Metrics

There exist a range of evaluation metrics for the 2d and 3d human pose estimation tasks; see e.g. [32] for an overview of several of these. Below we list those metrics that are relevant for the papers in the latter part of this thesis. Note that we exclusively develop and evaluate 3d estimation approaches and therefore only explain the evaluation of 3d pose estimates.

The main error metric used is the *mean per-joint position error* (MPJPE), which for a single image of m (partially) visible people is computed as

$$\text{MPJPE} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left\| \hat{\boldsymbol{X}}_i^j - \boldsymbol{X}_i^j \right\|_2, \tag{1.64}$$

where n is the number of joints, $\boldsymbol{X}_i^j$ is the j:th ground truth 3d joint of the i:th person, and $\hat{\boldsymbol{X}}_i^j$ is the estimate of the j:th 3d joint of the i:th person. The assignment of pose estimates to the respective persons is done by computing the distances between each pose estimate and ground truth pose, followed by the Hungarian algorithm [36] to find the closest matches (i.e. the pairs of pose estimates and ground truths which jointly yield the lowest average error). If there are fewer pose estimates than ground truths, this can be penalized[7] e.g. by assigning all-zeros estimates to the corresponding persons (this will likely increase the MPJPE). Similarly, if there are fewer ground truths than pose estimates, the additional pose estimates can be penalized[6] e.g. by assigning them to an artificial ground truth consisting of all-zeros.

Another evaluation metric is the *mean reprojection error* (MRE) of 3d poses. For a single image, this error is computed by projecting the 3d poses onto the image (using the camera matrix $\boldsymbol{P}$ associated with the image), then computing the pixel displacements of the estimated joints relative to a set of reference joints (either 2d estimates or ground truths), and finally averaging over the visible joints. If $\hat{\boldsymbol{x}}_i^j \in \mathbb{R}^2$ denotes the projection of the j:th joint of the i:th person's 3d pose estimate $\hat{\boldsymbol{X}}_i$ onto the image with given 2d reference joint $\boldsymbol{x}_i^j \in \mathbb{R}^2$, then the MRE is given by

$$\text{MRE} = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n} \left\| \boldsymbol{v}_i^j \odot \left(\hat{\boldsymbol{x}}_i^j - \boldsymbol{x}_i^j \right) \right\|_2, \tag{1.65}$$

where m is the number of people for which 3d poses are reprojected, n_i is the number of visible reprojected joints of the i:th person, and $\boldsymbol{v}_i^j \in \{0,1\}^2$ is a binary array where both entries are equal to 1 if the j:th joint of the i:th person is visible in the image, and where both entries are otherwise equal to 0. Note that (1.65) depends on the image size; a doubling of the size will double the error. However, this can in practice be remedied by normalizing the MRE with the image size.

3.3 Semantic Segmentation

Semantic segmentation is a fundamental problem in computer vision, which shares some similarities with the object detection task described in Section 3.1. Given a set $\mathcal{C}$ of C semantic categories, which may include a generic background category, the task is to predict a label for every pixel in a given image. In semantic segmentation separate labels are often provided for non-objects such as 'wall' and 'floor', whereas these would be labeled simply as 'background' in the object detection task. On the one hand, semantic segmentation is more challenging than object detection, since it involves the prediction of free-form boundaries around arbitrary objects and surfaces, as opposed to merely proposing rectangular boxes. On the other hand, the semantic segmentation task does not involve separating different instances of the same object category, which is required in object detection.

The output of a segmentation system is typically a tensor of size $H \times W \times C$, where H and W denote respectively the image height and width. Each spatial location (i, j) of this tensor contains a predicted probability distribution over the semantic classes present at pixel (i, j) in the image. The channels of the tensor are subsequently transformed into

[7]Another possiblity is to consider a different evaluation metric which more explicitly takes into account the effects of too few or too many estimates, such as percentage of correct keypoints (PCK).

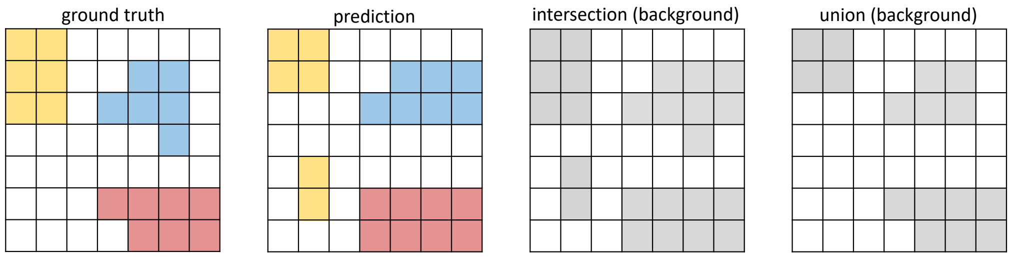


Figure 1.14: Conceptual illustration of semantic segmentation, including evaluation. Column 1: Ground truth (GT) segmentation mask, where each grid cell corresponds to a pixel in the associated image. There are three semantic categories present in the image; their pixel segmentations are shown in yellow, blue and red, respectively. White represents the background category. Column 2: Predicted segmentation mask. None of the three regions of the GT are perfectly predicted and there is a fourth yellow region that is incorrectly predicted. The accuracy is the average number of correctly classified pixels. Comparing the prediction and GT from top to bottom gives accuracy $(7+6+4+6+6+6+6)/49 \approx 0.84$. The mIoU is the mean IoU over the four categories and is equal to $(4/8+5/8+7/8+25/33)/4 \approx 0.69$ (order: yellow, blue, red, white). Note that in practice the accuracy and mIoU are computed over a dataset of images, not only a single image. Columns 3 - 4: The intersection and union, respectively, of background pixels in the GT and prediction masks. White and gray correspond to background and non-background pixels, respectively.

binary masks, such that the c:th channel contains the predicted mask of the c:th object category in the image. Specifically, to obtain the c:th binary mask the (i, j):th entry is set to 1 precisely if c is the index of the largest value in the probability distribution over the classes at pixel (i, j). For more details about semantic segmentation and an overview of several proposed methodologies, refer e.g. to the recent survey [37].

3.3.1 Evaluation Metrics

The two main metrics used to evaluate semantic segmentation performance are *accuracy* and *mIoU*, see Figure 1.14. The accuracy is the average number of correctly classified pixels in the dataset. While conceptually simple to understand, the accuracy metric has a drawback in that it favors the correct classification of larger regions (such as walls and floors) over smaller ones. This issue is remedied by the mIoU score, which is the mean IoU over all categories (this may or may not include background). For the c:th object category, the IoU is computed by averaging the intersections-over-unions between predicted and ground truth masks over all images in the dataset.

4 Summary of Contributions and Ideas for Future Work

The contributions of this thesis can be coarsely divided into three categories. In this section we summarize these contributions, state assumptions and potential limitations, and outline avenues for future work. It should be emphasized that while each contribution is associated with a distinct visual perception task, they are not attached specifically to those in principle – in each case, the core ideas and methodologies could be transferred to other domains.

In the first part (Paper I - II) we develop reinforcement learning-based search policies for object detection, which are partly inspired by the 'saccade-and-fixate' visual processing routine of many biological systems, including humans. Our proposed visual search policies sequentially inspect different parts of an input image, integrate information as they proceed, and automatically terminate search once they consider the image to have been sufficiently explored. This approach is considerably different from many established, typical detection methods, where i) the search strategy is exhaustive, fixed by design, and independent of the image content; ii) the detector response function is not harmonized or refined in connection with the search strategy; and iii) responses are processed independently without context accumulation as computation proceeds. While Paper I introduces the core ideas – including a search strategy which is guided by previously explored image regions, and a mechanism for adaptively determining the length of each search trajectory – Paper II extends and improves upon this formulation in every respect. For example, the drl-RPN model we propose in this paper can be adapted to a range of speed-accuracy trade-offs during inference (and is generally faster than the model in the first paper) and shows accuracy improvements over the popular Faster R-CNN detector [38].

Even though drl-RPN replaces the model introduced in Paper I, there are still ways in which the ideas can be further refined and extended. The main limitation of the current implementation is that it is somewhat slower than many established methods (e.g. Faster R-CNN), since a larger number of image regions are inspected on average during the detection process. More specifically, while drl-RPN provides a spatially compact set of image regions at each fixated image location, these sets typically contain a large amount of regions which correspond to various scales and aspect ratios of potential detections. The runtime of the model could thus be improved by attaching a mechanism for reducing the number of regions associated with each fixation. An ad-hoc mechanism for doing this is already explored in the paper, but a more sophisticated approach would be to add a module which predicts the approximate scales and aspect ratios of all objects in the vicinity of each fixated image location. This would allow the model to omit regions associated with unlikely scales and aspect ratios. Alternatively (or orthogonally), the processing speed could be increased by modifying the architecture of drl-RPN so that more computation is shared among the evaluated regions (see e.g. [39]).

It would also be interesting to attach drl-RPN to a moving observer, which would then have the capability to not only decide *which* images to inspect, but also *where* to look in each image. Ideally, the two types of decisions within this hierarchy should also be able to guide one another, since this could potentially result in improved performance and/or efficiency on the considered task. For example, the visual search process within the current viewpoint could potentially be performed in fewer fixations if the context from neighboring viewpoints is taken into account (e.g. the former views may have informed the agent that it is in a kitchen, which would imply that its current view is unlikely to contain a sofa).

In the second part of the thesis (Paper III - IV) we shift focus from within-image to between-image exploration. Specifically, this part examines and develops methods that actively integrate information from multiple viewpoints in order to estimate the 3d articulated poses of humans (the number of people does not need to be known a priori). This differs significantly from most prior works in human pose estimation, which either work with a single view or exhaustively scan a range of viewpoints to produce pose estimates. The proposed reinforcement learning-based methods outperform comparable heuristic alternatives significantly in task accuracy despite inspecting fewer views on average, yet compared to the heuristic strategies they add only a negligible amount of processing time for the selection of each viewpoint. To the best of our knowledge we were the first to introduce the concept of active 3d human pose estimation.

A drawback of the current formulations is that they assume a scene can be synchronously observed from a large set of viewpoints. This works given carefully constructed in-lab setups wherein our active pose estimation systems can be applied for example to perform pose and (offline) motion analysis, but a multi-camera pipeline is unlikely to be available in most practical scenarios. It would however be possible to adapt our framework to operate with a much smaller set of viewpoints, assuming those viewpoints are dynamic. This can be achieved for example by mounting cameras to a few drones. One could even consider working with a single moving observer, but this would require equipping the active pose estimation systems with forward temporal models which estimate future poses for multiple subsequent time steps, corresponding to future viewpoints. Otherwise the multi-view pose fusion is unlikely to yield reliable estimates, unless the people in a scene move only marginally during the time it takes for the observer to move to a new viewpoint. It would also be interesting to explore the active human pose estimation framework in a more free-form setting (e.g. performing pose estimation using an embodied agent), instead of the sphere-constrained setup we have considered so far.

The last part (Paper V) differs significantly from the previous ones in that it considers the problem of how to efficiently train (or refine) a given perception system – here a semantic segmentation network – as opposed to focusing or positioning a given pre-trained perception model. Some elements of this are present also in Paper II, where the detector head of drl-RPN is refined jointly with the search policy, but this is not the main contribution of that paper. In contrast, the formulation in Paper V is centered around the question of where an embodied agent should navigate in a 3d environment, and for which views to request annotations along its path, to optimally refine an underlying perception module. Several methods are proposed and we show that a model based on reinforcement learning is able to gather more informative training data compared to heuristic counterparts.

The main bottleneck of the reinforcement learning-based agent is its current training procedure, which is slow since it relies on the continual refinement of a segmentation network in each episode, and further assumes the availability of ground truth semantic segmentation masks for each viewpoint. Both issues are connected to the reward signal by which the the agent's annotated viewpoints are assessed by measuring the accuracy (or mIoU) of the segmentation network before and after each annotated image is obtained, which requires continual refinements within each episode. These problems could thus be circumvented by modifying the reward signal such that it evaluates the annotated views in a different way. For example, computing some form of diversity metric for the gathered training set could potentially be used as a proxy to estimate how useful those images are when training the segmentation system (in this case, a large diversity among annotated views would be associated with a high reward).

Another avenue for feature work would be to examine the embodied visual active learning framework in a real robotics context. By default this would however require a human-in-the-loop who provides segmentation masks for the images requested by the robot, which is likely to be impractical in most cases. To avoid this, one could replace the human annotator with a state-of-the-art semantic segmentation system for real images (assuming that the segmentation system associated with the robot is bounded from above by that other segmentation system in terms of maximum accuracy), and use the predictions of that system as pseudo ground truth.

To summarize, in this thesis we develop active methods for three different visual perception tasks. For the first two our focus is mainly on reducing computational costs and/or improving perception accuracy, by actively selecting where to focus a pre-trained perception system (from which viewpoints to observe a scene, and where to look in an image, respectively). In the last setup we study how an agent can actively explore a scene to refine a given perception model while querying only a limited amount of annotation. In each context considered in this thesis, the visual perception model which is to be favorably used or improved – be it in terms of active sensor positioning or perception refinement – is part of the agent's state space. Thus the proposed active visual perception systems are in principle able to adapt to any inherent limits or weaknesses of their underlying visual perception models. We empirically show for each task that active visual perception methods trained with reinforcement learning match or outperform established or heuristic approaches in task accuracy, typically while using less or a negligible amount of additional computation.

4.1 Overview of Scientific Papers

This section contains a brief description of the five papers that constitute the latter part of this thesis. For each paper a summary of contributions is provided.

Paper I: S. Mathe, A. Pirinen, C. Sminchisescu, "Reinforcement Learning for Visual Object Detection", *Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, 2016.

This paper proposes a reinforcement learning-based method for object detection that is partly inspired by the 'saccade-and-fixate' visual search routine of humans. At a high level, our detection agent is given an image with a dense set of candidate region proposals and the agent's goal is to detect an instance of a given object category while examining (evaluating) as few of these regions as possible. The agent achieves this by sequentially attending different spatial locations in the image, where at each location only a relatively small set of regions is investigated. The state space of the agent includes a history of earlier observed image regions, which is used as a contextual cue for where to look next in the image. The agent is additionally equipped with an action that lets it automatically terminate the detection process. Our empirical results show that the proposed method yields only a small drop in detection accuracy, while being up to two orders of magnitudes faster than sliding windows-based methods.

Author contributions: SM and CS conceived the project. SM developed most of the methodology, including all implementation, while AP and CS came with ideas and suggestions for improvements. AP made some corrections to the mathematical derivation of the model. SM did the experiments, which were independently validated by AP. SM and CS wrote most of the paper, while AP took some part in the paper writing.

Paper II: A. Pirinen, C. Sminchisescu, "Deep Reinforcement Learning of Region Proposal Networks for Object Detection", *Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, USA, 2018.

In this paper we propose drl-RPN, a deep reinforcement learning-based visual recognition model which consists of a sequential region proposal network (RPN) and an object detector. The drl-RPN model is conceptually inspired by that in Paper I, but extends and improves it along several dimensions. Specifically, different from the method in Paper I, i) it seamlessly handles multiple object categories and instances per image; ii) it is entirely deep and as such does not rely on hand-engineered feature design; iii) it jointly learns and refines the search policy and detector parameters; iv) the search process can be adapted to a range of exploration-accuracy trade-offs during inference; and v) it yields significantly more accurate detection results and is much faster than the model in the first paper.

Author contributions: AP and CS conceived the project. AP developed the methodology with ideas and improvement suggestions from CS. AP implemented all code and performed all experiments. AP wrote most of the paper with input from CS, while some parts were written jointly with CS.

Paper III: A. Pirinen, E. Gärtner, C. Sminchisescu, "Domes to Drones: Self-Supervised Active Triangulation for 3D Human Pose Reconstruction", *Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, 2019.

This paper introduces ACTOR, a self-supervised deep reinforcement learning agent for active triangulation of 2d human body joint detections into 3d pose estimates. ACTOR operates on a spherical camera rig with a dense set of viewpoints, which can be seen as a proxy for a moving observer such as a drone. Based on a random initial viewpoint of a scene containing a variable number of people, the agent should sequentially inspect views across the sphere and observe the people such that all body joints are detected from at least two different viewpoints (the minimum requirement for performing triangulation into 3d). The agent should achieve this while inspecting as few viewpoints as possible to avoid excessive computation. Once done with the active triangulation in the current time step of the multi-camera video stream, the agent proceeds to the next time step and repeats the process. Temporal backups of previously triangulated body joints can further be used by the model for those joints (if any) which are not adequately triangulated in the current time step. We compare ACTOR to heuristic multi-view baselines and show that it produces significantly more accurate 3d pose estimates using fewer viewpoints on average, yet it only negligibly increases the time required per viewpoint selection.

Author contributions: AP, EG and CS conceived the project. AP and EG developed the methodology with input from CS. AP and EG implemented all code and performed all experiments together. AP wrote most of the paper together with EG and got feedback from CS, while CS wrote some parts. Overall, AP and EG contributed equally to this paper.

Paper IV: E. Gärtner, A. Pirinen, C. Sminchisescu, "Deep Reinforcement Learning for Active Human Pose Estimation", *Association for the Advancement of Artificial Intelligence (AAAI)*, New York, USA, 2020.

The model proposed in this paper, Pose-DRL, shares similarities with ACTOR from Paper III in that it is also a deep reinforcement learning-based active observer operating on a camera dome. However, Pose-DRL differs from ACTOR in several respects. First, it is equipped with a 3d monocular pose estimator which allows it to predict even occluded joints. In particular, this enables Pose-DRL to predict 3d poses at each temporal step given only a single view, which potentially makes it more adaptable to settings with a single but moving observer. Second, Pose-DRL has an explicit mechanism which allows it to fully automatically terminate viewpoint selection; it may even terminate after observing a single view if it believes the view is good enough (ACTOR requires at least two views to perform triangulation). Third, due to significantly different designs of reward functions, Pose-DRL faces different challenges during viewpoint selection – while ACTOR requires each joint to be visible in at least two different views, Pose-DRL must take into account which viewpoints fuse into accurate 3d estimates when combined. As we demonstrate in the paper, this often involves terminating viewpoint selection early to avoid views where the pose estimator is inaccurate.

We develop two variants of Pose-DRL, one which simultaneously estimates the 3d poses of all the variably many people in a scene, and one which focuses the active pose estimation on a given target person (the target may nevertheless be present among other people who then act as potential occlusions). Our empirical evaluations show that Pose-DRL significantly outperforms heuristic multi-view baselines – even when observing fewer viewpoints on average – while requiring virtually the same processing time per viewpoint selection.

Author contributions: CS came up with the core idea for this project, which AP refined together with EG. AP and EG developed the methodology with input from CS. AP and EG implemented all code and performed all experiments together. AP wrote most of the paper together with EG and got feedback from CS, while CS wrote some parts. Overall, AP and EG contributed equally to this paper.

Paper V: D. Nilsson, A. Pirinen, E. Gärtner, C. Sminchisescu, "Embodied Visual Active Learning for Semantic Segmentation", *Association for the Advancement of Artificial Intelligence (AAAI)*, Virtual conference, 2021.

Common for Paper I - IV is that they focus on the problem of positioning a sensor to an image part or viewpoint where a given perception model is accurate and/or to avoid redundant processing with that perception model. This paper instead studies the question of which sensor viewpoints of an environment that annotations should be queried for, so as to maximally improve the average accuracy of a given perception model for all viewpoints in that environment. Specifically, we introduce the embodied visual active learning task, where an agent should explore a 3d environment to improve its visual perception by actively deciding for which views to request annotation. We propose and implement a variety of methods, both learnt and pre-specified ones, in the context of semantic segmentation. The agents aim to gather informative annotated views, use motion to explore and propagate annotations in the neighborhood of those views, and improve the underlying segmentation network by online re-training. The learning-based method uses reinforcement learning with a reward function that balances two competing objectives, where on the one hand task accuracy should be high (which requires environment exploration) and on the other hand the amount of annotated data should be as small as possible (since data annotation is expensive). We extensively evaluate our proposed models in photorealistic 3d environments and show that the learning-based model outperforms comparable pre-specified ones, even when it requests fewer annotations on average.

Author contributions: DN and CS conceived the project. DN implemented the largest part of the code base. AP and EG also implemented significant parts of the code. DN, AP and EG continually proposed ideas and refined the methodologies over the course of the project, with input and further ideas from CS. DN performed most of the experiments, while AP and EG performed some of the experiments. DN, AP and EG jointly wrote most of the paper with feedback from CS, while CS wrote some parts.

References

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[2] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *CVPR*, 2015.

[3] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision*, 2017.

[4] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, 1995.

[6] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.

[10] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[11] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *PAMI*, vol. 35, no. 1, 2012.

[12] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *CVPR*, 2017.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[15] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *NeurIPS*, 2015.

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, 2010.

[17] A. Beck, *First-order methods in optimization*. SIAM, 2017.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[25] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Aistats*, 2011.

[26] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, 1992.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[28] S. M. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, 2002.

[29] 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 *CVPR*, 2017.

[30] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[31] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *IJCV*, vol. 128, no. 2, 2020.

[32] C. Zheng, W. Wu, T. Yang, S. Zhu, C. Chen, R. Liu, J. Shen, N. Kehtarnavaz, and M. Shah, "Deep learning-based human pose estimation: A survey," *arXiv preprint arXiv:2012.13392*, 2020.

[33] Y. Chen, Y. Tian, and M. He, "Monocular human pose estimation: A survey of deep learning-based methods," *Computer Vision and Image Understanding*, vol. 192, 2020.

[34] N. Sarafianos, B. Boteanu, B. Ionescu, and I. A. Kakadiaris, "3d human pose estimation: A review of the literature and analysis of covariates," *Computer Vision and Image Understanding*, vol. 152, 2016.

[35] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *PAMI*, vol. 36, no. 7, 2014.

[36] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, 1955.

[37] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *PAMI*, 2021.

[38] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015.

[39] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *NeurIPS*, 2016.

Chapter 2

Scientific Publications

Paper I

Reinforcement Learning for Visual Object Detection

Stefan Mathe[2,3] **Aleksis Pirinen**[1] **Cristian Sminchisescu**[1,2]
[1]Centre for Mathematical Sciences, Lund University
[2]Institute of Mathematics of the Romanian Academy
[3]Department of Computer Science, University of Toronto

Abstract

One of the most widely used strategies for visual object detection is based on exhaustive spatial hypothesis search. While methods like sliding windows have been successful and effective for many years, they are still brute-force, independent of the image content and the visual category being searched. In this paper we present principled sequential models that accumulate evidence collected at a small set of image locations in order to detect visual objects effectively. By formulating sequential search as reinforcement learning of the search policy (*including* the stopping condition), our fully trainable model can explicitly balance for each class, specifically, the conflicting goals of *exploration* – sampling more image regions for better *accuracy* –, and *exploitation* – stopping the search *efficiently* when sufficiently confident about the target's location. The methodology is general and applicable to any detector response function. We report encouraging results on the PASCAL VOC 2012 object detection test set, showing that the proposed methodology achieves almost *two orders of magnitude speedup* over sliding window methods.

1 Introduction

Classically, detection has been formulated as the problem of maximizing a confidence function over a set of hypothesized target locations, where the confidence can be learned in a fully supervised [1] or weakly supervised [2] setup. In the sliding window formulation, the hypothesis set consists of a large set of rectangular windows, and the maximization problem is solved by exhaustive search. Since this process is generally too expensive in practice, many methods have been proposed to accelerate it, from methodologies that leverage properties of the confidence function, to proposal methods or cascade techniques. All these methods retain the exhaustive search property over the hypothesis space, aiming either to reduce the number of hypotheses to start with, or search these efficiently.

In contrast, biological systems have a pattern of search that can be characterized as 'saccade-and-fixate' [3], where a small set of scene locations are investigated sequentially, in order to accumulate sufficient evidence on the target location. Set aside efficiency (only a few regions of an image are explored) and biological plausibility, it appears still interesting to formally derive mathematical models that could optimally balance efficiency and accuracy, by integrating evidence, sequentially, in a principled way. The challenge is to be able to operate with delayed rewards, which rules out supervision at each step. At the same time, one wants to avoid having to completely pre-specify the environment, which for visual scenes would be impossible – given the complexity of images and visual object categories, the models should be effectively trained.

By formulating sequential search as reinforcement learning of the category and the image-dependent search policy, *including* the stopping conditions, in this work we develop fully trainable methods that can explicitly balance the conflicting goals of *exploration* – sampling more image regions for better *accuracy* –, and *exploitation* – stopping the search *efficiently* when sufficiently confident in the target's location. The methodology is general, applicable to any detector response function, and can learn search strategies and stopping conditions that are image and visual category specific. Two orders of magnitude speedups over sliding window methods are achieved in the challenging PASCAL VOC 2012 object detection benchmark.

2 Related Work

One class of efficient detectors focuses on the use of branch-and-bound heuristics [4] to prioritize exploration of the search space towards promising image regions. Unlike the present work, such techniques are only applicable to confidence function classes for which strong bounds are available. Additionally, in the absence of the target in the image, methods in this class degenerate to exhaustive search. Motivated by these limitations, other authors have proposed to use cascades of classifiers [5, 6, 7, 8] to progressively narrow the search space, where weak but fast classifiers are applied early to eliminate regions unlikely to contain the target, while investing computational resources to run more complex classifiers on promising regions. Such methods drastically reduce the computation cost, but classifiers early in the cascade still have to be applied exhaustively over all image regions. Instead of focusing on region exploration strategies, others have sought to optimize the evaluation of the confidence function. This includes sharing computation among neighboring image regions [9, 10] or among the different classifiers for multi-class detection problems [11].

Recent trends in object detection focus on a rapid content-based reduction of the set of candidates – in earlier methods, windows, cropped at different positions, and of different aspect ratios, in an image – to a smaller set (still thousands of hypotheses in most methods) which exhibits the statistical regularities of the objects found in the real world. Typical methodologies include parametric figure-ground segmentation with Gestalt, 'object-like' filtering [12], superpixels [13, 14] or edge-based cues [15]. In this work we will rely on the parametric segmentation method of Carreira et al. [12] to generate a set of free-form figure-ground proposals that capture most objects of interest, although our method can use any other state-of-the-art proposal generation method [13, 14, 15].

In contrast to methods based on branch and bound and cascades of classifiers, sequential search methods like [16] attempt to sparsely sample the image through a local search guided by the contextual relations among regions, previously shown to improve detection accuracy [17, 18, 19, 20]. Gonzalez-Garcia et al. [16] propose search policies that map contextual windows to the ground truth target location based on random forests, whereas [21, 22] learn a mapping from images to bounding box masks using a cascaded deep learning model. Palleta et al. [23] and Butko and Movellan [24] developed remarkable early sequential models based on POMDPs for recognition and face detection. However, those models are not fully trainable and require a complete and accurate specification of the environment, which makes them challenging to apply in complex multi-class visual detection setups. More recently, reinforcement learning [25] has been applied to visual analysis problems like image classification [26, 27, 28], face detection [29], tracking and recognizing objects in video [30], learning a sequential policy for RGB-D semantic segmentation [31], and scanpath prediction [32].

In independent work performed in parallel with ours [33], [34] also focus on object detection using reinforcement Q-learning. We differ, among others, in using policy search based on an analytic gradient computation with continuous as opposed to discrete reward (both in a supervised and weakly-supervised image labeling setup [33]), by operating on regions instead of deforming bounding boxes, in using different actions (infinite set via function approximation vs. 9 discrete actions in [34]), a different state representation (a set of 10 boxes in [34] vs. our manipulation of disjoint sets), and in the training procedure based on reinforcement learning with delayed rewards as opposed to an additional apprenticeship signal in [34]. This results in a different model behavior in both training and testing, as [34] requires the control of actions via short steps in order to prevent the apprenticeship learning process to immediately locate the target from any position. In testing [34] use 10 steps to locate the target, whereas our model takes 3.1 steps on average.

Relevant to our work is also the one of Karayev et al. [35] who differently however, focus on object detection in an anytime recognition framework where a multi-class detector can be stopped, asynchronously, during its execution. Karayev et al. sequentially schedule multi-class models, optimizing the order of applying sliding window object detectors (exhaustively evaluated at all image locations, in a cascade), stopping short of running detectors for some classes. In contrast, we spatially optimize each specific sequential class detector (stopping short of searching all image locations) and run the detectors for all classes in the standard way. Methodologically, there are significant differences: [35] use Q-learning and regress expected value of (state, action), we do policy search with analytic gradient to directly optimize expected reward. We have infinite action spaces (any image location), [35] operate over finite actions (1+#detector-classes in [35], or 1+#feature-types in [27, 36]); [35] can stop anytime, whereas we learn a stopping condition for each class. From a system viewpoint the methods are complementary, as one can benefit both from an efficient ordering of class detectors [35] and from efficient individual class detectors, as we propose, but we will not investigate this here.

3 Problem Formulation

Given an input image, we formulate object detection as the problem of maximizing a confidence function $f_c : R \to \mathbb{R}$ over a set of image regions R:

$$\boldsymbol{r}^* = \arg\max_{\boldsymbol{r} \in R} f_c(\boldsymbol{r}). \tag{2.1}$$

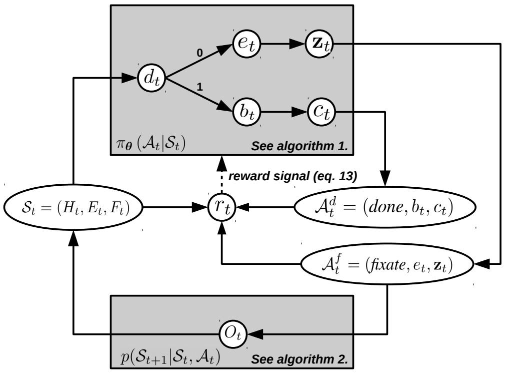
The set of image regions R can be defined either at the coarse level of bounding boxes or at the finer level of free-form image regions obtained with a state-of-the-art proposal generation method [12, 13, 14]. Good choices for the confidence function f_c that achieve state-of-the-art performance are associated with a high computational price tag. Therefore, solving the optimization problem (2.1) can still be expensive even for the comparatively smaller (versus e.g. bounding boxes) set of region proposals R obtained by a segmentation algorithm. To address this issue, in Section 3.1 we present a model to learn efficient search strategies, rigorously formulated in a reinforcement learning setup. Our model operates in an integrate, fixate and evaluate regime, and only explores a few locations before deciding on the presence of a target.

3.1 Sequential Detection Model

In this section we present the key components of our optimal sequential model for image exploration.[1] Our model is given a set of image regions R indexed by the set $B = \{1, \dots, |R|\}$ (with $|\cdot|$ the set cardinality), the confidence function as introduced in (2.1), $f_c(\boldsymbol{r}) = \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r})$ with parameters $\boldsymbol{\theta}_c$, and a feature extractor $\boldsymbol{q} : R \to \mathbb{R}^m$ of dimensionality m. The objective of the model is to locate the target with a minimal number of evaluations of these two computationally expensive functions.

[1]See the supplementary material for detailed derivations.

At each time step t during a detection sequence (except the last step), our model generates a *fixate* action $\mathcal{A}_t^f$ based on its internal state $\mathcal{S}_t$. Each fixation action specifies a location in the image that the model decides to explore and results in a set of observations O_t, which is a set of image regions in the proximity of this location. The observed regions are the only ones that are inspected by the algorithm. In particular, they are the only regions on which the confidence function f_c and feature extractor $\boldsymbol{q}$ need to be evaluated. The observations O_t are then used to update the state $\mathcal{S}_t$, summarizing all past observations and actions.



0
1
d_t
e_t
$\mathbf{z}_t$
b_t
c_t
$\pi_{\theta}(\mathcal{A}_t|\mathcal{S}_t)$
See algorithm 1.
reward signal (eq. 13)
$\mathcal{S}_t = (H_t, E_t, F_t)$
r_t
$\mathcal{A}_t^d = (done, b_t, c_t)$
$\mathcal{A}_t^f = (fixate, e_t, \mathbf{z}_t)$
O_t
$p(\mathcal{S}_{t+1}|\mathcal{S}_t, \mathcal{A}_t)$
See algorithm 2.

Figure 2.1: Sequential detector based on reinforcement learning. At each time step, the model may terminate search ($d_t = 1$) based on the history H_t of observed regions (Algorithm 1) and produce a detection hypothesis b_t with confidence c_t, receiving a reward measuring the detection quality. Otherwise, an evidence region e_t is chosen from the set $H_t \setminus E_t$ of unselected regions and used to predict the next fixation location $\boldsymbol{z}_t$. The set O_t of all regions in the neighborhood of $\boldsymbol{z}_t$ become observed (Algorithm 2) and a negative reward is received, reflecting the computational cost of extracting features for these regions.

When enough information has been collected about the image, the model issues a special *done* action, indicating that it has decided on the location of the detection target. The *done* action is associated with a detection target bounding box b_t and confidence c_t. The model has a set of trainable parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_c, \boldsymbol{\theta}_d, \boldsymbol{\theta}_e, \boldsymbol{\theta}_p, \boldsymbol{\Sigma}_p, \sigma_c)$ controlling, respectively, the detector response confidence, the stopping criteria, the informativeness of an image region with respect to the target location, the image location of the most probable next fixation and its variance, and the variance of the confidence c_t associated to the model output.

Each *fixate* action may reduce the uncertainty in localizing the detection target, but is associated with a computational cost due to the need to integrate the set of observations O_t into the state. The goal of our model is to balance the conflicting needs of information gathering (*fixate* actions) with the need to correctly locate the target (*done* actions).

3.2 Model Structure

We now proceed to describe in detail the actions, states, observations, and decision process of our model. The model components are shown in Figure 2.1 and several examples of search patterns are illustrated in Figure 2.2.

States. The state of our model is represented as a tuple with three elements: the observed region history H_t, the selected evidence region history E_t, and the fixation history F_t. This tuple $\mathcal{S}_t = (H_t, E_t, F_t)$ summarizes the history of observations and actions since the be-

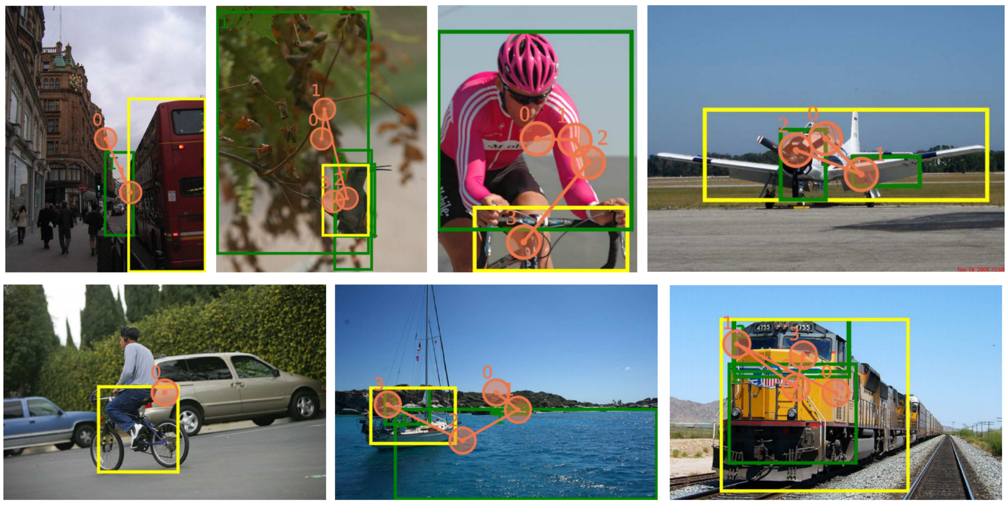


Figure 2.2: Sequences of fixation locations z_t (orange circles) generated by our model, together with the corresponding evidence regions e_t (green boxes), and the final detected bounding box b_t (yellow), for several images. The model may terminate the search early, if the target is found by the first central fixation (first image in the second row). When the target has not yet been found, regions that do not contain it are often exploited to guide the search to new promising locations (e.g. the street provides the context for finding the bus). When a small target lies inside a wider region (e.g. the bird in the tree), the model uses the wider region as a contextual cue to find the target, in a coarse-to-fine fashion. Similarly, fine-to-coarse search strategies involving several exploratory fixations are used to provide the foveal coverage needed to observe large targets (e.g. the airplane). See Table 2.1 for quantitative results and Section 4 for discussion.

ginning of the search sequence.

The observed region history H_t: At each time step t, the model keeps track of a history $H_t \subseteq B$ of image regions observed so far. The confidence function f_c is evaluated on these regions alone and is used to decide when to terminate the search. The history H_t is also used by the model to decide on promising locations to fixate during the next step, as these might provide context to guide the search.

The selected evidence region history E_t: The model decides on the next location to fixate based on an evidence region $e_t \in H_t$ from the observation history. This evidence region is deemed by the model to provide the necessary context that is indicative of the target's location. However, to encourage diversity during search, each region should be used as evidence at most once. For this reason, the model keeps track of the set $E_t \subseteq H_t$ of regions selected so far, and evidence regions are always selected from the set $H_t \setminus E_t$.

The fixation history F_t: The set of observed regions at each time step t depends on the history F_t of past fixation locations, cf. (2.2). We thus include this history in the state $\mathcal{S}_t$.

Actions. Actions in our model are represented as tuples. There are two kinds of actions, distinguished by their first element, which can be one of two discrete symbols: *fixate* or *done*. *Fixate* actions are represented as a three element tuple $\mathcal{A}_t^f = (\textit{fixate}, e_t, \boldsymbol{z}_t)$, where $e_t \in B$ represents the index of the evidence region and $\boldsymbol{z}_t \in \mathbb{R}^2$ is the image coordinate of the next fixation. *Done* actions are represented as $\mathcal{A}_t^d = (\textit{done}, b_t, c_t)$ where $b_t \in B$ is the index of the region representing the detection output and $c_t \in \mathbb{R}$ represents the detection confidence. To summarize, the action space of our model consists of the union of all *fixate* and *done* tuples, i.e. $\mathcal{A} = \mathcal{A}^f \cup \mathcal{A}^d$.

Observations. Following a *fixate* action, the set O_t of image regions in the neighborhood of the fixation location $\boldsymbol{z}_t$ become observed. To define this neighborhood, we use a circular area of radius T_R around the fixation center $\boldsymbol{z}_t$. We say that a pixel is fixated at time t if it falls within the area associated with $\boldsymbol{z}_t$. In order for a region $\boldsymbol{r}$ to become observed at time t, a sufficiently large fraction $h(\boldsymbol{r})$ of its pixels must have been fixated during the current or previous steps:

$$h(\boldsymbol{r}) = \frac{|\{\boldsymbol{x} \in \boldsymbol{r} | \exists \boldsymbol{z} \in F_t, \|\boldsymbol{z} - \boldsymbol{x}\|_2 \leq T_R\}|}{|\boldsymbol{r}|}, \tag{2.2}$$

$$O_t = \{i \in B \mid h(\boldsymbol{r}_i) \geq T_F\}, \tag{2.3}$$

where $F_t = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t\}$ is the history of locations fixated by the model up to time step t, and T_F is a threshold that controls the minimum fraction of fixated pixels in an observed segment.

3.3 Stochastic Policy

The model decides on the next action to take based on the current state. Its stochastic decision policy $\pi_{\boldsymbol{\theta}}(\mathcal{S}_t, \mathcal{A}_t)$ proceeds in three phases, each having its own set of learned parameters. The model first evaluates whether to *terminate search* (termination decision). If positive, a *done* action is performed, else a *fixate* action follows. We will review each of these next.

Termination decision. The model may decide to terminate search at any given time step, based on the current state $\mathcal{S}_t$, and produce a detection result. Rather than using an ad-hoc termination policy, e.g. a preset number of fixations (search locations), our model uses a learnt decision function that balances detection confidence against computational load:

- *Detection confidence:* If the model has already observed a region which is deemed to contain the detection target with high confidence, it may decide to terminate the search early. To capture this aspect we compute the maximum confidence over the regions observed so far, i.e. $\operatorname{asmax}\left(\{f_c(\boldsymbol{r}_i)\}_{i \in H_t}\right)$, where $\operatorname{asmax}(X) = \frac{\sum_{x \in X} x e^{\alpha x}}{\sum_{x \in X} e^{\alpha x}}$ for any set X and smoothness hyperparameter α.

- *Computational load:* The running cost of our detector has two components: first, the number of confidence function evaluations performed so far, which is proportional to the ratio $|H_t| \,/\, |R|$ of regions observed at the current time step t; second, the number of search policy evaluations. Since the policy is evaluated once per time step, this cost is proportional to the number of time steps t.

In order to allow the model to balance these termination criteria, we define a four-element feature vector for the current state:

$$\boldsymbol{v}\left(\mathcal{S}_t\right) = \begin{bmatrix} \operatorname{asmax}\left(\{f_c\left(\boldsymbol{r}_i\right)\}_{i \in H_t}\right) & t & \dfrac{|H_t|}{|R|} & 1 \end{bmatrix}^\top . \tag{2.4}$$

The search termination probability (*done* action) is given by a logistic classifier with parameters $\boldsymbol{\theta}_d$:

$$p_{\boldsymbol{\theta}}(d_t = 1 | \mathcal{S}_t) = \operatorname{sigm}\left[\boldsymbol{\theta}_d^\top \boldsymbol{v}\left(\mathcal{S}_t\right)\right], \tag{2.5}$$

where d_t is an binary variable indicating the decision to terminate the search at the current time step and $\operatorname{sigm}(x) = \left(1 + e^{-x}\right)^{-1}$ is the sigmoid function.

Done action. Upon termination ($d_t = 1$), the model outputs a bounding box b_t from the set H_t of observed regions, to represent the detection target location, together with a confidence score c_t. We use a soft maximum bounding box selection criterion, with smoothness hyperparameter α:

$$p_{\boldsymbol{\theta}}(b_t = k | d_t = 1, \mathcal{S}_t) = \frac{e^{\alpha f_c(\boldsymbol{r}_k)}}{\sum_{i \in H_t} e^{\alpha f_c(\boldsymbol{r}_i)}}. \tag{2.6}$$

The corresponding confidence c_t is normally distributed around the confidence for the selected bounding box, i.e.

$$p_{\boldsymbol{\theta}}(c_t | d_t = 1, b_t = k, \mathcal{S}_t) = N(c_t | f_c(\boldsymbol{r}_k), \sigma_c). \tag{2.7}$$

where $\sigma_c \in \mathbb{R}$ is a model parameter that controls the variance of the confidence of the predictions. Finally, the probability of a *done* action is given by

$$\pi_{\boldsymbol{\theta}}\left(\mathcal{A}_t = (\text{done}, b_t, c_t) \,|\mathcal{S}_t\right) = p_{\boldsymbol{\theta}}\left(d_t = 1 | \mathcal{S}_t\right) p_{\boldsymbol{\theta}}(b_t | d_t = 1, \mathcal{S}_t) p_{\boldsymbol{\theta}}(c_t | d_t = 1, b_t, \mathcal{S}_t). \tag{2.8}$$

Fixate action. If the search is not terminated ($d_t = 0$), the model selects a new evidence region $e_t \in H_t \setminus E_t$ from the set of observed regions, that it deems informative for the target location. We define an evidence function $f_e : B \to \mathbb{R}$, $f_e\left(i\right) = \exp\left[\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)\right]$ that evaluates the informativeness of image region i with respect to the target location, where $\boldsymbol{\theta}_e$ are learned model parameters. We pick the region e_t from a multinomial distribution defined by the evidence function over the set $H_t \setminus E_t$ of image regions not selected during previous steps:

$$p_{\boldsymbol{\theta}}(e_t|\mathcal{S}_t) = \frac{f_e(e_t)}{\sum_{i \in H_t \setminus E_t} f_e(i)}. \tag{2.9}$$

Once selected, the evidence region e_t is used to define a Gaussian probability distribution for the next fixation location $\boldsymbol{z}_t \in \mathbb{R}^2$. For convenience, let us denote by

$$\boldsymbol{\mu}(e_t) = \frac{\boldsymbol{x}_1(e_t) + \boldsymbol{x}_2(e_t)}{2} \tag{2.10}$$

the center of the bounding box tightly enclosing the evidence region $\boldsymbol{r}_{e_t}$, defined by its top-left and bottom-right corners $\boldsymbol{x}_1(e_t)$ and $\boldsymbol{x}_2(e_t)$, respectively. Similarly, let

$$\boldsymbol{\Delta}(e_t) = \operatorname{diag}\left(\frac{\boldsymbol{x}_1(e_t) - \boldsymbol{x}_2(e_t)}{2}\right) \tag{2.11}$$

be the diagonal matrix encoding half the width and height of this bounding box. Then, the probability for the next fixation location $\boldsymbol{z}_t$ is given by

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\mathcal{S}_t, e_t) = N\left(\boldsymbol{z}_t|\boldsymbol{f}_p(e_t), \boldsymbol{\Delta}(e_t)^\top \boldsymbol{\Sigma}_p \boldsymbol{\Delta}(e_t)\right), \tag{2.12}$$

where $\boldsymbol{\Sigma}_p$ is a learned covariance matrix that controls the spread of fixations, and the Gaussian center $\boldsymbol{f}_p(e_t)$ is based on a linear combination of the evidence region features $\boldsymbol{q}(\boldsymbol{r}_{e_t})$ with learned parameters $\boldsymbol{\theta}_p$:

$$\boldsymbol{f}_p(e_t) = \boldsymbol{\Delta}(e_t)\boldsymbol{\theta}_p^\top \boldsymbol{q}(e_t) + \boldsymbol{\mu}(e_t). \tag{2.13}$$

We make the position function f_p invariant to the scale of the image region $\boldsymbol{r}_{e_t}$ by normalizing with respect to its bounding box size, defined by the top-left and bottom-right corners, cf. first term in (2.13), and relative to the bounding box center (second term in (2.13)). Summarizing, the probability of a *fixate* action is given by

$$\pi_{\boldsymbol{\theta}}\left(\mathcal{A}_t = (\text{fixate}, e_t, \boldsymbol{z}_t)\,|\mathcal{S}_t\right) = p_{\boldsymbol{\theta}}\left(d_t = 0|\mathcal{S}_t\right) p_{\boldsymbol{\theta}}\left(e_t|d_t = 0, \mathcal{S}_t\right) p_{\boldsymbol{\theta}}\left(\boldsymbol{z}_t|d_t = 0, e_t, \mathcal{S}_t\right). \tag{2.14}$$

The model policy is completely specified by equations (2.8) and (2.14), which define a probability distribution over all possible actions $\mathcal{A}_t$. Note that out policy is highly (deeply) non-linear in the features and the parameters. The stochastic policy is given by a Gaussian distribution on top of highly non-linear predictions (in contrast, note that methodologies like [29, 34] are deterministic).

Algorithm 1 Policy sampling algorithm

```
1:  procedure SAMPLE ($\mathcal{S}_t = (H_t, E_t, F_t)$)
2:      $d_t \sim p(d_t|\mathcal{S}_t)$ using (2.4), (2.5)
3:      if $d_t = 1$ then
4:          $b_t \sim p(b_t|\mathcal{S}_t, d_t)$ using (2.6).
5:          $c_t \sim p(c_t|\mathcal{S}_t, d_t, b_t)$ using (2.7)
6:          return $\mathcal{A}_t = (\text{done}, b_t, c_t)$
7:      else
8:          $e_t \sim p(e_t|\mathcal{S}_t, d_t)$ using (2.9)
9:          $\boldsymbol{z}_t \sim p(\boldsymbol{z}_t|\mathcal{S}_t, d_t, e_t)$ using (2.12)
10:         return $\mathcal{A}_t = (\text{fixate}, e_t, \boldsymbol{z}_t)$
11:     end if
12: end procedure
```

Algorithm 2 State transition algorithm

```
1:  procedure OBSERVE ($\mathcal{S}_t = (H_t, E_t, F_t)$ , $\mathcal{A}_t = (\text{fixate}, e_t, \boldsymbol{z}_t)$)
2:      $O_t \leftarrow \{i \in B \mid h(\boldsymbol{r}_i) \geq T_F\}$
3:      $H_{t+1} \leftarrow H_t \cup O_t$
4:      $E_{t+1} \leftarrow E_t \cup \{e_t\}$
5:      $F_{t+1} \leftarrow F_t \cup \{\boldsymbol{z}_t\}$
6:      return $\mathcal{S}_{t+1} = (H_{t+1}, E_{t+1}, F_{t+1})$
7:  end procedure
```

3.4 Inference and Learning

Inference is carried out by repeated sampling of the policy $\pi_{\boldsymbol{\theta}}(\mathcal{A}_t|\mathcal{S}_t)$, until a *done* action is achieved (Algorithm 1). At each step the state $\mathcal{S}_t$ is updated according to the action $\mathcal{A}_t$ (Algorithm 2). When the search is finished, the region b_t and the confidence c_t are generated and returned as the detector output.

For learning we are given a set of images, represented as sets of regions B_j, together with confidence function f_c aimed to be maximal at target locations. For notational simplicity, without loss of generality, we will consider the equations for one image, containing n (possibly 0) detection targets, and the corresponding ground truth regions $\{\boldsymbol{g}_i\}_{i=1}^n$.

We wish to find the model parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_c, \boldsymbol{\theta}_d, \boldsymbol{\theta}_e, \boldsymbol{\theta}_p, \boldsymbol{\Sigma}_p, \sigma_c)$ maximizing the target detection accuracy based on the detected target location b_t and confidence c_t at the last step (when $d_t = 1$). At the same time, we aim to minimize the number of region evaluations. To capture this trade-off, and to avoid explicitly instructing the model how to achieve it, we formulate the training objective as a delayed reward, as typical in a reinforcement learning setup. Our reward function is sensitive to the detection location and the confidence at the final state, and incurs a penalty for each region evaluation:

$$r_t\left(\mathcal{S}_t, \mathcal{A}_t\right) = \begin{cases} -\beta \cdot \left|O_t \setminus H_t\right| & \text{if } d_t = 0 \\ \operatorname{sigm}\left(c_t\right) \cdot \left[\max_{i=1,n} \operatorname{IoU}\left(\boldsymbol{g}_i, \boldsymbol{r}_{b_t}\right)\right] & \text{if } d_t = 1 \wedge n > 0 \\ -\operatorname{sigm}\left(c_t\right) & \text{if } d_t = 1 \wedge n = 0, \end{cases} \tag{2.15}$$

where $\operatorname{IoU}(\cdot,\cdot)$ is the intersection-over-union function on regions and β is a penalty paid by the model for each confidence function evaluation.We found it straightforward to estimate the exploitation-exploration trade-off parameters, for each class detector, based on cross-validation. Typical values are e.g. $\beta = 10^{-3}$ and $\alpha = 30$ (cf. (2.6)).

The first branch in (2.15) associates a negative reward to each *fixate* action, proportional to the computational cost of evaluating the newly observed region set $O_t \setminus H_t$. The last two branches correspond to the *done* action, with different rewards for images in which the target is present and absent. In the former case (branch 2), the model receives a reward that is proportional to its confidence and the ground truth overlap. In the latter case (branch 3), the location is ignored, and the model receives a higher reward if its confidence is smaller. Concluding, the reward function defined in (2.15) balances detection accuracy and computational complexity.

In training, we maximize the expected reward function on the training set, defined as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{s})}\left[\sum_{t=1}^{|\boldsymbol{s}|} r_t\right] - \frac{\lambda}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta}, \tag{2.16}$$

where $\boldsymbol{s} = ((\mathcal{S}_0, \mathcal{A}_0), \ldots, (\mathcal{S}_k, \mathcal{A}_k), \ldots)$ represents a variable length sequence of states,[2] sampled by running the model (Algorithm 1 and 2), starting from an initial state $\mathcal{S}_0 = (H_0, E_0, F_0)$ and λ is an L2-regularizer. We set the initial H_0 to the set of segments observed by fixating the image center, and both E_0 and F_0 to $\emptyset$.

For one image, the gradient of the expected reward (2.16) is approximated as [37, 25]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{|\boldsymbol{s}^i|}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathcal{A}_t^i|\mathcal{S}_t^i)\left[\sum_{t=1}^{|\boldsymbol{s}^i|} r_t^i\right] + \lambda\boldsymbol{\theta}, \tag{2.17}$$

where $\boldsymbol{s}^i = ((\mathcal{S}_0^i, \mathcal{A}_0^i), \ldots, (\mathcal{S}_k^i, \mathcal{A}_k^i), \ldots)$ and r_t^i, represent sequences of states, actions and corresponding rewards, sampled by model simulation (total of M sampled sequences).

Training our sequential model involves computing the expected reward and its gradient, cf. (2.17) - (2.16). For each image, this involves simulating the model until the search is terminated, by generating sequences in the state-action space. At each time step t, an action $\mathcal{A}_t$ is sampled from the policy, using Algorithm 1. More precisely, first the distribution

[2]As the model decides when to terminate search, individually, for each search path.

$p_{\boldsymbol{\theta}}(d_t|\mathcal{S}_t)$ is sampled to decide whether the search is to be terminated (*done* action, i.e. $d_t = 1$). If so, then the output region index b_t and the confidence c_t are sampled from $p_{\boldsymbol{\theta}}(b_t|d_t = 1, \mathcal{S}_t)$ and $p_{\boldsymbol{\theta}}(c_t|d_t = 1, b_t, \mathcal{S}_t)$, respectively. Otherwise (not *done*, i.e. $d_t = 0$), an evidence region is selected by sampling $p_{\boldsymbol{\theta}}(e_t|d_t = 0, \mathcal{S}_t)$, and then the next fixation location is sampled from $p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|d_t = 0, e_t, \mathcal{S}_t)$. Finally, the state of the model is updated as described in Algorithm 2. Multiple sample sequences are generated in this way, for each image, and are used to estimate the expectations.[3]

4 Experiments and Results

In this section we present experiments to validate our search method on the challenging Pascal VOC 2012 object detection benchmark [38], over the withheld test set available via the evaluation server. In most of our experiments, the region space R consists of all segments extracted using a figure-ground region proposal method, and any state-of-the-art method applies. Without loss of generality, we select the CPMC algorithm [12] as their segments can be mapped with reasonable accuracy to detection targets (according to our studies, the average intersection-over-union overlap of the best segment enclosing rectangle with the ground truth bounding box, is 0.687).

Pipelines. To quantify the performance of different standard search models, we either solve the maximization problem (2.1) exactly, by performing exhaustive sliding window search (SW), exhaustive search over the CPMC region proposal set (RP), or by using our sequential reinforcement learning search model (RL).

4.1 Experimental Procedure

We now present and discuss the details of our experiments.

Proposal generation. To obtain our RP hypotheses, we run the public implementation [12] over the input image. For the sliding window (SW) baseline, region hypotheses are windows obtained by iterating over various window sizes and aspect ratios, and, for each scale and aspect ratio setting, by sliding the window with a fixed stride over the image. Our sliding window enumeration strategy results in $25,000$ windows per image. For region proposal, we use an optimized version of CPMC, which operates on a reduced search space formed by free-form regions. Note however that the optimization only applies to the segment generation step. Therefore, as of recent trends in region proposal-based detection, we work with larger pools typically having thousands of segments, and avoid the expensive segment filtering and ranking steps.

[3]For a training set of images, we will naturally aggregate (sum over) such estimates, for each image.

Feature extraction. For our feature extractor $\boldsymbol{q}$, we use the deep neural network of Krizhevsky et al. [39]. For a region, we invoke the network over the contents of the bounding box, and to capture context, on the entire image where the bounding box has been masked out (filled by its mean color). For each neural network evaluation, we record the output of the last fully connected layer. We concatenate the resulting feature vector with a representation of the bounding box size and aspect ratio, and obtain a final vector of $8,204$ values. Deep neural networks can be refined to further increase detection accuracy [1, 40], but in this work we have focused on optimal search models and have therefore opted to illustrate our model with a simpler linear SVM model trained using a generic feature extractor [39]. Note however that our method is sufficiently general to operate in conjunction with any confidence function.

Training the sequential reinforcement learning detector. We find the optimal parameter vector $\boldsymbol{\theta}$ that maximizes the expected reward (2.16) on the training set of the Pascal VOC 2012 Object detection challenge, using a BFGS optimizer. However, due to the high number of parameters the model is prone to overfit the data. Therefore, in practice we have chosen to initialize our confidence function parameters $\boldsymbol{\theta}_c$ by pre-training using a linear SVM where positive instances are ground truth bounding boxes and negative instances are sampled from other image locations (from the region proposal set R).

We initialize $\boldsymbol{\theta}_p$ by performing a regression from image regions to the centers of ground truth bounding boxes. We bias $\boldsymbol{\theta}_c$ towards their initial values while the rest of the parameters $(\boldsymbol{\theta}_d, \boldsymbol{\theta}_e, \sigma_c, \boldsymbol{\Sigma}_p)$ are initialized by uniform random sampling, in the range $[0, 1]$ and optimized using a zero-mean quadratic penalty, cf. (2.16). We validate the observation model parameters T_R as in (2.2) and T_F as in (2.3) on the Pascal VOC validation set, setting them to 64 pixels and 0.25, respectively. In practice the sensitivity associated to these parameters is not high; even if the model runs for several fixations, only a small fraction of the the total number of regions is observed. Empirically, we found the model to produce fairly short and effective search patterns with a number of 3.1 image locations inspected on average. As our policy is stochastic, multiple object instances can be found. Moreover, in evaluation, all visited regions above a threshold (e.g. all attended regions) are identified (locate and restart strategies are also possible).

4.2 Computational Efficiency and Accuracy

The running time and accuracy of our method is shown in Table 2.1. The accuracy of our sequential detector is close to that of the much more expensive sliding window baseline, although it is more than 70 times faster on an Intel Xeon 2.2Ghz CPU. This speedup takes into account the overhead of the RP algorithm (6.1 seconds) and the small overhead needed to sample the policy of our sequential detector (32 ms). We explicitly chose to give speedups in running times (as opposed to e.g. number of inspected locations or detector evaluations) as these also cover the overheads (e.g. in our case the additional work for the segment proposal generation step or estimating the next action), for fair comparisons with sliding windows or region proposal methods.

Besides comparisons with the SW and RP baselines, presented in Table 2.1, it could be useful to relate to other efficient search methods like [16]. As code is not available and there are quite significant methodological as well as region and feature representation differences, one can still consider overall speedups reported for similar datasets. For example, by operating over free-form regions obtained from selective search [13], [16] achieve a 9x acceleration, respectively, over sliding-windows methods in the PASCAL VOC 2007 dataset. Both us and [16] could additionally benefit from embedding our accelerated spatial class detectors into the complementary, effective multi-class detector scheduling mechanism for anytime recognition proposed in [35]. This could further produce a 2x speedup at roughly similar AP loss.

4.3 Qualitative Analysis

We note that the length of the search sequence is greatly dependent on the image (see Figure 2.2). If the target is close to the image center (e.g. the bicycle in Figure 2.2), the method tends to terminate the search after the first fixation, as it has already confidently located the object. If the target is located near the periphery, our model tends to continue the search over a longer time horizon. This behavior illustrates the model's capacity to adapt the search sequence length to the input image, as opposed to other fixed-lenght search methods in the literature.

Our visualizations of the results reveal three ways in which an evidence region (shown in green in Figure 2.2) may guide the search: i) The contextual region may not contain the target, but instead provide cues on its location (e.g. the ocean for the boat, or the road for the bus). In this case the model navigates from the surrounding context to the object itself. ii) The contextual region may include the target (e.g. the tree branches in which the bird is hiding), and inform the model to fixate a subregion likely to represent it. This situation corresponds to a coarse-to-fine search for the target. iii) Finally, the target may be too big, and fixations inside its region may initially not foveate it sufficiently for an observation to be made. In such cases, object subparts are often chosen as evidence regions to guide the search to other subparts (e.g. the various features of the front of the train engine), until the object is included in the observation set and therefore the confidence function is evaluated on its full extent. In this case, the model behavior resembles a perceptual grouping process in which smaller scale parts are integrated to deduce the extent of a large object.

5 Conclusions

We have presented a reinforcement learning model for visual object detection. In contrast to methods that operate exhaustively over a hypothesis space, we have derived a fully trainable sequential model that can efficiently sample only a few image locations in order to accumulate evidence on the target location. Our model is image and category specific and can explicitly balance the trade-off between exploration (improving accuracy) and exploitation (efficiently terminating search when sufficient evidence has been gathered). Our methodology is general and applicable to any detector response function. We report encouraging results in the PASCAL VOC 2012 object detection dataset, showing that the proposed methodology achieves almost *two orders of magnitude speedup* over sliding window methods.

Acknowledgments: Work partly supported by CNCS-UEFISCDI under PCE-2011-3-0438, JRP-RO-FR-2014-16.

Table 2.1: Detection accuracy (reported as Average Precision, AP) and running times of different methods in the test set associated to the PASCAL VOC 2012 object detection benchmark. Shown are results for the proposed sequential detection model (RL) as well as the classical sliding window (SW) and region proposal (RP) approach. The average running time of our sliding window baseline is 2, 691 seconds, regardless of the class. In the current experiments, we chose to optimize speedup to obtain a speedup of almost two orders of magnitude over sliding windows, but our method can also be tuned for accuracy, cf. (2.15). For instance, similar accuracy with exhaustive search methods can be achieved with an 18x speedup.

method	metric	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv	mean
SW	**AP (%)**	49.2	46.1	21.8	12.8	6.7	46.8	25.4	50.4	9.4	27.1	21.3	47.9	39.5	47.7	22.4	10.9	26.4	25.1	45.1	41.4	31.2
RP	**AP (%)**	44.5	36.3	28.0	14.4	3.6	44.7	27.0	57.6	8.8	26.6	20.2	47.7	39.7	45.1	22.6	8.7	25.6	23.6	42.1	39.2	30.3
RL	**AP (%)**	47.4	31.4	21.0	9.5	2.5	44.7	19.4	50.3	6.1	18.1	21.1	46.8	35.8	40.4	18.7	8.5	17.8	18.6	41.5	38.8	27.0
	evaluated regions	102	105	110	109	119	99	115	103	120	98	112	106	113	101	107	111	105	110	103	102	107
	running time (s)	37.6	38.8	40.0	39.8	41.9	36.4	40.8	37.1	42.3	36.6	39.9	38.0	40.2	36.5	38.2	39.5	37.6	39.3	36.9	37.0	38.7
	speedup (SW)	69.4	69.3	67.3	67.6	64.2	73.3	66.0	72.5	63.6	71.2	67.4	70.9	66.9	73.7	70.3	68.1	71.6	68.4	72.9	72.8	69.6
	speedup (RP)	8.3	8.1	7.9	7.8	7.3	8.5	7.5	8.3	7.3	8.6	7.7	8.1	7.6	8.4	8.0	7.8	8.1	7.8	8.2	8.3	8.0

References

[1] R. Girschick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[2] N. Shapovalova, M. Raptis, L. Sigal, and G. Mori, "Action is in the eye of the beholder: Eye-gaze driven model for spatio-temporal action localization," in *NeurIPS*, 2013.

[3] L. Itti, G. Rees, and J. K. Tsotsos, *Neurobiology of attention*. Elsevier, 2005.

[4] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Beyond sliding windows: Object localization by efficient subwindow search," in *CVPR*, 2008.

[5] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *ICCV*, 2009.

[6] P. F. Felzenszwalb, R. B. Girschick, and D. McAllester, "Cascade object detection with deformable part models," in *CVPR*, 2010.

[7] H. Harzallah, F. Jurie, and C. Schmid, "Combining efficient object localization and image classification," in *ICCV*, 2009.

[8] M. Pedersoli, A. Vedaldi, and J. Gonzales, "A coarse-to-fine approach for fast deformable object detection," in *CVPR*, 2011.

[9] Y. Wei and L. Tao, "Efficient histogram-based sliding window," in *CVPR*, 2010.

[10] E. Sudderth, A. Torralba, W. Freeman, and A. Willsky, "Learning hierarchical models of scenes, objects, and parts," in *ICCV*, 2005.

[11] I. Kokkinos, "Shufflets: Shared mid-level parts for fast object detection," in *ICCV*, 2013.

[12] J. Carreira and C. Sminchisescu, "CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts," *PAMI*, 2012.

[13] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *IJCV*, vol. 104, 2013.

[14] P. Rantalankila, J. Kannala, and E. Rahtu, "Generating object segmentation proposals using global and local search," in *CVPR*, 2014.

[15] P. Krahenbuhl and V. Koltun, "Learning to propose objects," in *CVPR*, 2015.

[16] A. Gonzalez-Garcia, A. Vezhnevets, and V. Ferrari, "An active search strategy for efficient object class detection," in *CVPR*, 2015.

[17] M. Choi, J. Lim, A. Torralba, and A. Willsky, "Exploiting hierarchical context on a large database of object categories," in *CVPR*, 2010.

[18] C. Desai, D. Ramanan, and C. Fowlkes, "Discriminative models for multi-class object layout," in *ICCV*, 2009.

[19] G. Heitz and D. Koller, "Learning spatial context: Using stuff to find things," in *ECCV*, 2008.

[20] A. Rabinovich, A. Vedaldi, C. Calleguillos, E. Wiewiora, and S. Belongie, "Objects in context," in *ICCV*, 2007.

[21] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *NeurIPS*, 2013.

[22] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *CVPR*, 2014.

[23] G. F. Lucas Paletta and C. Seifert, "Q-learning of sequential attention for visual object recognition from informative local descriptors," in *ICML*, 2005.

[24] N. J. Butko and J. R. Movellan, "Infomax control of eye movements," *IEEE TAMD*, vol. 2, no. 2, 2010.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 1998.

[26] H. Larochelle and G. E. Hinton, "Learning to combine foveal glimpses with a third-order boltzmann machine," in *NeurIPS*, 2009.

[27] S. Karayev, M. Fritz, and T. Darrell, "Anytime recognition of objects and scenes," in *CVPR*, 2014.

[28] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *NeurIPS*, 2014.

[29] B. Goodrich and I. Arel, "Reinforcement learning based visual attention with application to face detection," in *CVPR*, 2012.

[30] L. Bazzani, d. N. Freitas, H. Larochelle, and V. Muriono, "Learning attentional policies for tracking and recognition in video with deep networks," in *ICML*, 2011.

[31] D. Banica and C. Sminchisescu, "Second-Order Constrained Parametric Proposals and Sequential Search-Based Structured Prediction for Semantic Segmentation in RGB-D Images," in *CVPR*, 2015.

[32] S. Mathe and C. Sminchisescu, "Action from still image dataset and inverse optimal control to learn task specific visual scanpaths," in *NeurIPS*, 2013.

[33] S. Mathe and C. Sminchisescu, "Multiple instance reinforcement learning for efficient weakly-supervised detection in images," *CoRR*, vol. abs/1412.0100, 2014.

[34] J. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *ICCV*, 2015.

[35] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell, "Timely object recognition," in *NeurIPS*, 2012.

[36] G. Dulac-Arnold, L. Denoyer, N. Thome, M. Cord, and P. Gallinari, "Sequentially generated instance-dependent image representations for classification," *ICLR*, 2014.

[37] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992.

[38] M. Everingham, L. V. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/.

[39] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[40] R. Shaoqing, H. Kaiming, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *NeurIPS*, 2015.

[41] S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection," in *CVPR*, 2016.

A Supplementary Material

Training our sequential model involves computing the expected reward and its gradient, cf. (2.16) - (2.17) from the main paper. For each image, this involves simulating the model until the search is terminated, by generating sequences in the state-action space $\boldsymbol{s} = ((\mathcal{S}_0, \mathcal{A}_0), \ldots, (\mathcal{S}_t, \mathcal{A}_t), \ldots)$. At each time step t, an action $\mathcal{A}_t$ is sampled from the policy, using Algorithm 1. More precisely, first the distribution $p_{\boldsymbol{\theta}}(d_t|\mathcal{S}_t)$ is sampled to decide whether the search is to be terminated (*done* action, i.e. $d_t = 1$). If so, then the output region index b_t and the confidence c_t are sampled from $p_{\boldsymbol{\theta}}(b_t|d_t = 1, \mathcal{S}_t)$ and $p_{\boldsymbol{\theta}}(c_t|d_t = 1, b_t, \mathcal{S}_t)$, respectively. Otherwise (not *done*, i.e. $d_t = 0$), an evidence region is selected by sampling $p_{\boldsymbol{\theta}}(e_t|d_t = 0, \mathcal{S}_t)$, and then the next fixation location is sampled from $p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|d_t = 0, e_t, \mathcal{S}_t)$. Finally, the state of the model is updated, as described in Algorithm 2. In practice, multiple sample sequences are generated in this way, for each image, and are used to estimate the required expectations.[4] To evaluate the objective function gradient (see (2.17) in the main paper), one also needs to compute, at each time step t, the gradient of the log likelihood of each sampled action $\mathcal{A}_t$ under the current policy $\pi_{\boldsymbol{\theta}}(\mathcal{A}_t|\mathcal{S}_t)$, with respect to the model parameters $\boldsymbol{\theta}$. In the following, we present the analytical derivations that lead to a closed form expression for this gradient.

A.1 Useful Identities

The following identities will be useful during derivations:

$$\frac{\mathrm{d} \log \operatorname{sigm}(x)}{\mathrm{d}x} = 1 - \operatorname{sigm}(x), \tag{2.18}$$

$$\frac{\mathrm{d} \log \left[1 - \operatorname{sigm}(x)\right]}{\mathrm{d}x} = -\operatorname{sigm}(x), \tag{2.19}$$

$$\frac{\partial \operatorname{asmax}\left(\{x_j\}_{j=1}^n\right)}{\partial x_i} = \frac{e^{\alpha x_i}}{\sum_{j=1}^n e^{\alpha x_j}} \left[1 + \alpha \left[x_i - \operatorname{asmax}\left(\{x_j\}_{j=1}^n\right)\right]\right], \tag{2.20}$$

where $\operatorname{asmax}(X) = \frac{\sum_{x \in X} x e^{\alpha x}}{\sum_{x \in X} e^{\alpha x}}$. For any symmetric matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and for any $\boldsymbol{x} \in \mathbb{R}^{d \times 1}$, $\boldsymbol{Y} \in \mathbb{R}^{d \times d}$, $\boldsymbol{A} \in \mathbb{R}^{d \times n}$, $\boldsymbol{b} \in \mathbb{R}^{n \times 1}$, the following identity holds:

$$\frac{\partial}{\partial \boldsymbol{A}} \left(\boldsymbol{x} - \boldsymbol{Y}\boldsymbol{A}\boldsymbol{b}\right)^\top \boldsymbol{W} \left(\boldsymbol{x} - \boldsymbol{Y}\boldsymbol{A}\boldsymbol{b}\right) = -2\boldsymbol{Y}^\top \boldsymbol{W} \left(\boldsymbol{x} - \boldsymbol{Y}\boldsymbol{A}\boldsymbol{b}\right) \boldsymbol{b}^\top. \tag{2.21}$$

Furthermore, for any symmetric positive definite matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$, the following holds:

$$\frac{\partial \log |\boldsymbol{A}|}{\partial \boldsymbol{A}} = \boldsymbol{A}^{-1}. \tag{2.22}$$

[4]For a training set of images, we will naturally aggregate (sum over) such estimates, for each image.

Finally, for any symmetric invertible matrix $\boldsymbol{X} \in \mathbb{R}^{d\times d}$ and for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^{d\times 1}$, the following identity holds:

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}^{-1}\boldsymbol{b}}{\partial \boldsymbol{X}} = -\boldsymbol{X}^{-1}\boldsymbol{a}\boldsymbol{b}^\top \boldsymbol{X}^{-1}. \tag{2.23}$$

A.2 Model Equations

For convenience, let us denote by $\boldsymbol{\mu}(\boldsymbol{r}) = \frac{\boldsymbol{x}_1(\boldsymbol{r})+\boldsymbol{x}_2(\boldsymbol{r})}{2}$ the center of the bounding box tightly enclosing region $\boldsymbol{r}$, defined by its top-left and bottom-right corners $\boldsymbol{x}_1(\boldsymbol{r})$ and $\boldsymbol{x}_2(\boldsymbol{r})$, respectively. Similarly, let $\boldsymbol{\Delta}(\boldsymbol{r}) = \operatorname{diag}\left(\frac{\boldsymbol{x}_1(\boldsymbol{r})-\boldsymbol{x}_2(\boldsymbol{r})}{2}\right)$ be the diagonal matrix encoding the half width and height of this bounding box. We reproduce below the equations defining the model from the main body of the paper. They shall be referenced as needed for the derivations in section Section $A.3$.

$$\begin{aligned}
p_{\boldsymbol{\theta}}(d_t = 1|\mathcal{S}_t) &= \operatorname{sigm}\left[\boldsymbol{\theta}_d^\top \boldsymbol{v}\left(\mathcal{S}_t\right)\right], && (2.24)\\
p_{\boldsymbol{\theta}}(b_t = k|d_t = 1, \mathcal{S}_t) &= \frac{e^{\alpha f_c(\boldsymbol{r}_k)}}{\sum_{i\in H_t} e^{\alpha f_c(\boldsymbol{r}_i)}}, && (2.25)\\
p_{\boldsymbol{\theta}}(c_t|d_t = 1, b_t = k, \mathcal{S}_t) &= N(c_t|f_c(\boldsymbol{r}_k), \sigma_c), && (2.26)\\
p_{\boldsymbol{\theta}}(e_t|d_t = 0, \mathcal{S}_t) &= \frac{f_e(e_t)}{\sum_{i\in H_t\setminus E_t} f_e(i)}, && (2.27)\\
p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|d_t = 0, e_t, \mathcal{S}_t) &= N\left(\boldsymbol{z}_t \mid \boldsymbol{f}_p(e_t), \boldsymbol{\Delta}(\boldsymbol{r}_{e_t})^\top \boldsymbol{\Sigma}_p \boldsymbol{\Delta}(\boldsymbol{r}_{e_t})\right), && (2.28)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{v}\left(\mathcal{S}_t\right) &= \begin{bmatrix}\operatorname{asmax}\left(\{f_c\left(\boldsymbol{r}_i\right)\}_{i\in H_t}\right) & t & \frac{|H_t|}{|R|} & 1\end{bmatrix}^\top, && (2.29)\\
f_c(\boldsymbol{r}_i) &= \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i), && (2.30)\\
f_e\left(e_t\right) &= \exp\left[\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_{e_t})\right], && (2.31)\\
\boldsymbol{f}_p(e_t) &= \boldsymbol{\Delta}(\boldsymbol{r}_{e_t})\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_{e_t}) + \boldsymbol{\mu}(\boldsymbol{r}_{e_t}). && (2.32)
\end{aligned}$$

A.3 Gradient Derivations

A.3.1 Termination decision

The probability for terminating the search depends only on the confidence function regression parameters ($\boldsymbol{\theta}_c$) and on the termination decision parameters ($\boldsymbol{\theta}_d$):

$$\begin{aligned}
\frac{\partial \log p(d_t = 1|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} &= [1 - p(d_t = 1|\mathcal{S}_t)] \frac{\partial}{\partial \boldsymbol{\theta}_c} \left[\boldsymbol{\theta}_d^\top \boldsymbol{v}(\mathcal{S}_t)\right] \quad [(2.24), (2.18)] \\
&= [1 - p(d_t = 1|\mathcal{S}_t)]\, \theta_{d,1} \frac{\partial}{\partial \boldsymbol{\theta}_c} \left[\text{asmax}\left(\{f_c(\boldsymbol{r}_i)\}_{i \in H_t}\right)\right] \quad [(2.29)] \\
&= [1 - p(d_t = 1|\mathcal{S}_t)]\, \theta_{d,1} \\
&\cdot \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \left[1 + \alpha \left[\boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i) - \text{asmax}\left(\left\{\boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_j)\right\}_{j \in H_t}\right)\right]\right]}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}} \\
&\cdot \boldsymbol{q}(\boldsymbol{r}_i), \quad [(2.20)]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \log p(d_t = 1|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_d} &= [1 - p(d_t = 1|\mathcal{S}_t)] \frac{\partial}{\partial \boldsymbol{\theta}_d} \left[\boldsymbol{\theta}_d^\top \boldsymbol{v}(\mathcal{S}_t)\right] \quad [(2.24), (2.18)] \\
&= [1 - p(d_t = 1|\mathcal{S}_t)]\, \boldsymbol{v}(\mathcal{S}_t).
\end{aligned}$$

The derivatives corresponding to the probability for continuing the search can be similarly derived as:

$$\begin{aligned}
\frac{\partial \log p(d_t = 0|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} &= -p(d_t = 1|\mathcal{S}_t) \frac{\partial}{\partial \boldsymbol{\theta}_c} \left[\boldsymbol{\theta}_d^\top \boldsymbol{v}(\mathcal{S}_t)\right] \quad [(2.24), (2.19)] \\
&= -p(d_t = 1|\mathcal{S}_t) \theta_{d,1} \frac{\partial}{\partial \boldsymbol{\theta}_c} \left[\text{asmax}\left(\{f_c(\boldsymbol{r}_i)\}_{i \in H_t}\right)\right] \quad [(2.29)] \\
&= -p(d_t = 1|\mathcal{S}_t) \theta_{d,1} \\
&\cdot \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \left[1 + \alpha \left[\boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i) - \text{asmax}\left(\left\{\boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_j)\right\}_{j \in H_t}\right)\right]\right]}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}} \\
&\cdot \boldsymbol{q}(\boldsymbol{r}_i), \quad [(2.20)]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \log p(d_t = 0|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_d} &= -p(d_t = 1|\mathcal{S}_t) \frac{\partial}{\partial \boldsymbol{\theta}_d} \left[\boldsymbol{\theta}_d^\top \boldsymbol{v}(\mathcal{S}_t)\right] \quad [(2.24), (2.19)] \\
&= -p(d_t = 1|\mathcal{S}_t) \boldsymbol{v}(\mathcal{S}_t).
\end{aligned}$$

A.3.2 Bounding Box Selection

Given termination, the bounding box selection depends on the confidence function regression parameters ($\boldsymbol{\theta}_c$):

$$\begin{aligned}\frac{\partial \log p(b_t = k | d_t = 1, \mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} &= \frac{\partial}{\partial \boldsymbol{\theta}_c}\left[\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_k) - \log\left[\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}\right]\right] \qquad [(2.25)]\\ &= \alpha\left[\boldsymbol{q}(\boldsymbol{r}_k) - \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \boldsymbol{q}(\boldsymbol{r}_i)}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}}\right].\end{aligned}$$

The confidence value generated upon termination depends only on the confidence function regression parameters ($\boldsymbol{\theta}_c$) and on the confidence function standard deviation (σ_c):

$$\begin{aligned}\frac{\partial \log p(c_t = c | d_t = 1, b_t = k, \mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} &= \frac{c - \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_k)}{\sigma_c^2} \boldsymbol{q}(\boldsymbol{r}_k), \quad [(2.26)]\\ \frac{\partial \log p(c_t = c | d_t = 1, b_t = k, \mathcal{S}_t)}{\partial \sigma_c} &= -\frac{1}{\sigma_c} + \frac{\left[c - \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_k)\right]^2}{\sigma_c^3}. \quad [(2.26)]\end{aligned}$$

A.3.3 Evidence Region Selection

Given that the search is not terminated, the probability of selecting a specific region e_t at step t depends only on the evidence parameters $\boldsymbol{\theta}_e$:

$$\begin{aligned}\frac{\partial \log p(e_t = j | d_t = 0, \mathcal{S}_t)}{\partial \boldsymbol{\theta}_e} &= \frac{\partial}{\partial \boldsymbol{\theta}_e}\left[\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_j) - \log\left(\sum_{i \in H_t \setminus E_t} e^{\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)}\right)\right] \qquad [(2.27)]\\ &= \boldsymbol{q}(\boldsymbol{r}_j) - \frac{\sum_{i \in H_t \setminus E_t} e^{\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)} \boldsymbol{q}(\boldsymbol{r}_i)}{\sum_{i \in H_t \setminus E_t} e^{\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)}}.\end{aligned}$$

A.3.4 Position Selection

Given that the search has not been terminated and evidence region e has been selected, the likelihood for fixating location $\boldsymbol{z}_t$ depends on the position regression parameters $\boldsymbol{\theta}_p$ and on the covariance matrix $\boldsymbol{\Sigma}_p$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_t | d_t = 0, e_t = e, \mathcal{S}_t) = N\left(\boldsymbol{z}_t \,\middle|\, \boldsymbol{\Delta}(\boldsymbol{r}_e) \boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e) \boldsymbol{\mu}(\boldsymbol{r}_e), \boldsymbol{\Delta}(\boldsymbol{r}_e) \boldsymbol{\Sigma}_p \boldsymbol{\Delta}(\boldsymbol{r}_e)\right). \tag{2.33}$$

The gradient of the log-likelihood is given by

$$\frac{\partial \log p(\boldsymbol{z}_t = \boldsymbol{z} | d_t = 0, e_t = e, \mathcal{S}_t)}{\partial \boldsymbol{\theta}_p}$$

$$= -\frac{1}{2}\frac{\partial}{\partial \boldsymbol{\theta}_p}\left[\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]^\top\right.$$

$$\left.\cdot \boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\right] \qquad [(2.33)]$$

$$= -\frac{1}{2}\left[\frac{\partial}{\partial \boldsymbol{\theta}_p}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]^\top\right.$$

$$\left.\cdot\ \boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\right]^\top$$

$$= \left[\boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\boldsymbol{q}(\boldsymbol{r}_e)^\top\right]^\top \qquad [(2.21)]$$

$$= \boldsymbol{q}(\boldsymbol{r}_e)\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]^\top \boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1},$$

$$\frac{\partial \log p(\boldsymbol{z}_t = \boldsymbol{z} | d_t = 0, e_t = e, \mathcal{S}_t)}{\partial \boldsymbol{\Sigma}_p}$$

$$= -\frac{1}{2}\boldsymbol{\Sigma}_p^{-1} - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\Sigma}_p}\left[\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]^\top\right.$$

$$\left.\cdot \boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\right] \qquad [(2.33), (2.22)]$$

$$= -\frac{1}{2}\boldsymbol{\Sigma}_p^{-1} - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\Sigma}_p}\left[\left[\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\right]^\top\right.$$

$$\left.\cdot \boldsymbol{\Sigma}_p^{-1}\left[\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]\right]\right]$$

$$= -\frac{1}{2}\boldsymbol{\Sigma}_p^{-1} + \frac{1}{2}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]$$

$$\cdot\left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_e) - \boldsymbol{\Delta}(\boldsymbol{r}_e)\boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_e)\right]^\top \boldsymbol{\Delta}(\boldsymbol{r}_e)^{-1}\boldsymbol{\Sigma}_p^{-1}. \qquad [(2.23)]$$

A.3.5 Centralized results

The final derivatives of action likelihoods with respect to model parameters are summarized below:

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} = [1 - p(d_t = 1|\mathcal{S}_t)]\, \theta_{d,1}$$

$$\cdot \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \left[1 + \alpha \left[\boldsymbol{\theta}_c^\top \boldsymbol{q}\left(\boldsymbol{r}_i\right) - \text{asmax}\left(\left\{\boldsymbol{\theta}_c^\top \boldsymbol{q}\left(\boldsymbol{r}_j\right)\right\}_{j \in H_t}\right)\right]\right]}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}} \cdot \boldsymbol{q}\left(\boldsymbol{r}_i\right)$$

$$+ \alpha \left[\boldsymbol{q}(\boldsymbol{r}_k) - \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \boldsymbol{q}(\boldsymbol{r}_i)}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}}\right] + \frac{c - \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_k)}{\sigma_c^2} \boldsymbol{q}(\boldsymbol{r}_k),$$

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \sigma_c} = -\frac{1}{\sigma_c} + \frac{\left[c - \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_k)\right]^2}{\sigma_c^3},$$

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_d} = [1 - p(d_t = 1|\mathcal{S}_t)]\, \boldsymbol{v}(\mathcal{S}_t),$$

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_e} = 0,$$

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_p} = 0,$$

$$\frac{\partial \log p(d_t = 1, b_t = k, c_t = c|\mathcal{S}_t)}{\partial \boldsymbol{\Sigma}_p} = 0,$$

$$\frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z}|\mathcal{S}_t)}{\partial \boldsymbol{\theta}_c} = -p(d_t = 1|\mathcal{S}_t)\theta_{d,1}$$

$$\cdot \frac{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)} \left[1 + \alpha \left[\boldsymbol{\theta}_c^\top \boldsymbol{q}\left(\boldsymbol{r}_i\right) - \text{asmax}\left(\left\{\boldsymbol{\theta}_c^\top \boldsymbol{q}\left(\boldsymbol{r}_j\right)\right\}_{j \in H_t}\right)\right]\right]}{\sum_{i \in H_t} e^{\alpha \boldsymbol{\theta}_c^\top \boldsymbol{q}(\boldsymbol{r}_i)}} \cdot \boldsymbol{q}\left(\boldsymbol{r}_i\right),$$

$$\frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z}|\mathcal{S}_t)}{\partial \sigma_c} = 0,$$

$$\frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z} | \mathcal{S}_t)}{\partial \boldsymbol{\theta}_d} = -\, p(d_t = 1 | \mathcal{S}_t) \boldsymbol{v}(\mathcal{S}_t),$$

$$\frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z} | \mathcal{S}_t)}{\partial \boldsymbol{\theta}_e} = \boldsymbol{q}(\boldsymbol{r}_k) - \frac{\sum_{i \in H_t \setminus E_t} e^{\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)} \boldsymbol{q}(\boldsymbol{r}_i)}{\sum_{i \in H_t \setminus E_t} e^{\boldsymbol{\theta}_e^\top \boldsymbol{q}(\boldsymbol{r}_i)}},$$

$$\begin{aligned} \frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z} | \mathcal{S}_t)}{\partial \boldsymbol{\theta}_p} = {} & \boldsymbol{q}(\boldsymbol{r}_k) \left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_k) - \boldsymbol{\Delta}(\boldsymbol{r}_k) \boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_k) \right]^\top \\ & \cdot \boldsymbol{\Delta}(\boldsymbol{r}_k)^{-1} \boldsymbol{\Sigma}_p^{-1}, \end{aligned}$$

$$\begin{aligned} & \frac{\partial \log p(d_t = 0, e_t = k, \boldsymbol{z}_t = \boldsymbol{z} | \mathcal{S}_t)}{\partial \boldsymbol{\Sigma}_p} = -\frac{1}{2} \boldsymbol{\Sigma}_p^{-1} + \frac{1}{2} \boldsymbol{\Sigma}_p^{-1} \boldsymbol{\Delta}(\boldsymbol{r}_k)^{-1} \\ & \cdot \left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_k) - \boldsymbol{\Delta}(\boldsymbol{r}_k) \boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_k) \right] \left[\boldsymbol{z} - \boldsymbol{\mu}(\boldsymbol{r}_k) - \boldsymbol{\Delta}(\boldsymbol{r}_k) \boldsymbol{\theta}_p^\top \boldsymbol{q}(\boldsymbol{r}_k) \right]^\top \boldsymbol{\Delta}(\boldsymbol{r}_k)^{-1} \boldsymbol{\Sigma}_p^{-1}. \end{aligned}$$

Paper II

Deep Reinforcement Learning of Region Proposal Networks for Object Detection

Aleksis Pirinen[1] **Cristian Sminchisescu**[1,2]
[1]Centre for Mathematical Sciences, Lund University
[2]Institute of Mathematics of the Romanian Academy

Abstract

We propose *drl-RPN*, a deep reinforcement learning-based visual recognition model consisting of a sequential region proposal network (RPN) and an object detector. In contrast to typical RPNs, where candidate object regions (RoIs) are selected greedily via class-agnostic NMS, drl-RPN optimizes an objective closer to the final detection task. This is achieved by replacing the greedy RoI selection process with a sequential attention mechanism which is trained via deep reinforcement learning (RL). Our model is capable of accumulating class-specific evidence over time, potentially affecting subsequent proposals and classification scores, and we show that such context integration significantly boosts detection accuracy. Moreover, drl-RPN automatically decides when to stop the search process and has the benefit of being able to jointly learn the parameters of the policy and the detector, both represented as deep networks. Our model can further learn to search over a wide range of exploration-accuracy trade-offs, which makes it possible to specify or adapt the exploration extent at test time. The resulting search trajectories are image- and category-dependent, yet rely only on a single policy over all object categories. Results on the MS COCO and PASCAL VOC challenges show that our approach outperforms established, typical state-of-the-art object detection pipelines.

1 Introduction

Visual object detection focuses on localizing each instance within a pre-defined set of object categories in an image, most commonly by estimating bounding boxes with associated confidence values. Accuracy on this task has increased dramatically over the last years [13, 14, 15], reaping the benefits of increasingly deep and expressive feature extractors [16, 17, 18, 19]. Several contemporary state-of-the-art detectors [14, 15, 20] follow a two-step process. First bottom-up region proposals are obtained, either from an internal region proposal network (RPN) [14], trained alongside the detection network, or from an external one [21, 22, 23, 24, 25]. In the second step proposals are classified and their localization accuracy may be fine-tuned.

There has recently been an increased interest in active, sequential search methods [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 4, 36]. This class of approaches, to which our model belongs, seek to only inspect parts of each image sequentially. In this work we aim to make active recognition models more flexible, as characterized by i) a finely-tuned active search process where decisions of where to look next and when to stop searching are image- and category-dependent; ii) context information is aggregated as search proceeds and is used in decision making and to boost detection accuracy; iii) the detector and search policy parameters are tightly linked into a single deep RL-based optimization problem where they are estimated jointly; iv) the search process can be adapted to a variety of exploration-accuracy trade-offs during inference; and v) learning to search is only weakly supervised, as we indicate the model what success means without telling it exactly how to achieve it – there is no apprenticeship learning or trajectory demonstration.

Methodologically we propose *drl-RPN*, a sequential region proposal network combining an RL-based top-down search strategy, implemented as a convolutional gated recurrent unit, and a two-stage bottom-up object detector. Notably, our model is used for class-agnostic proposal generation but leverages class-specific information from earlier time steps when proposing subsequent regions (RoIs). This context aggregation is also used to increase detection accuracy. Our model offers the flexibility of jointly training the policy and detector, both represented as deep networks, which we perform in alternation in the framework of deep RL. We emphasize that drl-RPN can be used, in principle, in conjunction with any exhaustive two-stage state-of-the-art object detector operating on a set of object proposals, such as Faster R-CNN (Fr R-CNN) [14] or R-FCN [15].

2 Related Work

Among the first to use deep feature extractors for object detection was [37], whereas [13] combined the power of smaller and more plausible region proposal sets with such deep architectures. This was followed up in [20, 14, 15, 38] with impressive results. There is also a recent trend towards solutions where bounding box and classification predictions are produced in one shot [39, 40, 41]. Such methods increase detection speed, but sometimes at the cost of a lower accuracy.

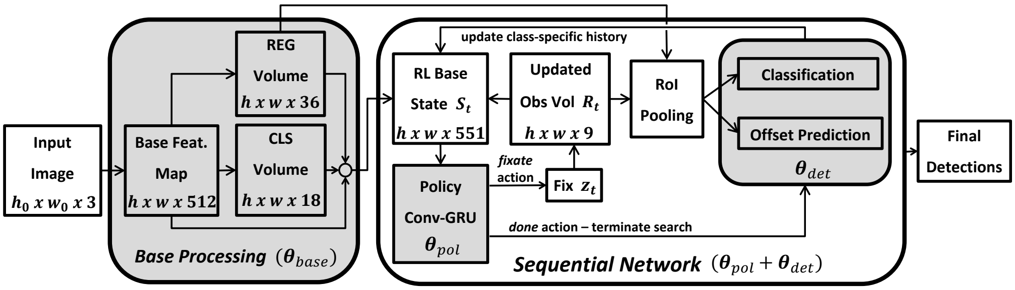
The general detection pipeline above is characterized by its exhaustive, non-sequential nature: even if the set of windows to classify is reduced a-priori, all windows are still classified simultaneously and independently of each other. In contrast, sequential methods for object detection can in principle be designed to accumulate evidence over time to potentially improve accuracy at the given task. Such approaches can coarsely be divided as RL-based [26, 27, 28, 29, 30, 31, 35, 4, 36] and non-RL-based [32, 33, 34]. Our drl-RPN model is of the former category.

Orthogonally from us, [4] propose anytime models where a detector can be stopped asynchronously during inference: multi-class models are scheduled sequentially and the order of exhaustively applying sliding window detectors is optimized, potentially without running detectors for some classes. Our drl-RPN is also a multi-class detector but instead avoids searching all image locations. In [26] a class-specific agent observes the entire image, then performs a sequence of bounding box transformations until tightly enclosing the object. Results were improved in [29] where a joint multi-agent Q-learning system [42] is used and sub-agents cooperate to find several objects. In contrast, [27] use policy gradients to train a 'saccade-and-fixate' search policy over pre-computed RoIs that automatically determines when to stop searching. The formulation in [27] is however one-versus-all, not entirely deep, and is primarily designed for single instance detection. On the contrary, the deep model we propose detects multiple instances and categories and thus circumvents the need to train multiple search policies as in [26, 27]. Fast R-CNN [20] is coupled with a tree-structured search strategy in [28] and results exceed or match the basic Fast R-CNN. Differently from us however, [28] manually specify the number of proposals during inference (hence stopping is not automatic but preset) and the detector is not refined jointly with the search policy.

Notable non-RL-based active search approaches include [32, 33, 34]. A soft attention mechanism is learned in [32] where directions for the next step are predicted, akin to a gradual shift in attention; [33] apply a search strategy for partial image exploration guided by statistical relations between regions; and [34] use adjacency and zoom prediction to focus processing on sub-regions likely to contain objects.

3 Two-Step Proposal-based Detection

We now briefly review those standard two-step proposal-based object detection components which will form some of the building blocks of our sequential drl-RPN model. Such detectors take as input an image of size $h_0 \times w_0 \times 3$ and process it through a base network. We use the same VGG-16 base network [17] as in [14]. The base network outputs the *base feature map* with dimension $h \times w \times d$, where h and w depend on h_0 and w_0, and $d = 512$ for VGG-16. The network then separates into two branches: RoI *generation* followed by RoI *pooling* and *classification*.



Input Image $h_0 \, x \, w_0 \, x \, 3$
Base Feat. Map $h \, x \, w \, x \, 512$
REG Volume $h \, x \, w \, x \, 36$
CLS Volume $h \, x \, w \, x \, 18$
Base Processing (θ_{base})
update class-specific history
RL Base State S_t $h \, x \, w \, x \, 551$
Updated Obs Vol R_t $h \, x \, w \, x \, 9$
RoI Pooling
Classification
Offset Prediction
θ_{det}
Policy Conv-GRU θ_{pol}
fixate action
Fix z_t
done action – terminate search
Sequential Network $(\theta_{pol} + \theta_{det})$
Final Detections

Figure 3.1: Overview of our proposed drl-RPN model. The base processing is illustrated on the left and shows how the initial state $\boldsymbol{S}_0$ is formed. Each time step t, the agent decides whether to terminate search based on its stochastic policy $\pi_{\boldsymbol{\theta}}(a_t|s_t)$, cf. (3.1) - (3.8). As long as search has not terminated, a *fixate* action a_t^f is issued and a new location $\boldsymbol{z}_t$ is visited; the RoI observation volume $\boldsymbol{R}_t$ is updated in an area centered at $\boldsymbol{z}_t$. All corresponding RoIs are sent to the RoI pooling module followed by classification and class-specific bounding box offset predictions. The class-specific probability vectors are inserted to the history volume $\boldsymbol{V}_t^4$ which is merged with the RL base state volume $\boldsymbol{S}_t$. Based on the updated state, a new action is taken at time step $t+1$ and the process is repeated until the *done* action a_t^d is issued; then all the selected predictions throughout the trajectory are collected. The trainable parts of the network (including the feature extraction, classification and regression models, and policy) are highlighted in gray. See also Figure 3.4 for some visualizations of drl-RPN search strategies.

A region proposal network (RPN) is used for generating RoIs, where a d-dimensional feature vector is produced at each spatial location on the base feature map and is sent through two class-agnostic layers: box-regression (*reg*) and box-classification (*cls*). To increase object recall, several proposals are predicted relative to k anchor boxes (we use the same $k = 9$ anchors as [14]). The last task of the RPN is to reduce the number of RoIs forwarded to RoI pooling and classification. This is performed by class-agnostic NMS based on the objectness scores in *cls*. All RoIs forwarded by the RPN are converted to small spatially fix-sized feature maps by means of RoI max pooling and are subsequently sent to two fully-connected layers which perform class probability and bounding box offset predictions.

4 Sequential Region Proposal Network

We now present the architecture of drl-RPN, consisting of an object detector and a policy $\pi_{\boldsymbol{\theta}}$, see Figure 3.1. For the detector we use a publicly available TensorFlow [43] implementation[1] of Fr R-CNN, on top of which we implement our drl-RPN model. In principle however, drl-RPN can be integrated with any RPN-based detector, such as [15]. The search policy is based on a convolutional gated recurrent unit (Conv-GRU) which replaces fully-connected components of the GRU [44] with convolutions.

[1]`https://github.com/smallcorgi/Faster-RCNN_TF`

The input to the Conv-GRU at time t is the RL base state volume $\boldsymbol{S}_t$ (see Section 4.1) and the previous hidden state $\boldsymbol{H}_{t-1}$. The output is a two-channel action volume $\boldsymbol{A}_t$. The spatial extent of all inputs and outputs are $h \times w$. We denote by $*$ the convolution operator and $\odot$ denotes element-wise multiplication. Weights and biases are denoted $\boldsymbol{W}$ and $\boldsymbol{b}$, respectively, and $\sigma[\cdot]$ is the logistic sigmoid function. Below are the equations of our Conv-GRU agent:

$$\boldsymbol{O}_t = \sigma\left[\boldsymbol{W}_{so} * \boldsymbol{S}_t + \boldsymbol{W}_{ho} * \boldsymbol{H}_{t-1} + \boldsymbol{b}_o\right], \tag{3.1}$$

$$\tilde{\boldsymbol{H}}_t = \boldsymbol{W}_{sh} * \boldsymbol{S}_t + \boldsymbol{W}_{hh} * (\boldsymbol{O}_t \odot \boldsymbol{H}_{t-1}) + \boldsymbol{b}_h, \tag{3.2}$$

$$\boldsymbol{Z}_t = \sigma\left[\boldsymbol{W}_{sz} * \boldsymbol{S}_t + \boldsymbol{W}_{hz} * \boldsymbol{H}_{t-1} + \boldsymbol{b}_z\right], \tag{3.3}$$

$$\boldsymbol{H}_t = (1 - \boldsymbol{Z}_t) \odot \boldsymbol{H}_{t-1} + \boldsymbol{Z}_t \odot \tanh[\tilde{\boldsymbol{H}}_t], \tag{3.4}$$

$$\tilde{\boldsymbol{A}}_t = \mathrm{relu}[\boldsymbol{W}_{h\tilde{a}} * \boldsymbol{H}_t + \boldsymbol{b}_{\tilde{a}}], \tag{3.5}$$

$$\boldsymbol{A}_t = \tanh[\boldsymbol{W}_{\tilde{a}a} * \tilde{\boldsymbol{A}}_t + \boldsymbol{b}_a]. \tag{3.6}$$

The output $\boldsymbol{A}_t$ of the Conv-GRU corresponds to two possible actions, see Section 4.1.

We let $\boldsymbol{\theta}$ denote *all* parameters of the system where drl-RPN is used, which can be decomposed as $\boldsymbol{\theta}_{\text{base}}$, $\boldsymbol{\theta}_{\text{det}}$ and $\boldsymbol{\theta}_{\text{pol}}$. Here $\boldsymbol{\theta}_{\text{base}}$ are the parameters of the base network and the original RPN; $\boldsymbol{\theta}_{\text{det}}$ are the parameters of the classifier and bounding box offset predictor; and $\boldsymbol{\theta}_{\text{pol}}$ are the search policy parameters, cf. (3.1) - (3.8). The joint training of $\boldsymbol{\theta} = [\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}; \boldsymbol{\theta}_{\text{pol}}]$ is described in Section 5.

4.1 States and Actions

The state at time t is the tuple $s_t = (\boldsymbol{R}_t, \boldsymbol{S}_t, \boldsymbol{H}_t)$, where $\boldsymbol{R}_t \in \{0,1\}^{h \times w \times k}$ is the *RoI observation volume*, $\boldsymbol{S}_t \in \mathbb{R}^{h \times w \times (d+2k+N+1)}$ is the *base state* and $\boldsymbol{H}_t \in \mathbb{R}^{h \times w \times 300}$ is the hidden state of the Conv-GRU. Here N is the number of object categories considered. There are two types of actions, corresponding to one channel each of $\boldsymbol{A}_t$ in (3.6): a *fixate* action a_t^f and the *done* action a_t^d. The done action is binary, where $a_t^d = 1$ corresponds to terminating search. A *fixate* action $a_t^f = \boldsymbol{z}_t$ is issued if $a_t^d = 0$, where $\boldsymbol{z}_t$ is the $(h \times w)$-plane coordinate of the next fixation. We next define $\boldsymbol{R}_t$ and explain how it relates to *fixate* actions a_t^f, after which we present $\boldsymbol{S}_t$ and explain its connection to $\boldsymbol{R}_t$. Finally, we describe how actions are sampled using our parametric stochastic policy $\pi_{\boldsymbol{\theta}}$.

RoI observation volume. The drl-RPN agent maintains a binary volume $\boldsymbol{R}_t$ of size $h \times w \times k$ in which the (i,j,l):th entry is equal to 1 if and only if the corresponding RoI is part of the region proposal set forwarded to the RoI pooling / classification module of the network. We initialize $\boldsymbol{R}_t$ as an all-zeros volume. After a *fixate* action a_t^f, a part of $\boldsymbol{R}_t$ in a neighborhood of the fixation location $\boldsymbol{z}_t$ is updated. This neighborhood is a rectangular area centered at $\boldsymbol{z}_t$, in which the side lengths h_{rect} and w_{rect} are a fraction each of h and w, respectively (we set $h_{\text{rect}} = h/4$ and $w_{\text{rect}} = w/4$). We set all entries of $\boldsymbol{R}_t$ inside this rectangle to 1 to indicate that these RoIs have been selected. Note that we here restrict our algorithm to determine at what *spatial* locations to sample RoIs in the $(h \times w)$-plane, so all k anchor candidates are used per spatial location.

Base state volume. The state $\boldsymbol{S}_t$ consists of $\boldsymbol{V}_t^1 \in \mathbb{R}^{h \times w \times d}$, $\boldsymbol{V}_t^2, \boldsymbol{V}_t^3 \in \mathbb{R}^{h \times w \times k}$, and $\boldsymbol{V}_t^4 \in \mathbb{R}^{h \times w \times (N+1)}$. We set $\boldsymbol{V}_0^1$ to the base feature map (conv5_3) and $\boldsymbol{V}_0^2$ to the objectness layers (in *cls*) of the RPN. The *reg* volume of the RPN is used for $\boldsymbol{V}_t^3$, where $\boldsymbol{V}_0^3$ is set to the magnitude of the $[0,1]$-normalized offsets $[\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2]$. We use $\boldsymbol{R}_t$ to update these volumes, setting the corresponding locations in $\boldsymbol{V}_t^1$, $\boldsymbol{V}_t^2$ and $\boldsymbol{V}_t^3$ to -1 to indicate that those locations have been inspected.

The volume $\boldsymbol{V}_t^4$ is a class-specific history of what has been observed so far ($\boldsymbol{V}_0^4 = \boldsymbol{0}$). After a fixation the selected RoIs are sent to class-specific predictors. Then *local* class-specific NMS is applied to the classified RoIs to get the most salient information at that location. As we have final bounding box predictions for the surviving RoIs, we map them to certain spatial locations of $\boldsymbol{V}_t^4$. Specifically, the input image is divided into $L \times L$ bins of size $\approx h_0/L \times w_0/L$ to get a coarse representation of where the agent has looked by assigning each NMS-survivor to the bin containing its center coordinates ($L = 3$). The history $\boldsymbol{V}_t^4$ at these locations is updated with those class probability vectors as a running average.

We use 3×3 convolutional kernels for the base input $\boldsymbol{V}_t^1$ since the effective receptive field is already wide given that we are operating on deep feature maps. For the auxiliary input $\boldsymbol{V}_t^2$ - $\boldsymbol{V}_t^4$ we apply larger 9×9 kernels.

Stochastic policy. We now describe how $\boldsymbol{A}_t$ in (3.6) is used to select actions. Let $\boldsymbol{A}_t^d$ and $\boldsymbol{A}_t^f$ denote the first (*done*) and second (*fixate*) layers of $\boldsymbol{A}_t$, respectively. The done layer $\boldsymbol{A}_t^d$ is bilinearly re-sized to 25×25 and stacked as a vector $\boldsymbol{d}_t \in \mathbb{R}^{625}$. The probability of terminating in state s_t is then given by

$$\pi_{\boldsymbol{\theta}} \left(a_t^d = 1 | s_t \right) = \sigma \left[\boldsymbol{w}_d^\top \boldsymbol{d}_t + t \right], \tag{3.7}$$

where $\boldsymbol{w}_d$ is a trainable weight vector. The fixation layer $\boldsymbol{A}_t^f$ is transformed to a probability map $\hat{\boldsymbol{A}}_t^f$ by applying a spatial softmax to $\boldsymbol{A}_t^f$. The probability of fixating location $\boldsymbol{z}_t = (i,j)$ given that the agent did not terminate is $\hat{\boldsymbol{A}}_t^f[\boldsymbol{z}_t]$, where $\hat{\boldsymbol{A}}_t^f[\boldsymbol{z}_t]$ is the (i,j):th entry of $\hat{\boldsymbol{A}}_t^f$. The probability of fixating location $\boldsymbol{z}_t$ in state s_t is thus given by

$$\pi_{\boldsymbol{\theta}} \left(a_t^d = 0, a_t^f = \boldsymbol{z}_t | s_t \right) = \left(1 - \sigma \left[\boldsymbol{w}_d^\top \boldsymbol{d}_t + t \right] \right) \hat{\boldsymbol{A}}_t^f[\boldsymbol{z}_t]. \tag{3.8}$$

4.2 Contextual Class Probability Adjustment

Typical detection pipelines classify all regions simultaneously and independently of each another, which is simple and efficient but limits information exchange between regions. We argue for an alternative where the search process and the classification of candidate proposals are unified into a single system, creating synergies between both tasks. We already explained how classified regions are used to guide the search process and now augment drl-RPN to use context accumulation also to perform a posterior update of its detection probabilities based on the search trajectory. Note that we update detections *after* terminating search which gives also early detections an opportunity of being adjusted. In principle however, one could update detections as search proceeds based only on past detections.

The augmented model uses a summary of all object instances discovered during search to refine the final class probability scores for these detections. For this we use the history aggregation described in Section 4.1. Given the up to L^2 history vectors, we stack them as an $L^2(N+1)$-dimensional vector $\boldsymbol{x}_{\text{hist}}$ and represent non-observed regions by zeros in $\boldsymbol{x}_{\text{hist}}$. The final classification layer softmax$(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ is replaced with softmax$(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} + \boldsymbol{f}_{\text{hist}}(\boldsymbol{x}_{\text{hist}}))$ to account for the search trajectory. We use a one-layer activation $\boldsymbol{f}_{\text{hist}}(\boldsymbol{x}_{\text{hist}}) = \tanh(\boldsymbol{W}_{\text{hist}}\boldsymbol{x}_{\text{hist}} + \boldsymbol{b}_{\text{hist}})$.

5 Training

Training the full model (detector and policy) proceeds in alternation. Recall that we distinguish between three sets of parameters: $\boldsymbol{\theta} = [\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}; \boldsymbol{\theta}_{\text{pol}}]$, where $[\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}]$ are the parameters of the original Fr R-CNN. We use a pre-trained network as initialization of $[\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}]$, where in the pre-training we use the same settings as [14], including an additional anchor scale of 64^2 pixels for MS COCO. Xavier initialization [45] is used for the search policy parameters $\boldsymbol{\theta}_{\text{pol}}$. We next explain how to learn $\boldsymbol{\theta}_{\text{pol}}$ via deep RL; the joint training of the full system is described in Section 5.3.

5.1 Reward Signal

There are two criteria which the agent should balance. First, the chosen RoIs should yield high overlap with ground truth object instances, and second, the number of RoIs should be as low as possible to reduce the number of false positives and to maintain a manageable processing time.

Fixate action reward. To balance the above trade-off we give a small negative reward $-\beta$ for each *fixate* action (we set $\beta = 0.075$), but the agent also receives a positive reward for fixations yielding increased intersection-over-union (IoU) with any ground truth instances $\boldsymbol{g}_i$ for the current image. For each object instance $\boldsymbol{g}_i$ we keep track of the so-far maximum IoU-yielding[2] RoIs selected by the agent at previous time steps $0, \dots, t-1$. Let this be denoted IoU^i and note that IoU$^i = 0$ at $t = 0$. When $t \geq 1$ we compute the maximum IoU for all ground truth instances $\boldsymbol{g}_i$ given by RoIs from that particular time step, denoted

[2]This refers to IoU *after* class-specific bounding box adjustments to ensure that the objective lies as close as possible to the final detection task.

IoU_t^i, and check if $\text{IoU}_t^i > \text{IoU}^i \geq \tau$, where we set $\tau = 0.5$ in accordance with the positive threshold for PASCAL VOC. For each ground truth $\boldsymbol{g}_i$ satisfying this condition we give the positive reward $(\text{IoU}_t^i - \text{IoU}^i)/\text{IoU}_{\max}^i$, after which we set $\text{IoU}^i = \text{IoU}_t^i$. Here $\text{IoU}_{\max}^i$ is the maximum IoU that $\boldsymbol{g}_i$ has with *any* of all hwk possible regions. Hence the fixation reward r_t^f at time t is given by

$$r_t^f = -\beta + \sum_i \mathbb{1}\left[\boldsymbol{g}_i \colon \text{IoU}_t^i > \text{IoU}^i \geq \tau\right] \frac{\text{IoU}_t^i - \text{IoU}^i}{\text{IoU}_{\max}^i}. \tag{3.9}$$

Done action reward. Upon termination the agent receives a final reward reflecting the quality of the full search trajectory:

$$r_t^d = \sum_i \mathbb{1}\left[\boldsymbol{g}_i \colon \text{IoU}_{\max}^i \geq \tau\right] \frac{\text{IoU}^i - \text{IoU}_{\max}^i}{\text{IoU}_{\max}^i}. \tag{3.10}$$

Here IoU^i is the maximum IoU-yielding RoI (with $\boldsymbol{g}_i$) selected by the agent in the entire trajectory. Note that (3.10) evaluates to zero if all $\boldsymbol{g}_i$ are maximally covered and otherwise becomes increasingly negative depending on how severely the ground truths are missed.

5.1.1 Separation of Rewards

Although drl-RPN is a single-agent system taking one action per time step via the policy specified in (3.7) - (3.8), it may be viewed as consisting of two subagents `agt_d` and `agt_f` with some shared and some individual parameters. The agent `agt_d`, governed by (3.7), decides whether to keep searching, whereas `agt_f` is governed by (3.8) and controls where to look *given* that `agt_d` has not terminated search. We argue that `agt_d` should not necessarily be rewarded based on the performance of `agt_f`. For example, early in training `agt_f` may choose poor fixation locations, thus missing the objects. In a standard reward assignment both `agt_f` and `agt_d` receive negative reward based on the behavior of `agt_f`. However, only `agt_f` should be penalized in this situation as it alone is responsible for not localizing objects despite the opportunity given by `agt_d`.

Instead of giving the actual fixation reward r_t^f in (3.9) to `agt_d` we define an optimistic corresponding reward as

$$\tilde{r}_t^f = -\beta + \max_{\text{IoU} \geq \tau} \frac{\text{IoU} - \text{IoU}^i}{\text{IoU}_{\max}^i}. \tag{3.11}$$

The reward (3.11) reflects the maximum increase of IoU of *one* single ground truth instance $\boldsymbol{g}_i$ attainable by *any* fixate action. Note that (3.11) may not always be optimistic; the true fixation reward (3.9) can be higher in images with several objects (by covering multiple instances in one fixation). Early in training however, (3.11) is often higher than (3.9). Therefore we give $\max(r_t^f, \tilde{r}_t^f)$ as fixation reward to `agt_d`.

This *separation of rewards* between `agt_d` and `agt_f` helped drl-RPN find a reasonable termination policy; it otherwise tended to stop the search process too early. Separation of rewards does not increase computational cost and is easy to implement, making it a simple adjustment for improving learning efficiency. It is applicable in any RL problem where actions have similar hierarchical dependencies.

5.1.2 Adaptive Exploration-Accuracy Trade-Off

So far we have described drl-RPN with a fixed exploration penalty β in training, cf. (3.9). After training the exploration extent is hard-coded into the policy parameters. By treating β as an input we can instead obtain a *goal-agnostic* agent whose exploration extent may be specified at test time. Goal-agnostic agents have also been proposed in different contexts in contemporary work; see e.g. [46, 47].

Specifically, an adjustment is performed between equations (3.5) - (3.6), where a constant β-valued feature map is appended to $\tilde{\boldsymbol{A}}_t$. In training we define a set of β-values the model is exposed to and for each trajectory we randomly sample a β from this set. In testing we simply specify β, which does not have to be from the set of β-values seen in training.

5.2 Objective Function

To learn the policy parameters we maximize the expected cumulative reward on the training set, given by $J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=1}^{|\boldsymbol{s}|} r_t\right]$, where $\boldsymbol{s}$ represents a trajectory of states and actions, sampled by running the model from the initial state s_0 (cf. Section 4.1). A sample-based approximation to the gradient [48] of the objective function $J(\boldsymbol{\theta})$ is obtained using REINFORCE [49]. We use 50 search trajectories to approximate the true gradient, forming one batch in our gradient update (one image per batch), and update the policy parameters via backpropagation using Adam [50]. To increase sample efficiency we use the return normalization in [30], where cumulative rewards for each episode are normalized to mean 0 and variance 1 over the batch. The maximum trajectory length is set to 12.

5.3 Joint Training of Policy and Detector

As we use one image per batch it is straightforward to also tune the detector parameters $[\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}]$. Once $\boldsymbol{\theta}_{\text{pol}}$ has been updated for an image (with $[\boldsymbol{\theta}_{\text{base}}; \boldsymbol{\theta}_{\text{det}}]$ frozen[3]) we fix $\boldsymbol{\theta}_{\text{pol}}$ and produce one more search trajectory for that image. The RoIs selected by drl-RPN during this trajectory are used as RoIs in Fr R-CNN instead of RoIs from the standard RPN, but otherwise the detector is updated as in [14]. Once the full drl-RPN model has been trained it is simple to also learn (refine) the parameters of the posterior class probability predictor in Section 4.2. Specifically, we jointly train $\boldsymbol{W}$, $\boldsymbol{W}_{\text{hist}}$, $\boldsymbol{b}$ and $\boldsymbol{b}_{\text{hist}}$ as for the original Fr R-CNN model, except that drl-RPN is used for generating RoIs. The remaining parameters are kept frozen at this stage, although it is possible to alternate.

[3]We keep $\boldsymbol{\theta}_{\text{base}}$ frozen throughout as tuning the base network did not increase performance.

6 Experiments

We now compare our proposed drl-RPN to Fr R-CNN[4] on the MS COCO [51] and PASCAL VOC [52] detection challenges. Unless otherwise specified we refer by drl-RPN to the model using the posterior class probability adjustments introduced in Section 4.2. We report results mainly for drl-RPN models trained with a fixed exploration penalty $\beta = 0.075$; results for the goal-agnostic model presented in Section 5.1.2 are found in Section 6.3.

For PASCAL VOC we repeat the alternating training in Section 5.3 for 70k iterations on VOC 07+12 train-val (for VOC 2012 we include the 2007 test set in training, as typical). The learning rate for $\boldsymbol{\theta}_{\text{pol}}$ is initially $2 \cdot 10^{-5}$ ($4 \cdot 10^{-6}$ after 50k iterations) and $\boldsymbol{\theta}_{\text{det}}$ has corresponding learning rates $2.5 \cdot 10^{-4}$ and $2.5 \cdot 10^{-5}$. We use the same settings for MS COCO (trained on COCO 2014 train-val) but alternate for 350k iterations and update the learning rate after 250k iterations.

We compare drl-RPN to Fr R-CNN using the standard RPN and also investigate some variants of drl-RPN. Specifically, we compare to a model using the class-specific history only to guide the search process but not for posterior class probability adjustments (np); to a model completely void of a class-specific history (nh); and to a model enforcing 12 fixations per image (12-fix). Adaptively stopping (ads) variants of drl-RPN are used if not otherwise specified. For the various drl-RPN models we also show the average fraction of RoIs forwarded for class-specific predictions (called *exploration*, reported in %) and the average number of fixations per image.

6.1 Results on MS COCO

Results on MS COCO 2015 test-std and test-dev are shown in Table 3.1, together with PASCAL VOC 2007 and 2012 results for these models. On MS COCO the mAP of drl-RPN is 1.1 higher than for Fr R-CNN. Comparing with the ads-np and ads-nh models, the posterior class probability adjustments yield mAP boosts of 0.7 and 0.9, respectively. Enforcing 12 fixations marginally improves mAP by 0.1, while significantly increasing exploration by 25%. Also, drl-RPN increases mean average recall (mAR) by 0.6. As for PASCAL VOC, drl-RPN outperforms Fr R-CNN by 1.1 and 0.8 mAP on VOC 2012 and 2007, respectively. The class-specific history yields 0.5 and 0.7 mAP boosts respectively on VOC 2012 and 2007. Enforcing 12 fixations leads to negligible mAP improvements.

Overall, drl-RPN consistently outperforms the baseline Fr R-CNN model. We also see that the class-specific history with posterior adjustments yields significantly improved accuracy and that the adaptive stopping condition provides a drastic reduction in average exploration, yet matches the mAP of the corresponding 12-fixation policy.

[4]We report results obtained for the implementation we used, which are often higher than in [14]; this was achieved by training for more iterations.

Table 3.1: Detection results on the MS COCO 2015 test sets, as well as the PASCAL VOC 2012 and 2007 test sets (two right-most columns). The first row of each drl-RPN modification shows the detection performance (mAP or mAR) and the second row shows average exploration (% of forwarded RoIs) and average number of fixations per image.

model	settings	test-dev mAP@.5	test-dev mAP@.75	test-dev mAP@[.5, .95]	test-dev mAR@[.5, .95]	test-std mAP@.5	test-std mAP@.75	test-std mAP@[.5, .95]	test-std mAR@[.5, .95]	voc12-test mAP@.5	voc07-test mAP@.5
RPN	**default**	42.7	21.4	22.3	32.3	42.7	21.1	22.3	32.3	73.0	75.6
drl-RPN	**ads**	43.3	23.0	23.4	32.9	43.3	23.0	23.4	32.9	74.1	**76.4**
		40.9%, 8.1	40.9%, 8.1	40.9%, 8.1	40.9%, 8.1	40.7%, 8.0	40.7%, 8.0	40.7%, 8.0	40.7%, 8.0	37.7%, 7.1	39.9%, 7.6
	12-fix	**43.6**	**23.1**	**23.5**	33.3	**43.6**	**23.1**	**23.5**	33.3	**74.2**	**76.4**
		51.7%, 12	51.7%, 12	51.7%, 12	51.7%, 12	51.6%, 12	51.6%, 12	51.6%, 12	51.6%, 12	50.4%, 12.0	51.1%, 12.0
	ads, np	43.2	22.0	22.8	33.1	43.0	21.9	22.7	33.2	73.7	76.1
		40.9%, 8.1	40.9%, 8.1	40.9%, 8.1	40.9%, 8.1	40.7%, 8.0	40.7%, 8.0	40.7%, 8.0	40.7%, 8.0	37.7%, 7.1	39.9%, 7.6
	12-fix, np	43.4	22.2	23.0	**33.5**	43.3	22.0	22.8	**33.5**	74.0	76.0
		51.7%, 12	51.7%, 12	51.7%, 12	51.7%, 12	51.6%, 12	51.6%, 12	51.6%, 12	51.6%, 12	50.4%, 12.0	51.1%, 12.0
	ads, nh	43.1	21.8	22.6	33.0	42.9	21.7	22.5	33.2	73.6	75.7
		39.0%, 7.5	39.0%, 7.5	39.0%, 7.5	39.0%, 7.5	38.9%, 7.5	38.9%, 7.5	38.9%, 7.5	38.9%, 7.5	34.7%, 6.4	37.0%, 7.0

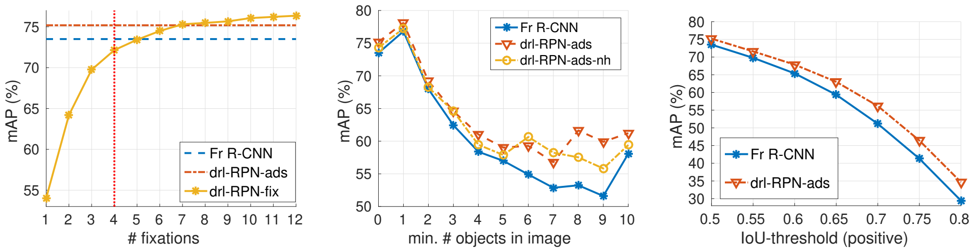


Figure 3.2: Additional results on the PASCAL VOC 2007 test set. Left: Using a constant, preset number of fixations per image requires almost twice as many fixations per image to reach the same detection accuracy as the adaptively stopping model. Middle: The mAP of drl-RPN compared to Fr R-CNN is relatively higher in more crowded scenes and the class-specific history appears more useful in such scenes. Right: The relative performance of drl-RPN compared to Fr R-CNN generally increases with increased IoU-threshold (cf. Figure 3.3).

6.2 Results on PASCAL VOC

Table 3.2: Detection results on the PASCAL VOC 2007 and 2012 test sets. We also show drl-RPN's average exploration and average number of fixations per image

model	settings	mAP - 2007	mAP - 2012
RPN	**default**	73.5	70.4
	drl-RPN det	73.6	70.6
	all RoIs	74.2	70.7
drl-RPN	**ads | 22.9%, 4.0**	75.2	70.8
	12-fix | 40.3%, 12.0	**76.4**	**72.2**
	ads, np | 22.9%, 4.0	74.5	70.4
	12-fix, np | 41.7%, 12.0	75.5	71.8
	ads, nh | 22.1%, 3.9	74.3	70.1

Table 3.2 shows results on PASCAL VOC 2007 and 2012. To show the effect of joint policy-detector training we also present Fr R-CNN results using the drl-RPN tuned detector parameters (drl-RPN det). For VOC 2007, drl-RPN-ads achieves 1.7 mAP above Fr R-CNN. By enforcing 12 fixations drl-RPN more significantly outperforms the Fr R-CNN baseline by 2.9 mAP. Moreover, both the ads- and 12-fix drl-RPN models achieve higher mAP compared to an exhaustive variant of Fr R-CNN which forwards all RoIs (without class-agnostic NMS), so increasing mAP is *not* merely a matter of detecting more RoIs. The Fr R-CNN results change negligibly when replacing the class-specific detector parameters to those of the tuned drl-RPN detector. We also tried drl-RPN without detector tuning, which caused an mAP drop of 2.0 (see the supplementary material). Hence, unsurprisingly, it is crucial to perform detector tuning jointly with the policy learning. Furthermore, the class-specific history yields considerably better results – see also Section 6.3 and Figure 3.2 (middle). Similar results apply to PASCAL VOC 2012. The adaptive stopping drl-RPN-ads improves upon Fr R-CNN by 0.4 mAP; it also surpasses the exhaustive "all RoIs" variant. At 12 fixations drl-RPN significantly outperforms Fr R-CNN by 1.8 mAP.

Comparing the VOC and COCO results, search trajectories for VOC are about 50% shorter on average. This is not surprising given that COCO scenes are significantly more crowded and complex; indeed, this further shows the benefit of an adaptive search with

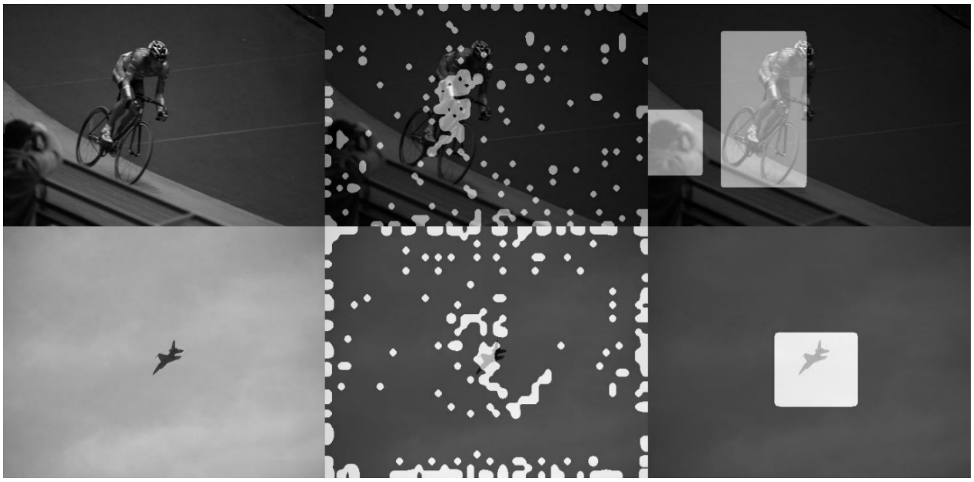


Figure 3.3: The drl-RPN attention (right) is more object-centric and less scattered over the image compared to the standard RPN (middle), resulting in fewer false positives. The respective input images are shown in the left column. These visualizations are shown in gray for clarity.

automatic stopping condition.

In Figure 3.2 (left) we show results on VOC 2007 when enforcing exactly n fixations per image for $n = 1, \ldots, 12$. The mAP increases with the number of fixations and surpasses drl-RPN-ads for $n \geq 7$ and Fr R-CNN for $n \geq 5$. Drawn is also a vertical line corresponding to the mean number of fixations of drl-RPN-ads (4.0). Comparing to the model with a preset number of fixations clearly shows the benefit of the automatic stopping (3.0 mAP difference).

6.3 Additional Results

To further investigate our model we evaluate drl-RPN and Fr R-CNN in a few settings on the PASCAL VOC 2007 test set. Some visualizations of drl-RPN search strategies and final detections are shown in Figure 3.4.

mAP vs. number of objects per image. Comparing drl-RPN-ads to drl-RPN-ads-nh in Figure 3.2 (middle) shows that class-specific context aggregation becomes increasingly useful in crowded scenes, which is quite expected (exceptions for 6, 7 objects). Also, drl-RPN-ads outperforms Fr R-CNN at all object counts and the improvement gets more pronounced in more crowded scenes.

mAP vs. IoU-threshold. Figure 3.2 (right) shows that the relative performance of drl-RPN increases with box IoU-threshold τ, despite using the standard $\tau = 0.5$ during training. Comparing the COCO-style mAP scores (mAP@[.5, .95]), drl-RPN even more

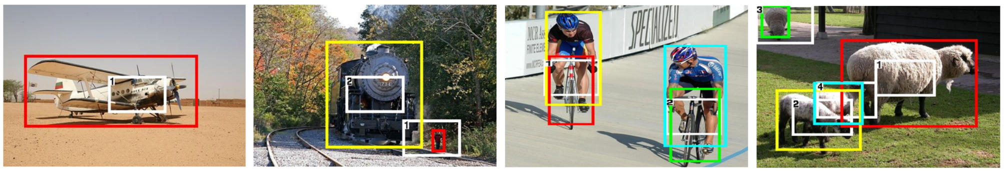


Figure 3.4: Upscaled fixation areas in white (cf. $\boldsymbol{R}_t$ in Section 4.1) generated by drl-RPN and detection boxes (colored) for a few PASCAL VOC 2007 test images. We also show the time step each area was observed. The sizes of the fixation areas are *not* related to the sizes of the selected RoIs; they simply determine where RoIs are being forwarded for class-specific predictions. More visualizations are provided in the supplementary material.

significantly outperforms Fr R-CNN with 44.3 against 41.3 mAP. See also the attention comparison in Figure 3.3 showing where (spatially) RoIs are forwarded for class-specific predictions. For drl-RPN this corresponds to the upscaled fixation areas (cf. $\boldsymbol{R}_t$ in Section 4.1). For the standard RPN we locate where the survivors of the class-agnostic NMS end up spatially and upsample those locations to match the image size.

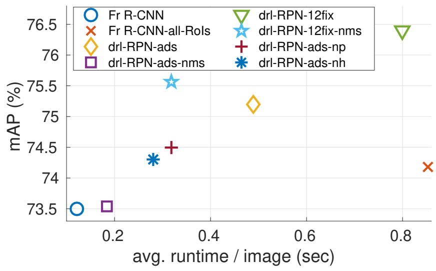


Figure 3.5: mAP vs. runtime for evaluated models on the VOC 2007 test set. Fr R-CNN, while fast, offers very limited tuning of the speed-accuracy trade-off, whereas drl-RPN can be adapted to a wide range of requirements on accuracy or speed. See also Figure 3.6.

Runtime and mAP comparisons. Figure 3.5 shows mAP and runtime comparisons between various models evaluated in this work. These results are reported using a Titan X GPU. Our drl-RPN model outperforms Fr R-CNN in detection accuracy but not in speed. This is mainly because drl-RPN forwards a larger set of RoIs (even though this set is much more spatially compact, cf. Figure 3.3). The sequential processing, based on the Conv-GRU described in Section 4, also adds an overhead of about 13 ms per fixation. Applying class-agnostic NMS to gate the drl-RPN proposals yields runtimes closer to that of Fr R-CNN while still improving mAP. Also, drl-RPN outperforms the exhaustive Fr R-CNN variant in both speed and accuracy.

Exploration-accuracy trade-off. Figure 3.6 shows results[5] for the goal-agnostic extension of drl-RPN which takes the exploration penalty β as an additional input (cf. Section 5.1.2), evaluated for the set $\{0.025, 0.050, \ldots, 0.750\}$ of β-values used in training. We also compare to a model using class-agnostic NMS to gate the drl-RPN proposals. With

[5]We here use the model without posterior class probability adjustments.

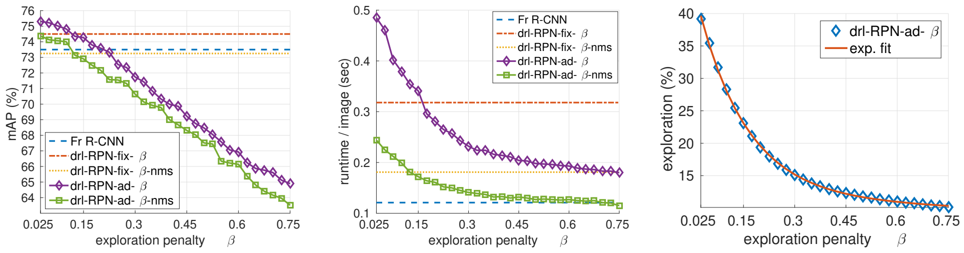


Figure 3.6: Investigation of exploration-accuracy trade-off on the PASCAL VOC 2007 test set. Left: For small β the goal-agnostic agents outperform the fixed-β counterparts as well as Fr R-CNN, while mAP expectedly decreases as β increases. Middle: Average runtime also decreases with increased β; at $\beta = 0.15$ (twice the β used for fixed-β models) the goal-agnostic models become faster than the fixed-β counterparts. Right: Exploration vs. β with a fitted two-term exponential $ae^{-b\beta} + ce^{-d\beta}$. The accurate functional fit allows for specifying the exploration extent at test time.

this straightforward extension we obtain models which can be adjusted to a wide range of speed-accuracy trade-offs.

7 Conclusions

We have presented drl-RPN, a sequential deep reinforcement learning model of 'where to look next' for visual object detection, which automatically determines when to terminate the search process. The model produces image- and category-dependent search trajectories, yet it features a single policy over all object categories. All the (deep) parameters – including the fixation policy, stopping conditions, and object classifiers – can be trained jointly and experiments show that such joint refinement improves detection accuracy. Overall, drl-RPN achieves results superior to exhaustive, typical state-of-the-art methods and is particularly accurate in applications demanding higher IoU-thresholds for positive detections.

Results showing the advantages of a class-specific memory and context-aggregation within drl-RPN have also been presented. This offers a mechanism to incrementally accumulate evidence at earlier visited image regions and detections to guide the search process and boost detection accuracy. As expected, such a mechanism leads to even more dramatic improvements in more crowded scenes. Finally, we have shown that drl-RPN can learn to operate across a wide variety of exploration-accuracy trade-offs, which makes it possible to specify the exploration extent at test time.

Acknowledgments: This work was supported by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535, the EU Horizon 2020 Grant DE-ENIGMA, and SSF.

References

[1] G. F. Lucas Paletta and C. Seifert, “Q-learning of sequential attention for visual object recognition from informative local descriptors,” in *ICML*, 2005.

[2] N. J. Butko and J. R. Movellan, “Infomax control of eye movements,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, 2010.

[3] S. Karayev, M. Fritz, and T. Darrell, “Anytime recognition of objects and scenes,” in *CVPR*, 2014.

[4] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell, “Timely object recognition,” in *NeurIPS*, 2012.

[5] H. Larochelle and G. E. Hinton, “Learning to combine foveal glimpses with a third-order boltzmann machine,” in *NeurIPS*, 2009.

[6] B. Goodrich and I. Arel, “Reinforcement learning based visual attention with application to face detection,” in *CVPR*, 2012.

[7] L. Bazzani, d. N. Freitas, H. Larochelle, and V. Muriono, “Learning attentional policies for tracking and recognition in video with deep networks,” in *ICML*, 2011.

[8] S. Mathe and C. Sminchisescu, “Action from still image dataset and inverse optimal control to learn task specific visual scanpaths,” in *NeurIPS*, 2013.

[9] X. Chen and A. Gupta, “An implementation of faster rcnn with study for region sampling,” *arXiv preprint arXiv:1702.02138*, 2017.

[10] S. Singh, D. Hoiem, and D. Forsyth, “Learning a sequential search for landmarks,” in *CVPR*, 2015.

[11] X. Chen and A. Gupta, “Spatial memory for context reasoning in object detection,” in *ICCV*, 2017.

[12] T. Kong, F. Sun, A. Yao, H. Liu, M. Lu, and Y. Chen, “Ron: Reverse connection with objectness prior networks for object detection,” in *CVPR*, 2017.

[13] R. Girschick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *CVPR*, 2014.

[14] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *NeurIPS*, 2015.

[15] J. Dai, Y. Li, K. He, and J. Sun, “R-fcn: Object detection via region-based fully convolutional networks,” in *NeurIPS*, 2016.

[16] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[20] R. Girshick, "Fast r-cnn," in *ICCV*, 2015.

[21] J. Carreira and C. Sminchisescu, "CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts," *PAMI*, 2012.

[22] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *IJCV*, vol. 104, 2013.

[23] P. Rantalankila, J. Kannala, and E. Rahtu, "Generating object segmentation proposals using global and local search," in *CVPR*, 2014.

[24] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, "Multiscale combinatorial grouping," in *CVPR*, 2014.

[25] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *ECCV*, 2014.

[26] J. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *ICCV*, 2015.

[27] S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection," in *CVPR*, 2016.

[28] Z. Jie, X. Liang, J. Feng, X. Jin, W. Lu, and S. Yan, "Tree-structured reinforcement learning for sequential object localization," in *NeurIPS*, 2016.

[29] X. Kong, B. Xin, Y. Wang, and G. Hua, "Collaborative deep reinforcement learning for joint object search," *CVPR*, 2017.

[30] K. Hara, M.-Y. Liu, O. Tuzel, and A.-M. Farahmand, "Attentional network for visual object detection," *arXiv preprint arXiv:1702.01478*, 2017.

[31] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *NeurIPS*, 2014.

[32] D. Yoo, S. Park, K. Paeng, J.-Y. Lee, and I. S. Kweon, "Action-driven object detection with top-down visual attentions," *arXiv preprint arXiv:1612.06704*, 2016.

[33] A. Gonzalez-Garcia, A. Vezhnevets, and V. Ferrari, "An active search strategy for efficient object class detection," in *CVPR*, 2015.

[34] Y. Lu, T. Javidi, and S. Lazebnik, "Adaptive object detection using adjacency and zoom prediction," in *CVPR*, 2016.

[35] Z. Li, Y. Yang, X. Liu, F. Zhou, S. Wen, and W. Xu, "Dynamic computational time for visual attention," in *ICCV Workshops*, 2017.

[36] X. S. Chen, H. He, and L. S. Davis, "Object detection in 20 questions," in *WACV*, 2016.

[37] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *ICLR*, 2014.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *ECCV*, 2014.

[39] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed, "Ssd: Single shot multibox detector," *ECCV*, 2016.

[40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[41] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *CVPR*, 2017.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NeurIPS Deep Learning Workshop*, 2013.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *NeurIPS Workshop on Deep Learning*, 2014.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, 2010.

[46] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *ICLR*, 2017.

[47] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*, 2017.

[48] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 1998.

[49] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[51] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[52] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[53] A. Pirinen and C. Sminchisescu, "Deep reinforcement learning of region proposal networks for object detection," in *CVPR*, 2018.

A Supplementary Material

In this supplementary material we provide additional results for our drl-RPN agent. We also show several visualizations of drl-RPN operating on images in Figure 3.7 - 3.9.

Tables 3.3 and 3.4 show per-category results for drl-RPN and Faster R-CNN (Fr R-CNN) on the PASCAL VOC 2007 and 2012 test sets, respectively. The corresponding results showing only the mean average precisions (mAPs) are given in Table 3.2 of the main paper. As a reminder, for the Fr R-CNN model using a standard region proposal network (RPN), these are the various model settings we evaluate (which were also evaluated in the main paper):

- *default* is the standard Fr R-CNN detector;
- *drl-RPN det* is the standard Fr R-CNN detector, but with drl-RPN tuned parameters for the detector head;
- *all RoIs* is an exhaustive variant of the standard Fr R-CNN detector, which does not use any class-agnostic NMS (i.e. it forwards all RoIs from the RPN).

We also evaluate drl-RPN in several different settings. The first four were evaluated already in the main paper, whereas the latter three are new to this supplementary material – see the bottom three rows of Table 3.3. The different drl-RPN settings we evaluate are as follows:

- *ads* denotes the adaptively stopping variant;
- 12-*fix* denotes a variant which always performs 12 fixations per image;
- *np* denotes a variant which uses a class-specific history only to guide the search process but not for posterior class probability adjustments;
- *nh* denotes a variant which is completely void of a class-specific history;
- *nt* denotes a variant which has not tuned the detection head in parallel with the fixation policy learning;
- *ads-rng* is an adaptively stopping variant with a uniform random fixation policy (we ensure it does not visit the same region multiple times);
- *ads-obj* is an adaptively stopping variant with a heuristic fixation policy. Specifically, it extracts the objectness layer of the RPN and attends spatial locations in descending order of objectness (we ensure that the same locations are not visited multiple times by setting objectness values to 0 at already explored regions).

As evident in Table 3.3, performing joint detector and policy training is superior to only training the policy (drl-RPN-nt yields mAP 1.9 points below that of the fully tuned drl-RPN model, despite exploring more). Conversely, the Fr R-CNN results change only marginally (from mAP 73.5 to 73.6) if we replace the class-specific detector parameters to those of the tuned drl-RPN detector. Hence, unsurprisingly, it is crucial to perform detector tuning jointly with the policy learning.

The fully reinforcement learnt drl-RPN-ads significantly outperforms the simpler drl-RPN-obs and drl-RPN-rng baselines with 2.4 and 4.6 mAP points, respectively, despite exploring much less (e.g. almost half the amount of fixations compared to drl-RPN-rng). Thus, a finely tuned search strategy – jointly trained with the adaptive stopping conditions – is essential for efficient and accurate exploration and detection. The reason why drl-RPN-obs is significantly worse than drl-RPN-ads could be because the objectness scores are often high also where there are no objects; see Figure 3.3 in the main paper.

Table 3.3: Detection results for different methods on the PASCAL VOC 2007 test set. All models were trained on VOC 07+12 train-val. Shown are results for Fr R-CNN, using either drl-RPN or the original RPN. Shown is also average exploration and average number of fixations per image.

model	settings	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv	mean
RPN	**default**	76.9	79.2	73.1	61.4	59.8	81.6	84.7	86.7	55.7	81.8	70.4	83.0	84.6	78.2	77.9	41.8	71.9	71.9	77.0	72.1	73.5
	drl-RPN det	76.5	78.3	72.5	59.7	60.6	82.3	84.3	86.8	55.5	81.4	70.1	83.9	84.5	78.2	78.2	42.1	71.6	72.3	77.5	72.0	73.6
	all RoIs	77.0	84.5	74.7	68.4	59.5	87.4	86.5	85.7	59.2	81.9	69.3	84.8	86.3	77.2	78.4	43.3	75.5	72.2	82.7	72.3	75.3
drl-RPN	**ads 23.2%, 4.0**	79.2	80.1	70.4	67.7	57.3	86.6	84.9	86.4	57.3	83.6	73.9	84.2	87.4	79.3	78.9	44.7	74.1	77.8	83.2	67.0	75.2
	12-fix 40.3%, 12.0	79.3	80.3	74.5	69.4	60.6	86.9	85.9	87.3	58.4	83.7	73.2	85.6	87.8	79.4	79.0	45.2	75.9	77.7	84.3	72.9	76.4
	ads, np 23.2%, 4.0	78.6	79.4	70.3	66.9	56.3	87.1	85.0	80.8	56.9	81.8	73.0	85.3	87.6	78.0	78.2	43.5	76.7	75.8	83.0	65.6	74.5
	12-fix, np 40.3%, 12.0	78.2	80.0	74.6	65.7	59.5	87.4	86.4	86.7	57.2	82.7	72.5	85.3	87.5	78.3	78.3	43.3	76.8	75.3	83.6	69.6	75.5
	ads, nh 23.3%, 4.2	77.8	79.8	69.2	67.0	55.6	87.6	85.3	81.4	58.2	82.3	73.7	83.4	86.7	78.0	78.6	41.7	75.8	75.6	84.1	64.9	74.3
	ads, nt 24.8%, 4.4	73.6	79.6	69.8	67.1	54.2	86.1	80.5	86.8	56.1	81.8	69.5	83.3	86.8	77.1	78.5	42.5	75.7	74.0	78.9	63.5	73.3
	ads-obj 26.9%, 5.2	77.0	79.7	69.9	66.4	56.6	86.0	79.8	81.0	56.5	77.5	71.7	78.8	86.6	78.2	77.9	42.3	75.3	75.4	79.4	59.6	72.8
	ads-rng 35.4%, 7.4	70.7	78.8	68.6	61.1	48.5	85.5	77.7	80.6	56.2	75.2	70.9	79.0	84.8	76.7	70.5	41.3	68.9	74.6	78.8	63.2	70.6

Table 3.4: Detection results for different methods on the PASCAL VOC 2012 test set. All models were trained on the union of VOC 07+12 train-val and VOC 07 test. Shown are results for Fr R-CNN, using either drl-RPN or the original RPN. Shown is also average exploration and average number of fixations per image.

model	settings	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv	mean
RPN	**default**	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5	70.4
	drl-RPN det	82.6	79.9	72.4	53.7	52.8	78.0	76.4	88.0	48.6	74.8	55.6	84.8	80.7	81.1	79.6	44.8	72.0	61.0	80.9	64.4	70.6
	all RoIs	83.5	79.6	71.3	55.6	51.1	77.5	76.6	89.5	50.3	74.5	55.0	85.7	78.7	81.7	80.9	43.6	71.7	60.7	80.1	66.7	70.7
drl-RPN	**ads 22.5%, 4.0**	82.7	79.7	72.0	57.0	51.4	76.9	76.2	87.7	48.7	75.1	57.6	85.4	81.2	82.4	80.6	45.3	72.8	63.5	81.9	57.8	70.8
	12-fix 40.3%, 12.0	84.4	80.6	72.8	57.5	53.0	78.9	77.3	89.5	49.8	76.5	56.5	86.4	81.5	83.1	81.3	47.7	73.2	64.3	83.2	65.9	72.2
	ads, np 22.5%, 4.0	81.7	80.0	72.1	56.1	51.0	77.7	76.5	88.0	48.8	74.8	57.5	85.6	80.7	81.6	80.5	44.1	71.9	61.7	80.9	57.3	70.4
	12-fix, np 43.0%, 12.0	84.1	80.8	72.3	56.6	52.5	79.5	77.5	89.5	50.2	76.7	56.5	86.5	80.9	82.1	81.2	46.8	72.6	62.1	82.3	65.3	71.8
	ads, nh 20.9%, 3.6	82.3	79.4	72.2	56.0	50.6	75.0	75.4	88.4	48.0	74.7	56.4	84.7	80.7	81.9	80.7	43.7	71.9	61.6	80.1	58.1	70.1

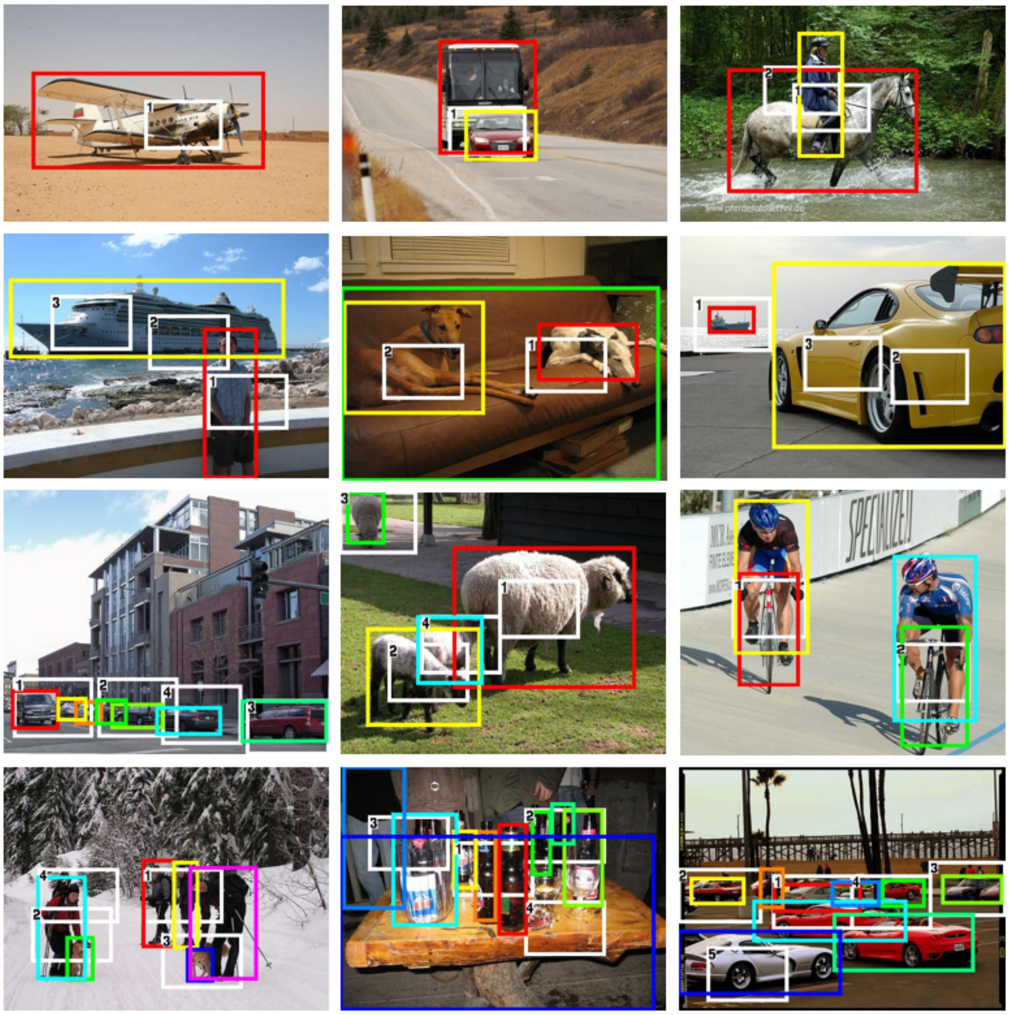


Figure 3.7: Upscaled fixation areas (*attention boxes*, white) generated by our sequential search model drl-RPN, together with final detection boxes (colored), for several images from the PASCAL VOC 2007 test. Each attention box has an associated number, showing at which time-step $t \geq 1$ the corresponding area in the feature map was observed. Depending on the complexity of a scene, such as the number and layout of objects, the model automatically determines when to stop the search process. The top two rows show examples of short search trajectories, in which only a few object instances exist in the image. In contrast, longer trajectories are shown in the bottom two rows, corresponding to images with more object instances and/or categories. As such, the number of *fixate* actions is *not* necessarily equal to the number of object instances, but depends also on the layout of the objects (e.g. how close objects are to each other). For example, in the top-mid image, only one *fixate* action is necessary to simultaneously locate the bus and the car, whereas an additional fixation is produced for the image to the right on the second row. Overall however, the number of fixations typically increases with the number of object instances, as would be expected. Note that the fix-sized attention boxes are *not* related to the sizes of the RoIs being forwarded for class-specific predictions. These boxes only correspond to what subset (and where) of RoIs are selected.

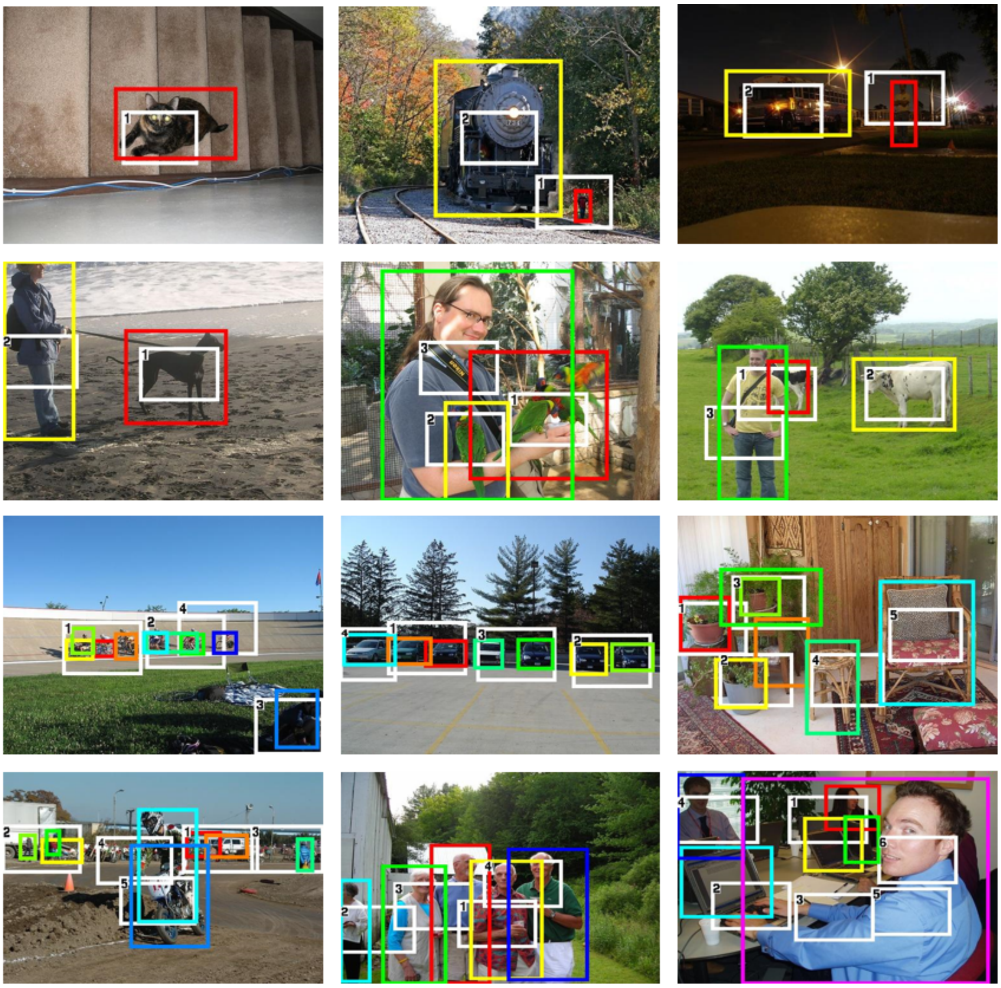


Figure 3.8: Additional visualizations of successful search strategies of our drl-RPN model on the PASCAL VOC 2007 test, c.f. Figure 3.7. Depending on the complexity of a scene, such as the number and layout of objects, the model automatically determines when to stop the search process. The top two rows show examples of short search trajectories, in which only one or a few object instances exist in the image. In contrast, longer trajectories are shown in the bottom two rows, corresponding to images containing more object instances and/or categories. Note how our model is able to adapt its strategy to a variety of situations. For example, in the top-mid image the large train and tiny person are both captured, and in the poorly illuminated image to the top-right both the bus and person are discovered in what appears to be the smallest possible number of fixations. The distance between different fixation locations can vary quite drastically too. An example of this is seen in left image of the third row: the agent moves its view from the cluster of bicycles (fixations 1 - 2) to the barely visible person sitting in the shadow (fixation 3), and then back to investigating the bicycles (fixation 4).

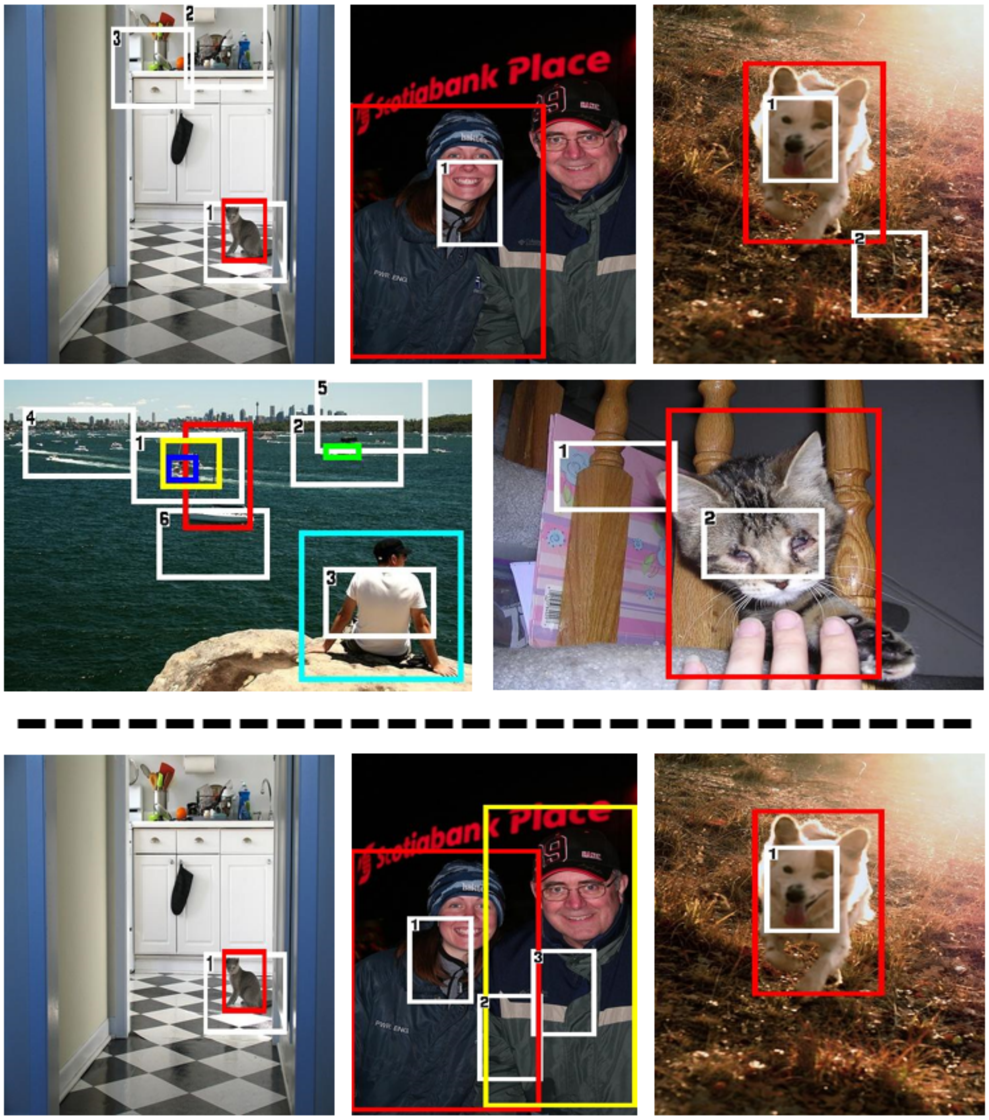


Figure 3.9: Similar visualizations of search strategies as in Figure 3.7 - 3.8; in this case a few examples of slightly less successful and/or unexpected search trajectories are shown (above dashed line) on the PASCAL VOC 2007 test. The model sometimes appears to do one or a few additional, unnecessary fixations (such as in the image of the dog to the top-right, in which the dog is detected already at the first fixation). It may also be the case that the additional fixations occur at locations which are object-like in a more generic sense ('stuff'). An example of this can be seen in the image of the cat to the top-left, with two additional fixations at the kitchen counter, which contains several items that are not labeled in the training data. Similarly, on the mid-left the agent searches among all the tiny boats in the distance, of which only a few are detected in the end. Finally, the model may occasionally stop the search too early, as is apparent in the top-mid figure, in which the man to the right is not detected. Below the dashed line are the corresponding top images where the stopping condition has manually been altered to perform more/less fixations.

Paper III

Domes to Drones: Self-Supervised Active Triangulation for 3D Human Pose Reconstruction

Aleksis Pirinen[1*] **Erik Gärtner**[1*] **Cristian Sminchisescu**[1,2]
[1]Centre for Mathematical Sciences, Lund University
[2]Google Research

Abstract

Existing state-of-the-art estimation systems can detect 2d poses of multiple people in images quite reliably. In contrast, 3d pose estimation from a single image is ill-posed due to occlusion and depth ambiguities. Assuming access to multiple cameras, or given an *active* system able to position itself to observe the scene from multiple viewpoints, reconstructing 3d pose from 2d measurements becomes well-posed within the framework of standard multi-view geometry. Less clear is what is an informative set of viewpoints for accurate 3d reconstruction, particularly in complex scenes where people are occluded by others or scene objects. In order to address the view selection problem in a principled way, we here introduce *ACTOR*, an *active triangulation agent for 3d human pose reconstruction*. Our fully trainable agent consists of a 2d pose estimation network (any of which would work) and a deep reinforcement learning-based policy for camera viewpoint selection. The policy predicts observation viewpoints, the number of which varies adaptively depending on scene content, and the associated images are fed to an underlying pose estimator. Importantly, training the view selection policy requires *no annotations* – given a pre-trained 2d pose estimator, ACTOR is trained in a self-supervised manner. In extensive evaluations on complex multi-people scenes filmed in a Panoptic dome, under multiple viewpoints, we compare our active triangulation agent to strong multi-view baselines, and show that ACTOR produces significantly more accurate 3d pose reconstructions. We also provide a proof-of-concept experiment indicating the potential of connecting our view selection policy to a physical drone observer.

*Denotes equal contribution, order determined by coin flip.

1 Introduction

Estimating 2d and 3d human pose from *given* images or video is an active research area, with deep learning playing a prominent role in most of today's state-of-the-art pose and shape estimation models [1, 2, 3, 4, 5, 6, 7]. Monocular 3d pose estimation is however ill-posed [8] due to depth ambiguities, and these cannot always be resolved by priors or by increasing a feedforward model's predictive power. Given access to multiple cameras, or given an *active* observer which can capture images from multiple viewpoints, reconstructing 3d pose from 2d estimates however becomes tractable within the framework of standard multi-view geometry. An active setup for triangulating 2d estimates also addresses many common practical issues, such as partial observability due to occlusion, either self-induced or due to other people or objects.

Given sufficiently many viewpoints, 3d pose reconstructions from 2d estimates can be made robust and accurate, and such results have even been used as (pseudo-)ground truth [9, 10]. While inferring 3d reconstructions from tens or hundreds of viewpoints works in carefully constructed setups, it is not always practical or desirable to rely on so many cameras. In this work we take a different approach, introducing *ACTOR*, an *active triangulation agent for obtaining 3d human pose reconstructions*. ACTOR consists of a 2d pose (human body joints) estimation network – any of which could be used – and a deep reinforcement learning-based policy for observer (i.e. camera location and pose) prediction, within a fully trainable system. Instead of operating exhaustively over all cameras, ACTOR is able to select a much smaller set of cameras, yet still produces accurate 3d pose reconstructions.

Our proposed methodology is implemented in the Panoptic multi-view framework [10], where the scene can be observed in time-freeze, from a dense set of viewpoints, and over time, providing a proxy for an active observer. In evaluations using Panoptic we show that our system learns to select camera locations that yield more accurate 3d pose reconstructions compared to strong multi-view baselines. We also provide a proof-of-concept experiment indicating the potential of connecting ACTOR to a physical drone observer. Training our policy for view selection requires no 2d or 3d pose annotations – given a pre-trained 2d pose estimator, ACTOR can be trained in a self-supervised manner.

2 Related Work

In addition to recent literature focusing on extracting 3d human representations from a single image or video [11, 2, 12, 1, 13, 3, 4, 5, 14], a parallel line of work concentrates on lifting 2d estimates to 3d. Chen et al. [15] present an unsupervised approach for recovering 3d human pose from 2d estimates in single images. This is achieved by a self-consistency loss based on a lift-reproject-lift process, relying on a network that discriminates between real and fake 2d poses in the reprojection step. Related ideas based on an adversarial framework [16] are also pursued in [17]. A self-supervised learning methodology for monocular 3d human pose estimation is described in [18]. During training, the system leverages multi-view

2d pose estimates and epipolar geometry to obtain 3d pose estimates, which are then used to train the monocular 3d pose prediction system. These weakly-supervised methods for monocular 3d pose estimation eliminate the need for expensive 3d ground truth annotations but tend to not be as accurate as their fully-supervised counterparts.

Multi-view frameworks can, on the other hand, rely on triangulation in order to obtain accurate 3d pose reconstructions given 2d estimates. In contrast to methods performing exhaustive fusion over all cameras, ACTOR actively selects a smaller subset of viewpoints over which to triangulate. Our approach can be considered as a generalization of next-best-view (NBV) selection, and is superficially similar to other NBV-works [19, 20, 21, 22, 23, 24, 25]. Differently from them, the number of viewpoints explored by our agent varies adaptively based on the complexity of the scene. Also, NBV approaches typically decide the next view by greedily and locally evaluating some hand-crafted utility function exhaustively over a set of candidates – we instead frame the task as a deep RL problem where the policy is trained to maximize an explicit *global objective*, searching over entire sequences of viewpoints, and by triangulating as many joints as possible. In a broader sense, ACTOR relates to work on active agents trained to perform various tasks in 3d environments [26, 27, 28, 29, 30, 31]. We are not aware of any prior work that tackles the problem of active triangulation in multi-view setups.

3 Human Pose Reconstruction from Active Triangulation

We here describe the terminology and concepts of 3d human pose reconstruction from active triangulation. The proposed framework is applicable to any number of people as we aim for a system able to actively reconstruct *all* people in the scene, the number of which may vary. We study the active triangulation problem in the CMU Panoptic multi-camera framework [10] since its data consists of real videos of people and allows for reproducible experiments. The subjects are filmed by densely positioned time-synchronized HD-cameras as they perform movements ranging from basic pose demonstrations to different social interactions. Panoptic offers 2d and 3d joint annotations, but as we will show no such annotations are required for training our viewpoint selection system. See Figure 4.1 for an overview of our active 3d human pose reconstruction model.

Terminology. Triangulation of 3d pose reconstructions from 2d estimates requires observing the targets from several cameras, each capturing an image v_i^t (referred to as a *view* or *viewpoint*) indexed by time-step t and camera i. The set $\{v_1^t, \ldots, v_N^t\}$ of all views in a time-step t is called a *time-freeze*. A subset of these is an *active-view*, $\mathcal{V}^t = \{v_1^t, \ldots, v_k^t\}$, which contains k cameras selected (by some agent or heuristic) from the time-freeze at time t. A sequence of temporally contiguous active-views is referred to as an *active-sequence*, $\mathcal{S}^{1:T} = \{\mathcal{V}^1, \mathcal{V}^2, \ldots, \mathcal{V}^T\}$, where T is its length. Unless the context requires both indices we will omit the time superscript t to simplify notation, which implies that all elements belong to the same time step. The set of all predicted 2d pose estimates corresponding to

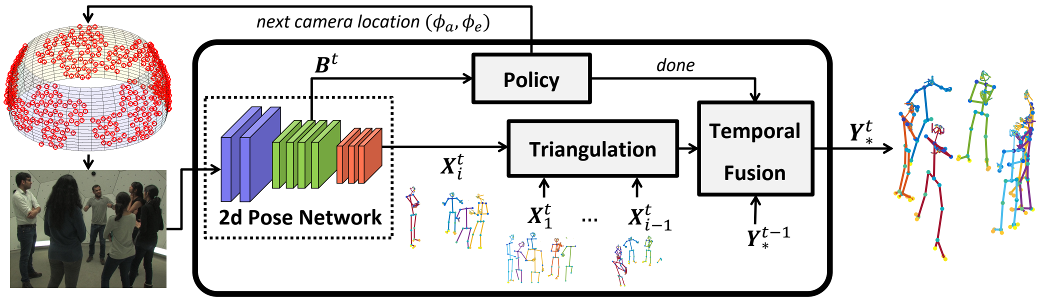


Figure 4.1: Overview of *ACTOR*, our proposed active triangulation agent for 3d human pose reconstruction. A random view is initially given and the image is fed to a 2d body joint predictor, which produces estimates for all visible people ($\boldsymbol{X}_i^t$) and the core of the agent's state ($\boldsymbol{B}^t$). The policy network predicts camera locations until all joints have been triangulated and then switches to the next active-view at time $t+1$. A predicted camera location is encoded via spherical angles relative to the agent's location on the viewing sphere, and the closest camera is selected. When the agent is done it outputs $\boldsymbol{Y}_\star^t$, the final 3d reconstructions of all people in the scene, which are obtained by a combination of spatial fusion (triangulation of 2d poses $\boldsymbol{X}_1^t, \dots, \boldsymbol{X}_k^t$) and temporal fusion with $\boldsymbol{Y}_\star^{t-1}$ on a per-joint basis. As described in Section 4.2, we train the viewpoint selection policy using self-supervision.

a view v_i is denoted $\boldsymbol{X}_i = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_M] \in \mathbb{R}^{30 \times M}$, where $\boldsymbol{x}$ denotes a single 2d pose estimate, based on detecting 15 human body joints, and M is the number of people observed from that viewpoint.

Task description. *Active triangulation for 3d human pose reconstruction* is the task of producing active-views with corresponding accurate fused 3d pose reconstructions for all people present, $\boldsymbol{Y}_\star = [\boldsymbol{y}_{1\star}, \dots, \boldsymbol{y}_{M\star}]$, given 2d pose estimates $\boldsymbol{X}_1, \dots, \boldsymbol{X}_k$ associated with the active-view. These active-views then form an active-sequence of accurate 3d pose reconstructions. As it is challenging to select appropriate viewpoints for satisfactory triangulation, especially in crowded scenes where people are often occluding each other, the task is considered completed once each individual's joint has been observed from at least two different viewpoints (the minimum requirement for performing a triangulation), or after a given exploration budget is exceeded.

Matching and triangulating people. The active triangulation system must tackle the problems of tracking and identifying people across various views and through time. The agent receives appearance models[1] for the different people at the beginning of an active-sequence. For each view, the agent compares the people detected by the 2d pose estimator with the given appearance models and matches them across space and time using the Hungarian algorithm. To reconstruct 3d poses from 2d estimates associated with the selected view-

[1]Instance-sensitive features generated using a VGG-19 based [32] siamese instance classifier, trained with a contrastive loss to differentiate people on the training set. More details are found in the supplement.

points, we compute triangulation between each pair of viewpoints [33, 34] and perform per-body-joint fusion (averaging) of the associated 3d reconstructions. More sophisticated triangulation methods would be possible; here we selected pairwise averaging due to computational efficiency which is important during training.

4 Active Triangulation Agent

We now introduce our active triangulation agent, ACTOR, and describe its state representation and action space in Section 4.1. In Section 4.2 we describe the *annotation-free* reward signal for training ACTOR to efficiently triangulate the joints of all people.

In the first active-view $\mathcal{V}^1$, the agent is given an initial random view v_1^1. It then predicts camera locations $v_2^1, \dots, v_k^1$ until the active-view is completed. An active-view is considered completed once the agent has triangulated the joints of all people within the time-freeze, or after a given exploration budget has been exceeded. The 2d pose estimator is computed for images collected at every visited viewpoint v_i^t, yielding estimates $\boldsymbol{X}_i^t$ for all visible people. Camera locations are specified by the relative azimuth and elevation angles (jointly referred to as spherical angles) on the viewing sphere.

Once the agent has triangulated the joints of all people within a time-freeze, it continues to the next active-view $\mathcal{V}^{t+1}$. At this time the triangulated 3d pose reconstructions $\boldsymbol{Y}^t$ are temporally fused with the reconstructions $\boldsymbol{Y}_\star^{t-1}$ from the previous active-view, $\boldsymbol{Y}_\star^t = f(\boldsymbol{Y}_\star^{t-1}, \boldsymbol{Y}^t)$. As the 2d pose estimator we use in this work is accurate we have opted for a straightforward temporal fusion. We define $I = I_{\text{tri}} \cup I_{\text{miss}}$, where I indexes all joints, I_{tri} indexes the successfully triangulated joints in the current time-step, and I_{miss} indexes joints missed in the current time-step. Then we set $\boldsymbol{Y}_\star^t[I_{\text{tri}}] = \boldsymbol{Y}^t[I_{\text{tri}}]$ for the joints that were successfully triangulated in the current time-step, and $\boldsymbol{Y}_\star^t[I_{\text{miss}}] = \boldsymbol{Y}_\star^{t-1}[I_{\text{miss}}]$. Hence we temporally propagate from the previous time-step only those joint reconstructions that were missed in the current time-step. The initial viewpoint v_1^{t+1} for $\mathcal{V}^{t+1}$ is set to the final viewpoint v_k^t of $\mathcal{V}^t$, i.e. $v_1^{t+1} = v_k^t$. The process repeats until the end of the active-sequence. Figure 4.1 shows a schematic overview of ACTOR.

4.1 State-Action Representation

In this section, while describing the state and action representations, we will assume that the agent acts in a single time-freeze. This allows us to simplify notation and index steps within the active-view by t. The state is represented as a tuple $s^t = (\boldsymbol{B}^t, \boldsymbol{C}^t, \boldsymbol{u}^t)$, where $\boldsymbol{B}^t \in \mathbb{R}^{H \times W \times C}$ is the deep feature map from the 2d pose estimator. $\boldsymbol{C}^t \in \mathbb{N}^{w \times h \times 2}$ is a *camera history*, which encodes[2] the previously visited cameras on the rig. It also contains a representation of the distribution of cameras on the rig. Finally, the auxiliary array $\boldsymbol{u}^t \in$

[2]The camera history consists of w bins in the azimuth direction and h bins in the elevation direction. It is agent-centered, i.e. relative to the agent's current viewpoint. We set $w = 9$ and $h = 5$.

$\mathbb{R}^{17}$ contains the number of actions taken, the number of people detected, as well as a binary vector indicating which joints have been triangulated for all people.

A deep stochastic policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{c}^t|s^t)$ parametrized by $\boldsymbol{\theta}$ is used to predict the next camera location $\boldsymbol{c}^t = (\phi_a^t, \phi_e^t)$, were (ϕ_a^t, ϕ_e^t) is the azimuth-elevation angle pair encoding the camera location. To estimate the camera location probability density, the base feature map $\boldsymbol{B}^t$ is processed through two convolutional blocks. The output of the second convolutional block is concatenated with $\boldsymbol{C}^t$ and $\boldsymbol{u}^t$ and fed to the policy head, which consists of three fully connected layers with $\tanh$ activations.

As the policy predicts spherical angles, we choose to sample these from the periodical von Mises distribution. We use individual distributions in the azimuth and elevation directions. The probability density function for the azimuth angle is given by

$$\pi_{\boldsymbol{\theta}}\left(\phi_a^t|s^t\right) = \frac{1}{2\pi I_0(m_a)} \exp\{m_a \cos(\phi_a^t - \tilde{\phi}_a(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a))\}, \tag{4.1}$$

where the zeroth-order Bessel function I_0 normalizes (4.1) to a probability distribution on the unit circle. Here $\tilde{\phi}^a$ is the mean of the distribution (parameterized by the deep network), m_a is the precision parameter,[3] and $\boldsymbol{w}_a$ and b_a are trainable weights and bias, respectively. The second to last layer of the policy head outputs $\boldsymbol{z}_a^t$. For the azimuth prediction, the support is the full circle. Therefore we set

$$\tilde{\phi}_a(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a) = \pi \tanh(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a). \tag{4.2}$$

The probability density for the elevation prediction has the same form (4.1) as the azimuth. As there are no cameras below the ground-plane of the rig, nor cameras directly above the people (cf. Figure 4.1), we limit the elevation angle range to $[-\kappa, \kappa]$, where $\kappa = \pi/6$. Thus the mean elevation angle becomes

$$\tilde{\phi}_e(\boldsymbol{w}_e^\top \boldsymbol{z}_e^t + b_e) = \kappa \tanh(\boldsymbol{w}_e^\top \boldsymbol{z}_e^t + b_e). \tag{4.3}$$

4.2 Reward Signal for Self-Supervised Active Triangulation

As explained in Section 4, ACTOR predicts camera locations until the individual body joints of all people have been detected from at least two different views (minimum requirement for 3d triangulation) or after a given exploration budget B is exceeded; we set $B = 10$ during training. We use the indicator variable d_t to denote whether or not the agent has triangulated all joints ($d_t = 1$ if all joints have been triangulated). We want to encourage the agent to fulfill the task while selecting as few camera locations as possible, which gives rise to the reward design in (4.4) below. Note that our reward is *not* based on ground truth

[3]The precision parameters m_a and m_e are treated as constants, but we anneal them over training as the policy becomes better at predicting camera locations.

pose annotations – it relies solely on automatic 2d pose (body joint) detections.

$$r^t = \begin{cases} -\beta/M, & \text{if } d_t = 0, t < B \text{ and camera not already visited} \\ -\beta/M - \epsilon, & \text{if } d_t = 0, t < B \text{ and camera already visited} \\ 1, & \text{if } d_t = 1, t \leq B \\ \tau_{min}, & \text{if } d_t = 0, t = B. \end{cases} \tag{4.4}$$

The first and second rows of (4.4) reflect intermediate rewards, where the agent receives a penalty ϵ (we set $\epsilon = 2.5$) if it predicts a previous camera location. To encourage efficiency the agent also receives a time-step penalty β for not yet having completed the triangulation (β is set to 0.2). This penalty is normalized by the number of people M for scaling purposes, as we expect more cameras are typically required to triangulate multiple people. The third and fourth rows represent rewards the agent obtains at the end of the active-view. It receives $+1$ if it triangulates the joints of all M persons within its exploration budget B. The fourth row defines the reward if the agent fails to triangulate some joints within the exploration budget. It then receives the minimum fraction of covered joints for any person, τ_{min}.

Policy gradients are used to learn ACTOR's policy parameters, where we maximize expected cumulative reward on the training set with the objective $J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=1}^{|\boldsymbol{s}|} r^t\right]$, where $\boldsymbol{s}$ denotes state-action trajectories. This objective function is approximated using REINFORCE [35].

5 Experiments

Dataset. We consider both multi-people scenes (named *Mafia* and *Ultimatum* in Panoptic) and single-people ones (*Pose*). The scenes with multiple people are expected to be particularly challenging for the agent, as occlusions are common. Panoptic data comes as 30 FPS time-synchronized videos. To make the size more manageable and increase movement between frames we downsample the data to 2 FPS. We use the HD cameras, of which there are about 30 per scene, since they provide better image quality than VGA and are sufficiently dense, yet spread apart far enough to make each viewpoint unique. We select 20 scenes (343k images) which are split randomly into training, validation and test sets with 10, 4, and 6 scenes, respectively. There is no overlap of scenes between the three sets, which forces the agent to learn a fairly general policy.

Implementation details. ACTOR is implemented on top of the OpenPose 2d pose estimation system [6], though any 2d pose predictor would work. As described in Section 4, temporal fusion of 3d reconstructions across active-views ensures that missed joints are instead covered by the associated estimates from an earlier point in time. In case there is no previous estimate for a missing joint, it is set to the average of the successfully triangulated ones (to be able to compute errors). We use per-joint median averaging for fusing 3d pose reconstructions across views and temporal steps.

Training. We train the policy network with batches consisting of experiences from 5 active-sequences, each of length 10. Adam [36] is used for parameter updates. We normalize cumulative rewards for each episode to zero mean and unit variance over each batch to reduce variance in the policy updates. The exploration budget B (maximum trajectory length) is set to 10 camera locations per active-view, including the initial camera. The policy is trained for 75k episodes with learning rate initially set to $5 \cdot 10^{-7}$, which is halved after 720k steps and again after 1440k steps. The precision parameters (m_a, m_e) of the von Mises distributions are linearly annealed from $(1, 10)$ to $(25, 50)$ during training, which makes the camera prediction increasingly deterministic as the training progresses.

Baselines. We evaluate ACTOR against several multi-view baselines. They use the same 2d pose estimator, matching algorithm, triangulation method and temporal fusion. All methods receive the same initial random camera at the start of an active-sequence. We compare to the following baselines: i) *Random:* Selects random cameras (it never selects the same camera twice); *Max-Azim:* The first three views are selected at 90, 180 and 270 degrees azimuth relative to the initial view, so the four first views are at 90 degrees azimuth from each other. The subsequent four views are also selected at 90 degrees azimuth from each other, but at a 45 degree azimuth offset relative to the first four views. At each azimuth, it samples a random elevation angle. The last two cameras are selected randomly, and we ensure each camera is different. This baseline produces a wide coverage of the viewing sphere without the need to know in advance how many cameras will be selected; iii) *Oracle:* Before selecting the next camera, this baseline computes the improvement in 3d pose reconstruction error associated with all available cameras. It then selects the camera that maximally decreases the error. In addition to cheating when it selects views, the oracle is also impractically slow since it exhaustively computes errors for all cameras in each step. Thus it is only shown as a gold standard.

5.1 Main Results

Our ACTOR agent is compared to the baselines on the Panoptic test sets on active-sequences consisting of 10 active-views. We train ACTOR with 5 different random network initializations and report average results with standard errors of the means (we early stop training separately for each network initialization based on errors on the validation set). For the non-deterministic heuristic baselines (*Random* and *Max-Azim*; the oracle is deterministic) we report results across 5 seeds, including standard errors of the means. In Table 4.1 we report 3d pose reconstruction errors for auto stopping and for a fixed number of views. ACTOR is more accurate and uses fewer cameras on average, compared to the heuristics.

Figure 4.2 shows 3d pose reconstruction error versus number of views. ACTOR significantly outperforms the heuristic baselines, especially for complex multi-person scenes

Table 4.1: Mean 3d reconstruction error (mm/joint) for ACTOR and baselines on the Panoptic test sets. *Multi* denotes multi-people data (union of *Mafia* and *Ultimatum*); *single* is the single-person *Pose* split. We show total errors which include translation errors (top) and hip-aligned errors (bottom). Columns indicate the number of cameras inspected, ranging from 2 to 10. We also show results for auto-mode, where camera location selection terminates when the joints of all people have been triangulated, but using 10 cameras at most. For this column we also show the average number of cameras inspected in parentheses. ACTOR outperforms both the heuristic baselines on all types of scenes. The advantage of a trained system is most pronounced for complex multi-people scenes where selecting informative viewpoints is important. ACTOR-ob and ACTOR-ntf denote ablated versions of our agent, cf. Section 5.2.

Model	Data	Auto	2	3	4	5	6	7	8	9	10
ACTOR	multi	125.6 (8.84) 96.2 (8.84)	502.4 247.2	281.5 179.3	201.0 146.4	168.4 131.1	151.6 118.5	141.2 111.6	132.1 101.9	126.1 95.2	122.1 92.3
	single	74.6 (4.28) 60.5 (4.28)	172.1 151.3	107.5 92.8	81.9 68.9	71.2 59.4	67.1 55.6	64.9 53.2	63.3 51.3	62.1 49.9	61.3 49.0
ACTOR-ob	multi	148.9 (8.79) 108.1 (8.79)	555.2 299.6	372.9 305.7	276.4 231.2	217.4 182.4	185.2 155.9	166.6 131.9	154.0 119.4	146.1 112.3	142.5 109.3
	single	80.2 (4.58) 67.3 (4.58)	187.3 159.7	122.6 104.4	95.1 77.7	80.6 64.2	72.4 56.6	68.9 53.3	67.4 52.3	67.0 51.8	66.8 51.6
ACTOR-ntf	multi	138.9 (8.84) 102.0 (8.84)	925.7 334.4	565.2 258.0	353.1 198.4	242.8 159.1	196.5 138.4	172.2 124.5	154.8 112.0	143.7 102.9	136.6 98.3
	single	75.9 (4.28) 61.6 (4.28)	274.0 228.1	151.4 132.4	99.6 85.3	79.3 66.9	71.8 59.8	67.9 55.9	65.5 53.3	63.9 51.5	62.7 50.3
Random	multi	142.7 (9.34) 125.9 (9.34)	570.1 347.3	469.9 406.4	316.1 350.1	259.9 278.0	269.3 263.0	238.5 218.8	220.2 196.2	198.8 179.5	188.3 160.0
	single	82.6 (4.90) 68.7 (4.90)	203.6 178.0	139.4 125.7	107.2 93.8	89.9 76.4	81.1 67.6	75.1 61.3	71.0 56.8	67.9 53.4	65.8 51.0
Max-Azim	multi	132.0 (9.01) 102.7 (9.01)	479.3 259.4	375.8 282.1	288.4 235.0	226.0 200.0	195.7 196.8	170.2 158.2	149.2 131.3	137.7 114.1	128.6 103.7
	single	75.5 (4.41) 63.61 (4.41)	185.7 161.2	119.5 106.3	88.0 76.5	79.5 67.7	73.7 61.0	68.8 56.3	64.5 52.0	63.2 50.0	62.1 48.5
Oracle	multi	94.5 (6.67) 74.0 (6.67)	254.4 163.1	147.6 110.3	113.1 89.2	98.2 78.8	90.3 72.8	86.4 69.0	84.1 66.4	82.8 64.5	81.9 63.0
	single	54.0 (2.97) 48.1 (2.97)	123.0 108.2	60.2 54.5	49.2 43.3	45.3 39.5	43.6 37.5	42.8 36.2	42.3 35.2	42.2 34.6	42.4 34.2

(e.g. 103 and 78 mm/joint improvements over *Max-Azim* at 3 and 6 cameras, respectively). Multi-people scenes are more difficult to analyze due to occlusions and thus require intelligent view selection – one clearly sees the advantages of a learned system in such scenarios.

Figure 4.3 shows how the exploration budget B (max number of views) affects 3d reconstruction error. At smaller budgets ACTOR's improvements over the heuristic baselines are even larger, which shows that our trained system is significantly more efficient at finding good views over which to triangulate the body joints. Runtimes versus number of cameras are shown in column 3 of Figure 4.2 and versus number of people in column 3 of Figure 4.3. OpenPose runs at about 0.134 seconds per image, while the policy network adds an overhead of 0.005 seconds per action, which is negligible compared to the 2d pose

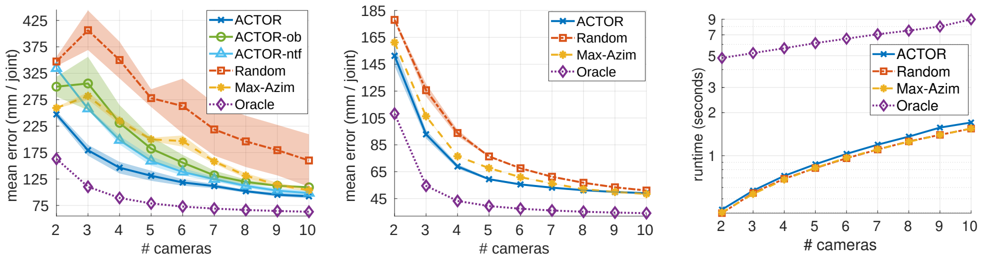


Figure 4.2: Columns 1 - 2: Mean 3d reconstruction error per joint vs number of cameras on the test sets (means and standard errors over 5 seeds). Column 1: Multi-people data. Column 2: Single-people data. ACTOR decreases errors faster than the heuristic baselines, particularly for multi-people data with occlusions. The oracle uses 3d ground truth and is shown as a gold standard. ACTOR also outperforms the ablated variants ACTOR-ob and ACTOR-ntf, cf. Section 5.2. Ablated models are not plotted in the single-people setting to avoid clutter – see Table 4.1 for these results. Column 3: Runtime (log-scale) per active-view vs number of cameras for a 3-people scene. The oracle computes errors for all views and persons to select its next camera, so it very slow.

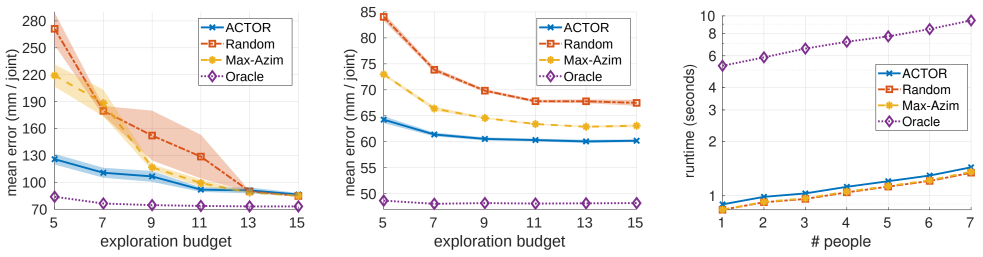


Figure 4.3: Columns 1 - 2: Mean 3d reconstruction error per joint vs exploration budget B (maximum number of cameras) on the test sets. As mentioned in Section 4.2, ACTOR was trained solely at budget $B = 10$. Column 1: Multi-people data. Column 2: Single-people data. The relative gain to the baselines is higher at smaller exploration budgets (e.g. 93 mm/joint improvement over *Max-Azim* on multi-people data at $B = 5$), where the system quickly needs to select cameras triangulating the joints of all people. The accuracy curves are flatter for single-people data, as in general the systems need fewer cameras to triangulate the joints of a single person – the models hence tend to stop before their budget is exhausted. Column 3: Runtime (log-scale) when varying the number of people in a scene while keeping the the number of selected cameras constant at 6 per active-view.

estimator. For visualizations[4] of ACTOR operating in various scenes, see Figure 4.5 - 4.6.

5.2 Ablation Studies

In this section we study how ACTOR is affected by i) removing all state features except the deep feature blob $\boldsymbol{B}^t$ (ACTOR-ob; *ob* stands for *only blob*), and ii) using no temporal fusion of 3d pose reconstructions (ACTOR-ntf). Similarly as for the main ACTOR model,

[4]In this case we equip ACTOR with an OpenPose system that estimates detailed faces, hands and feet. We do not refine the pre-trained ACTOR model that was trained using the standard OpenPose estimator.

ACTOR-ob is trained over 5 different network initializations with individual early stopping on the validation set (ACTOR-ntf uses the same parameters as ACTOR but without temporal fusion during inference). The results are shown in Table 4.1 and Figure 4.2 (left). The full ACTOR agent outperforms the ablated variants for all data splits. For multi-people data, ACTOR drastically outperforms ACTOR-ob, which indicates the need for representing earlier visited cameras ($\boldsymbol{C}^t$) as part of the state space. For single-people data, ACTOR-ob is almost as good as ACTOR, but this data is very simple and occlusion-free and does not require too sophisticated camera selection. Finally, the full agent significantly outperforms ACTOR-ntf when operating using few cameras, which makes sense as there is a big risk of the system missing to triangulate some joints, in which case a backup from earlier active-views may help.

5.3 From Domes to Drones

The dense Panoptic multi-camera dome provides an idealization in which we can generate controllable and reproducible experiments. It is also useful for training ACTOR, as we do not actually have to move a camera around. However, in many practical scenarios one does not have access to a multi-view setup and may instead have to resort to a single but moving camera. One such scenario is that of a drone circling a set of people, which aims to reconstruct their 3d poses.

To test ACTOR's drone-controlling capacity, we captured three small scenes where a drone circles around two people performing various poses. We then fine-tuned ACTOR with learning rate 10^{-6} for 3k episodes (15 minutes) on two scenes, keeping all other hyperparameters the same, and ran the model on the third scene. In Figure 4.4, ACTOR selects 5 different views to reconstruct the targets. It should be noted that the setting of this drone experiment differs drastically from that of Panoptic. For example, the drone's camera quality is worse (VGA rather than HD), and the loop generated by the drone has a much smaller radius than Panoptic's viewing sphere (about 1.5 meters for the drone versus about 3 meters for Panoptic), so there are fewer views where e.g. the feet are visible. In future work we plan to more tightly integrate ACTOR in the loop, so as to direct the drone to observe targets from informative views.

6 Conclusions

We have presented *ACTOR*, a deep RL-based agent to actively reconstruct 3d poses from 2d estimates via triangulation. Training the viewpoint selection policy requires no annotations and only uses an off-the-shelf 2d human pose estimator for self-supervision. We evaluated the model in complex scenarios with multiple interacting people and showed that by intelligently selecting informative views the agent outperforms strong multi-view baselines in both speed and accuracy. We also provided proof-of-concept results which indicate that ACTOR can be used in single-camera settings, e.g. to control a physical drone observer.

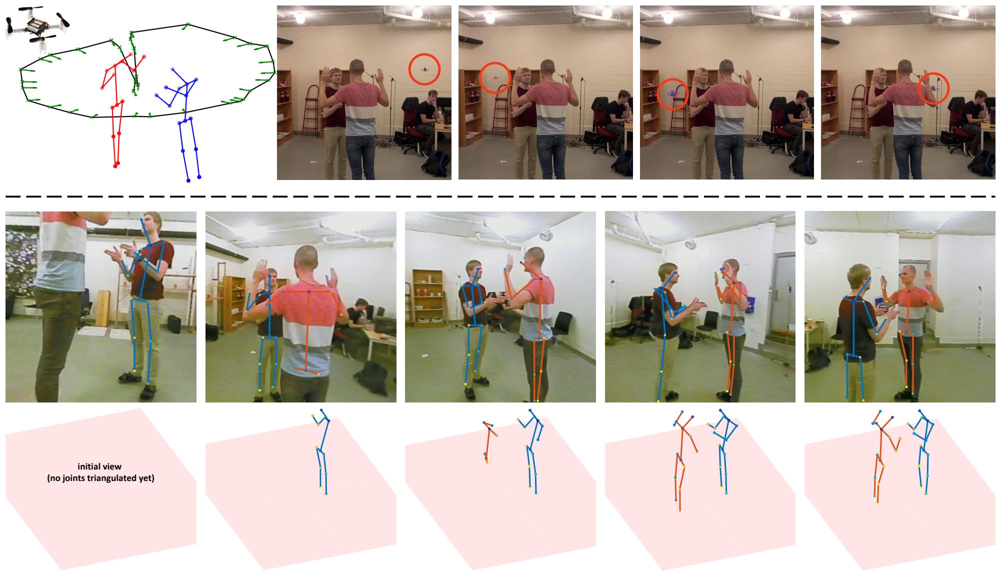


Figure 4.4: Proof-of-concept experiment illustrating that ACTOR can be connected to an active drone observer to reconstruct 3d poses from informative viewpoints. Above the dashed line to the left we show the drone's loop (the sharp peak is due to take-off and landing), with sampled camera locations as green arrows. We also show the 3d pose reconstructions obtained by triangulating from *all* 33 sampled camera locations. The 9-by-9 cm Crazyflie drone used is shown in the very top left corner; it can be used safely due to its small size and weight. Sample locations of the drone are also shown above the line (drone locations are highlighted with red circles in images). Below the line we show views seen by ACTOR and aggregated 3d pose reconstructions. After observing 5 viewpoints, the two bodies are fully 3d reconstructed, with an average 2d reprojection error of 11.5 pixels (averaged over all 33 cameras), significantly better than the exhaustively triangulated reconstructions to the left, with an average reprojection error of 35.4 pixels.

Acknowledgments: This work was supported by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535 and PCCF-2016-0180, the EU Horizon 2020 Grant DE-ENIGMA, Swedish Foundation for Strategic Research (SSF) Smart Systems Program, as well as the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We would also like to thank Patrik Persson for support with the drone experiments.

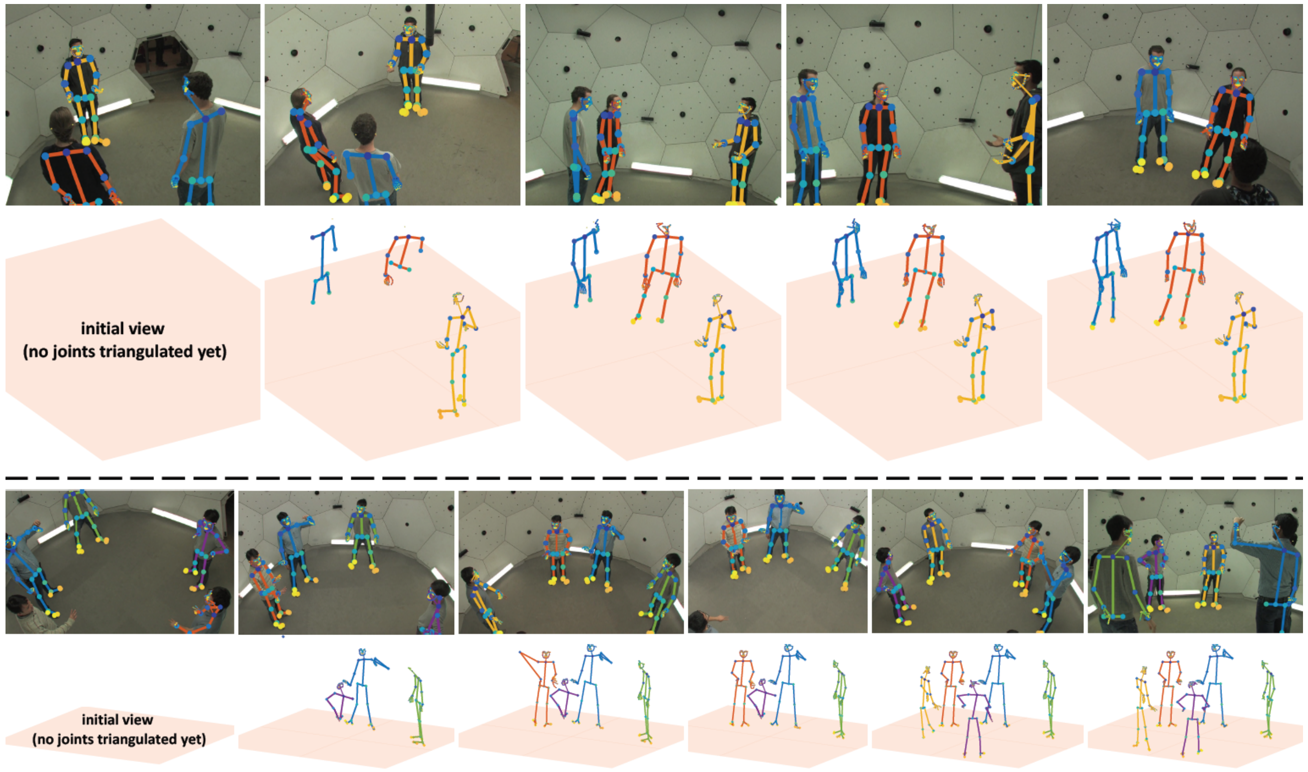


Figure 4.5: ACTOR operating in two different multi-people scenes. Visualizations are shown for initial active-views and thus have no propagated 3d pose estimates from earlier time steps. Each example shows the views selected by ACTOR, including 2d pose estimates (first view randomly given). Below these we show aggregated 3d pose reconstructions. Top: 3-person scene. One of the persons is reconstructed already at the second view; all of them are reconstructed at the fifth view. The mean 3d reconstruction error decreases from 268 to 51 mm/joint between the second and last view. Bottom: 5-person scene, where 3d pose reconstructions improve over the 6 views. The error decreases from 296 to 68 mm/joint.

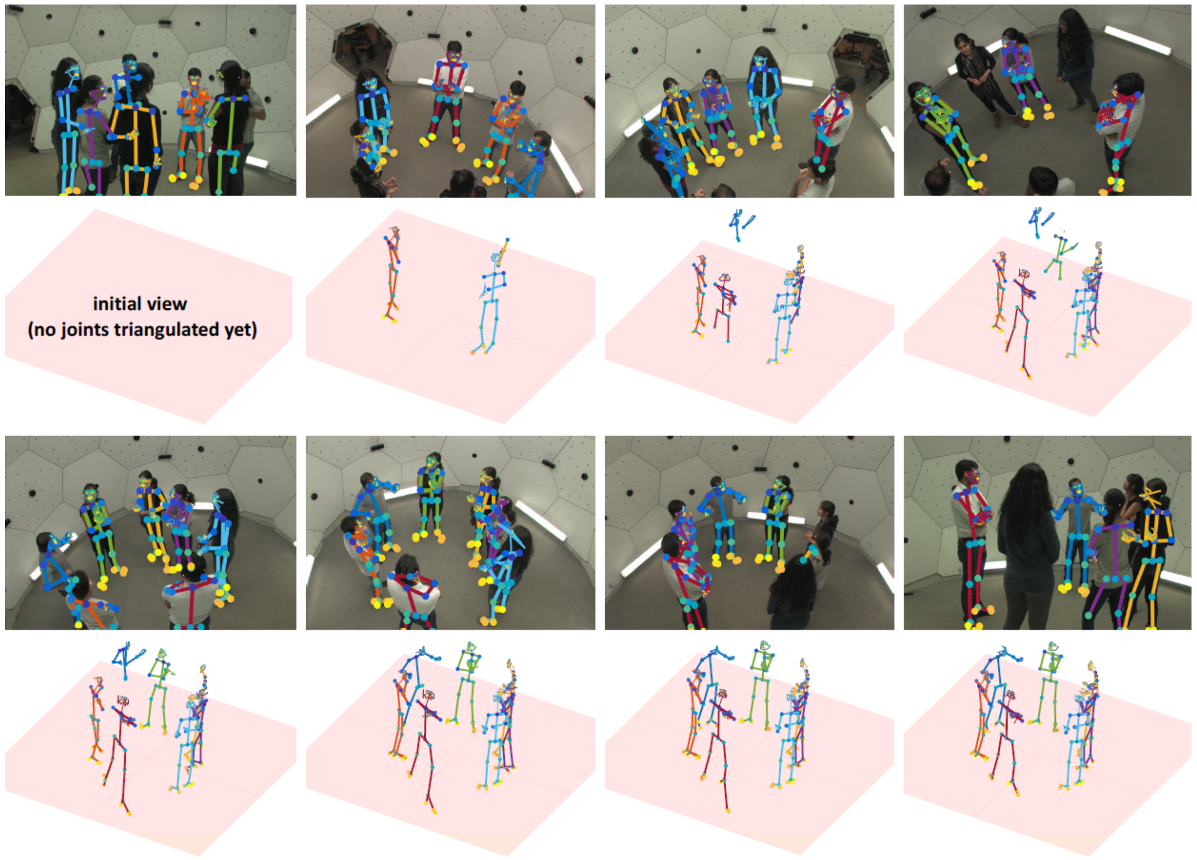


Figure 4.6: ACTOR operating in a 6-person scene where people stand quite close to each other, which makes it difficult to triangulate all joints due to occlusions. ACTOR observes the scene from 8 diverse views, and the error decreases from 342 to 69 mm/joint.

References

[1] H. Rhodin, N. Robertini, D. Casas, C. Richardt, H.-P. Seidel, and C. Theobalt, "General automatic human shape and motion capture using volumetric contour cues," in *ECCV*, 2016.

[2] A.-I. Popa, M. Zanfir, and C. Sminchisescu, "Deep multitask architecture for integrated 2d and 3d human sensing," in *CVPR*, 2017.

[3] A. Zanfir, E. Marinoiu, and C. Sminchisescu, "Monocular 3d pose and shape estimation of multiple people in natural scenes–the importance of multiple scene constraints," in *CVPR*, 2018.

[4] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, "Vnect: Real-time 3d human pose estimation with a single rgb camera," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, 2017.

[5] B. Tekin, P. Márquez-Neila, M. Salzmann, and P. Fua, "Learning to fuse 2d and 3d image cues for monocular body pose estimation," in *ICCV*, 2017.

[6] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *CVPR*, 2017.

[7] A. Arnab, C. Doersch, and A. Zisserman, "Exploiting temporal context for 3d human pose estimation in the wild," in *CVPR*, 2019.

[8] C. Sminchisescu and B. Triggs, "Building Roadmaps of Minima and Transitions in Visual Models," *IJCV*, vol. 61, no. 1, 2005.

[9] Z. Yu, J. S. Yoon, P. Venkatesh, J. Park, J. Yu, and H. S. Park, "Humbi 1.0: Human multiview behavioral imaging dataset," *arXiv preprint arXiv:1812.00281*, 2018.

[10] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *ICCV*, 2015.

[11] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[12] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it SMPL: Automatic estimation of 3d human pose and shape from a single image," in *ECCV*, 2016.

[13] G. Rogez, P. Weinzaepfel, and C. Schmid, "Lcr-net: Localization-classification-regression for human pose," in *CVPR*, 2017.

[14] A. Zanfir, E. Marinoiu, M. Zanfir, A.-I. Popa, and C. Sminchisescu, “Deep network for the integrated 3d sensing of multiple people in natural images,” in *NeurIPS*, 2018.

[15] C.-H. Chen, A. Tyagi, A. Agrawal, D. Drover, R. MV, S. Stojanov, and J. M. Rehg, “Unsupervised 3d pose estimation with geometric self-supervision,” in *CVPR*, 2019.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *NeurIPS*, 2014.

[17] D. Drover, C.-H. Chen, A. Agrawal, A. Tyagi, C. P. Huynh, *et al.*, “Can 3d pose be learned from 2d projections alone?,” in *ECCV*, Springer, 2018.

[18] M. Kocabas, S. Karagoz, and E. Akbas, “Self-supervised learning of 3d human pose using multi-view geometry,” in *CVPR*, 2019.

[19] J. I. Vasquez-Gomez, L. E. Sucar, R. Murrieta-Cid, and E. Lopez-Damian, “Volumetric next-best-view planning for 3d object reconstruction with positioning error,” *IJARS*, vol. 11, no. 10, 2014.

[20] S. Haner and A. Heyden, “Covariance propagation and next best view planning for 3d reconstruction,” in *ECCV*, 2012.

[21] D. Jayaraman and K. Grauman, “Learning to look around: Intelligently exploring unseen environments for unknown tasks,” in *CVPR*, 2018.

[22] B. Xiong and K. Grauman, “Snap angle prediction for 360 panoramas,” in *ECCV*, 2018.

[23] E. Johns, S. Leutenegger, and A. J. Davison, “Pairwise decomposition of image sequences for active multi-view recognition,” in *CVPR*, 2016.

[24] D. Jayaraman and K. Grauman, “Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion,” in *ECCV*, 2016.

[25] X. Han, Z. Zhang, D. Du, M. Yang, J. Yu, P. Pan, X. Yang, L. Liu, Z. Xiong, and S. Cui, “Deep reinforcement learning of volume-guided progressive view inpainting for 3d point scene completion from a single depth image,” in *CVPR*, 2019.

[26] P. Ammirato, P. Poirson, E. Park, J. Košecká, and A. C. Berg, “A dataset for developing and benchmarking active vision,” in *ICRA*, 2017.

[27] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi, “Iqa: Visual question answering in interactive environments,” in *CVPR*, 2018.

[28] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, “Embodied question answering,” in *CVPR*, 2018.

[29] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *CVPR*, 2018.

[30] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*, 2017.

[31] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *CoRL*, 2017.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[33] R. I. Hartley and P. Sturm, "Triangulation," *Computer Vision and Image Understanding*, vol. 68, no. 2, 1997.

[34] M. Lourakis, "Stereo triangulation." `https://www.mathworks.com/matlabcentral/fileexchange/67383-stereo-triangulation`, Nov 2018. Retrieved May 22, 2019.

[35] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

A Supplementary Material

This supplementary material provides more insights to our ACTOR model and experimental setup. Section $A.1$ describes the details of the network architecture, implementation, and hyperparameters. Section $A.2$ elaborates on how we match 2d pose estimates in space and time using instance features. In Section $A.3$ we provide 2d reprojection errors onto 2d OpenPose [1] estimates on the Panoptic test splits. Finally, Section $A.4$ describes further dataset details.

A.1 Model Architecture

See Figure 4.7 for a full description of the ACTOR network architecture. ACTOR was implemented in Caffe [2] and MATLAB. We used an open-source TensorFlow [3] implementation of OpenPose.[5] All code and pre-trained weights have been made publicly available.[6]

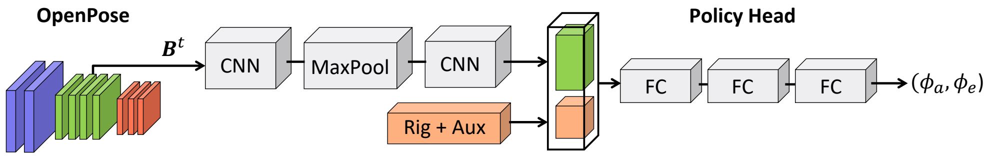


Figure 4.7: ACTOR policy architecture. A multi-people 2d pose estimation system (here OpenPose, but any similar deep system would would work) processes an input image. The deep feature maps $\boldsymbol{B}^t$ produced by OpenPose (`conv4_4_CPM`) is fed into the ACTOR policy network and is processed by two convolutional layers with ReLU-activations. The first and second convolutional layers both have 3×3 kernels with stride 1. Their output dimensions are $8 \times 39 \times 21$ and $4 \times 18 \times 9$, respectively. The max pooling layer has a 2×2 kernel with stride 2. The output from the second convolutional layer is then concatenated with agent-centric camera rig information about previously visited cameras relative to current position (*Rig*), and auxiliary information such as the number of joints triangulated and number of people detected in the view (*Aux*). The flattened and concatenated data is then fed into three fully connected layers with tanh-activations with 1024, 512 and 2 neurons respectively. The final output is scaled by two constants to produce radial angles on the viewing sphere.

A.1.1 Hyperparamters

Hyperparameter search was performed using two powerful workstations equipped with several NVIDIA Titan V100 GPU:s. Training a single model for 40k episodes took about 32 hours using one GPU and to speed up results while searching for optimal hyperparameters we trained several model configurations in parallell using Hyperdock [4]. The most important parameters for training ACTOR were learning rate, precision of the the von Mises

[5] `https://gist.github.com/alesolano/b073d8ec9603246f766f9f15d002f4f4`

[6] `https://github.com/ErikGartner/actor`

Table 4.2: The values tested for the most important hyperparameters when training ACTOR. The final and best values are highlighted in bold. For the von Mises precisions, the arrow indicates linear annealing performed during training (e.g. from $(m_a, m_e) = (1, 10)$ to $(m_a, m_e) = (25, 50)$ for the best configuration).

Hyperparameter	Attempted values
Learning Rate	{1e-7, **5e-7**, 1e-6, 5e-6}
von Mises precision	{**(1, 10) $\rightarrow$ (25, 50)**, (10, 50) $\rightarrow$ (20, 100), (10,50) $\rightarrow$ (100, 500)}

(m_a, m_e), and the annealing rate of the precision. See Table 4.2 for a summary of the values tested for these hyperparameters. In total we trained around 10 different versions of the final model with varying hyperparameters and evaluated each of them on the validation set. Finally, the best model was evaluated on the test set and retrained with four additional random seeds to measure the model's sensitivity to the random seed (the model is not very sensitive as indicated in Figure 4.2 of the main paper).

A.2 Matching Multiple People

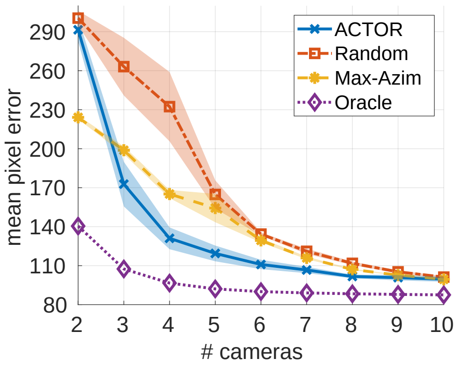
ACTOR reconstructs multiple people in both space and time from 2d pose estimates. In order to track and match these estimates we compute instance sensitive features. These deep features can then be stably matched to each other using the Hungarian algorithm, where the L2-distance is used to compute the matching cost.

We trained an instance classifier structured as a siamese network [5] using a contrastive loss [6] that aims to produce 50-dimensional features for each person that can be used to distinguish individuals. As input the instance classifier takes VGG-19 [7] features from the bounding box of the 2d pose estimate. The instance classifier is trained for 40k iterations on the training set with a minibatch size of 16, where half contains positive pairs and the other half contains negative pairs. The training examples are sampled randomly in both space and time, which yields a robust classifier. Lastly, the instance classifier is fine-tuned for 2k iterations on each scene, creating scene-specific versions of the classifier that are slightly adapted to the environment of those scenes. This tuning is performed outside the range of the active-sequence in which the agent operates.

At the start of an active-sequence the agents is given an appearance model for each target it should reconstruct. These appearance models are averages of K different instance features computed for each target in the scene but from time-freezes that are *not part of the current active-sequence*. We denote the i:th instance feature for the l:th person by $\boldsymbol{u}_i^l$, with $i = 1, \ldots, K$. In practice we use $K = 10$. Then we set as appearance model $\boldsymbol{m}^l$:

$$\boldsymbol{m}^l = \text{median}(\boldsymbol{u}_1^l, \ldots, \boldsymbol{u}_K^l). \tag{4.5}$$

For each camera location we compute the distance between the instance features of each detected person to all appearance models in that scene. This gives us a cost matrix whose



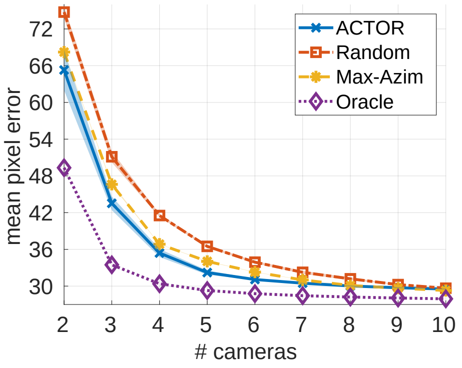


Figure 4.8: Mean 2d reprojection errors per joint relative to OpenPose 2d estimates vs number of cameras on the test sets. Left: Multi-people data. Right: Single-people data. ACTOR reduces the 2d reprojection error faster than the heuristic baselines, particularly for multi-people data. Single-person scenes are easier to reconstruct, especially when using many cameras – also note that all models converge close to the error of the oracle in this case.

elements are $c^{j,l} = \|\boldsymbol{u}^j - \boldsymbol{m}^l\|_2^2$, i.e. the cost to match detection j to person l. Given this matrix we assign detections according to the Hungarian algorithm. Since there might be false detections by the 2d pose estimator and not all people are visible from every camera location, we filter out matches with a cost larger than a threshold C, such that all matches satisfy $c^{j,l} \leq C$ (we set $C = 0.5$).

If a person is never detected in an active-view, and if it does not have a previous temporal backup to use as 3d pose reconstruction (cf. Section 4 and the implementation details in Section 5 of the main paper), we set each joint estimate to the ground truth center hip location. Obviously, this estimate is implausible and highly inaccurate – it is used only to compute average errors (not including such an estimate when computing average errors would be another option, but this would not penalize missed persons).

A.3 Reprojection Errors onto OpenPose 2d Estimates

The 3d ground truth in Panoptic is generated from exhaustive triangulation of 2d pose estimates [8], but those 2d pose estimates are not from OpenPose. Thus it is relevant to also look at reprojection errors onto the OpenPose 2d estimates, since these errors are not affected by any potential incorrect bias in the 3d ground truth. Such reprojection errors are shown in Figure 4.8. We note that ACTOR is more accurate relative to the oracle in this metric. For single-people data the agent converges close to the oracle, while the oracle is still slightly better for multi-people data due to its more difficult nature with occlusions. ACTOR yields lower reprojection errors than the heuristic baselines (exception at 2 cameras for multi-people data, where *Max-Azim* is more accurate). Note that ACTOR was not trained to produce accurate estimates at any fixed number of cameras, but rather to quickly triangulate all joints. Despite this we outperform the baselines in the vast majority of cases.

Table 4.3: The number of images in our dataset categorized by scene type and subset type (training, validation, testing). Note that *Mafia* and *Ultimatum* are complex multi-people scenes and that they account for more than half of the dataset.

	Training	**Validation**	**Testing**	**All**
Mafia	53,100	27,900	33,728	114,728
Ultimatum	27,960	4,340	55,825	88,125
Pose	51,079	29,672	59,288	140,039
All	132,139	61,912	148,841	342,892

A.4 Additional Dataset Insights

Table 4.3 shows the size and split of the Panoptic dataset [8] into training, validation and test sets. The data was created using scripts that downsampled from 30 FPS to 2 FPS to increase movement between frames.

References

[1] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *CVPR*, 2017.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[4] E. Gärtner, "Hyperdock." `https://github.com/ErikGartner/Hyperdock`, 2019.

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," in *NeurIPS*, 1994.

[6] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *CVPR*, 2006.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[8] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *ICCV*, 2015.

Paper IV

Deep Reinforcement Learning for Active Human Pose Estimation

Erik Gärtner[1*] **Aleksis Pirinen**[1*] **Cristian Sminchisescu**[1,2,3]
[1]Centre for Mathematical Sciences, Lund University
[2]Institute of Mathematics of the Romanian Academy
[3]Google Research

Abstract

Most 3d human pose estimation methods assume that input – be it images of a scene collected from one or several viewpoints, or from a video – is given. Consequently, they focus on estimates leveraging prior knowledge and measurement by fusing information spatially and/or temporally, whenever available. In this paper we address the problem of an active observer with freedom to move and explore the scene spatially – in 'time-freeze' mode – and/or temporally, by selecting informative viewpoints that improve its estimation accuracy. Towards this end, we introduce *Pose-DRL*, a fully trainable deep reinforcement learning-based active pose estimation architecture which learns to select appropriate views, in space and time, to feed an underlying monocular pose estimator. We evaluate our model using single- and multi-target estimators with strong results in both settings. Our system further learns automatic stopping conditions in time and transition functions to the next temporal processing step in videos. In extensive experiments with the Panoptic multi-view setup, and for complex scenes containing multiple people, we show that our model learns to select viewpoints that yield significantly more accurate pose estimates compared to strong multi-view baselines.

*Denotes equal contribution, order determined by coin flip.

1 Introduction

Existing human pose estimation models, be them designed for 2d or 3d reconstruction, typically assume that viewpoint selection is outside the control of the estimation agent. This problem is usually solved by a human, either once and for all, or by moving around and tracking the elements of interest in the scene. Consequently, the work is split between *sufficiency* (e.g. instrumenting the space with as many cameras as possible in motion capture setups), *minimalism* (work with as little as possible, ideally a single view, as given), or *pragmatism* (use whatever is available, e.g. a stereo system and lidar in a self-driving car). While each of these scenarios and their underlying methodologies make practical or conceptual sense in their context of applicability, none covers the case of an active observer moving in the scene in order to reduce uncertainty, with emphasis on trading accuracy and computational complexity. There are good reasons for this, as experimenting with an active system faces the difficulty of linking perception and action in the real world, or may have to resort on simulation, which can however lack visual appearance and motion realism, especially for complex articulated and deformable structures such as people.

In this work we consider 3d human pose estimation from the perspective of an active observer, and operate with an idealization that allows us to distill the active vision concepts, develop new methodology, and test it on real image data. We work with a Panoptic massive camera grid [1], where we can both observe the scene in time-freeze, from a dense variety of viewpoints, and process the scene temporally, thus being able to emulate a moving observer. An active setup for 3d human pose estimation addresses the incomplete body pose observability in any monocular image due to depth ambiguities or occlusions (self-induced or produced by other people or objects). It also enables adaptation with respect to any potential limitations of the associated monocular pose estimation system, by sequentially selecting views that when combined yield accurate pose predictions.

In this context we introduce *Pose-DRL*, a deep reinforcement learning (RL) based active pose estimation architecture operating in a dense camera rig, which learns to select appropriate viewpoints to feed an underlying monocular pose predictor. Moreover, our model learns when to stop viewpoint exploration in time-freeze, or continue to the next temporal step when processing video. In evaluations using Panoptic we show that our system learns to select sets of views which yield more accurate pose estimates compared to strong multi-view baselines. The results not only show the advantage of intelligent viewpoint selection, but also that often 'less is more', as fusing too many possibly incorrect viewpoint estimates leads to inferior results.

As our model consists of a deep RL-based active vision module on top of a task module, it can be easily adjusted for other visual routines in the context of a multi-camera setup by simply replacing the task module and retraining the active vision component, or refining them jointly in case of access and compatibility. We show encouraging results using different pose estimators and task settings.

2 Related Work

Extracting 2d and 3d human representations from given images or video is a vast research area, recently fueled by progress in keypoint detection [2, 3], semantic body parts segmentation [4], 3d human body models [5], and 3d motion capture data [6, 7]. Deep learning plays a key role in most human pose and shape estimation pipelines [8, 9, 4, 10, 11, 12, 13, 14], sometimes in connection with non-linear refinement [8, 12]. Systems integrating detailed face, body and hand models have also been proposed [15]. Even so, the monocular 3d case is challenging due to depth ambiguities, which motivated the use of additional ordering constraints during training [16].

In addition to recent literature for static pipelines, the community has recently seen an increased interest in active vision tasks, including RL-based visual navigation [17, 18, 19, 20]. In [17], a real-world dataset of sampled indoor locations along multiple viewing directions is introduced. An RL-agent is trained to navigate to views in which a given instance detector is accurate, similar in spirit to what we do, but in a different context and task. A joint gripping and viewing policy is introduced in [21], also related to us in seeking policies that choose occlusion-free views. The authors of [22] introduce an active view selection system and jointly learn a geometry-aware model for constructing a 3d feature tensor, which is fused together from views predicted by a policy network. In contrast to us, their policy predicts one of eight adjacent discrete camera locations, they do not consider moving objects, their model does not automatically stop view selection, and they do not use real data.

In [23, 24], active viewpoint selection is considered for panoramic completion and panorama projection, respectively. Differently from us, their viewpoint selection policies operate on discretized spheres and do not learn automatic stopping conditions. An approach for active multi-view object recognition is proposed in [25], where pairs of images in a viewpoint trajectory are sequentially fed to a CNN for recognition and for next-best-view prediction. Optimization is done over discretized movements and pre-set trajectory lengths, in contrast to us.

Most related to us is [26], who also consider active view selection in the context of human pose estimation. However, they work with 2d joint detectors and learn to actively triangulate those into 3d pose reconstructions. Thus we face different challenges – while [26] only require each joint to be visible in two views for triangulation, our model has to consider which views yield accurate fused estimates. Furthermore, their model does not learn a stopping action that trades accuracy for speed, and they do not study both the single-target and multi-target cases, as we do in this paper.

Aside from active vision applications in real or simulated environments, reinforcement learning has also been successfully applied to other vision tasks, e.g. object detection [27, 28], object tracking [29, 30] and visual question answering [31].

3 Active Human Pose Estimation

In this section we describe our active human pose estimation framework, arguing it is a good proxy for a set of problems where an agent has to actively explore to understand the scene and integrate task relevant information. For example, a single view may only contain parts of the human body (or be absent of the person altogether) and the agent needs to find a better view to capture the person's pose. Pose estimators are often trained on a limited set of viewing angles and yield lower performance for others. Our setup forces the agent to also take any estimation limitations into account when selecting multiple views. In particular, we show in Section 5.1 that learning to find good views and fusing them is more important than relying on a large number of random ones, or the full available set, as standard – see also Figure 5.1. Concepts in the following sections will, for simplicity, be described assuming the model is estimating the pose of a single *target person* (though scenes may contain multiple people occluding the target). The setting in which *all* people are reconstructed simultaneously is described in Section 4.4.

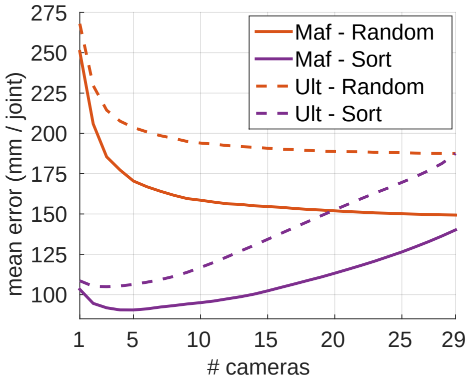


Figure 5.1: Mean per-joint pose reconstruction error for the monocular human pose estimator DMHS vs. number of viewpoints, both when randomly choosing views, and when using a sorting strategy which selects views in ascending order of reconstruction error. Results shown for multi-people data (*Mafia*, *Ultimatum*) on the CMU Panoptic dataset. For a good viewpoint selection policy such as *Sort*, estimation accuracy improves when adding only a few extra cameras, but then begins to deteriorate. This indicates the need to adaptively terminate view selection early enough.

3.1 Active Pose Estimation Setup

We idealize our active pose estimation setup using CMU's Panoptic installation [1] as it captures real video data of scenes with multiple people and cameras densely covering the viewing sphere. This allows us to simulate an active agent observing the scene from multiple views, without the complexity of actually moving a camera. It also enables controllable and reproducible experiments. The videos are captured in a large spherical dome fitted with synchronized HD cameras.[1] Inside the dome several human actors perform a range of movements, with 2d and 3d joint annotations available. The dataset is divided into a number of *scenes*, video recordings from all synchronized cameras capturing different actors and

[1]There are about 30 cameras per scene. The HD cameras provide better image quality than VGA and are sufficiently dense, yet spread apart far enough to make each viewpoint unique.

types of movements, ranging from simple pose demonstrations to intricate social games.

Terminology. We call a *time-freeze* $\{v_1^t, \ldots, v_N^t\}$ the collection of views from all N time-synchronized cameras at time t, with v_i^t the image (referred to as *view* or *viewpoint*) taken by camera i at time t. A subset of a time-freeze is an *active-view* $\mathcal{V}^t = \{v_1^t, \ldots, v_k^t\}$, containing k selected views from the time-freeze. A temporally contiguous sequence of active-views is referred to as an *active-sequence*, $\mathcal{S}^{1:T} = \{\mathcal{V}^1, \mathcal{V}^2, \ldots, \mathcal{V}^T\}$. We will often omit the time superfix t unless the context is unclear; most concepts will be explained at the level of time-freezes. The image corresponding to a view v_i can be fed to a 3d pose predictor to produce a pose estimate $\boldsymbol{x}_i \in \mathbb{R}^{45}$ ($15\times$ 3d joints).

Task definition. We define the task of *active pose estimation* at each time step as selecting views from a time-freeze to generate an active-view. The objective is to produce an accurate fused estimate $\boldsymbol{x}_\star$ from pose predictions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ associated with the active-view (k may vary between active-views). The deep pose estimation network is computationally demanding and therefore working with non-maximal sets of views decreases processing time. Moreover, it improves estimates by ignoring obstructed views, or those a given pose predictor cannot accurately handle. The goal of the full active pose estimation task is to produce accurate fused pose estimates over the full sequence, i.e. to produce an active-sequence with accurate corresponding fused pose estimates.

3.2 Detection and Matching of Multiple People

To solve active human pose estimation the model must address the problems of detecting, tracking, and distinguishing people in a scene. It must also be robust to variations in appearance since people are observed over time and from different viewpoints. We use Faster R-CNN [32] for detecting people. At the start of an active-sequence the agent is given appearance models, consisting of instance-sensitive features for each person. For each visited view, the agent computes instance features for all detected persons, comparing them with the given appearance models to identify the different people.

Obtaining appearance models. A generic instance classifier, implemented as a VGG-19 based [33] siamese network, is trained for 40k iterations on the training set with a contrastive loss to distinguish between different persons. Each minibatch consists of 16 randomly sampled pairs of ground truth crops of people in the training set. We ensure that the training is balanced by sampling pairs of person crops such that the probability of the two crops containing the same person is the same as that of containing two different persons. The people crops are sampled uniformly across scenes, spatially and temporally, yielding a robust instance classifier.

Once the instance classifier has been trained, we fine-tune it for 2k iterations for each scene and then use it to construct appearance models at the beginning of an active-sequence.

For each person, we sample L instance features from time-freezes from the same scene, but outside of the time span of the current active-sequence to limit overfitting. Denote by $\boldsymbol{u}_i^l$ the i:th instance feature for the l:th person, with $i = 1, \ldots, L$. Then we compute the appearance model as

$$\boldsymbol{m}^l = \text{median}(\boldsymbol{u}_1^l, \ldots, \boldsymbol{u}_L^l). \tag{5.1}$$

We set $L = 10$ to obtain a diverse set of instance features for each person, yielding a robust appearance model.

Stable matching of detections. In each visited viewpoint during an active-sequence, the agent computes instance features for all detected persons, comparing them with the given appearance models to identify the different people. To ensure a stable matching, we use the Hungarian algorithm. Specifically, the cost $c^{j,l}$ of matching the j:th detection with instance feature $\boldsymbol{u}^j$ in the current viewpoint to the appearance model $\boldsymbol{m}^l$ of the l:th person is $c^{j,l} = \|\boldsymbol{u}^j - \boldsymbol{m}^l\|_2^2$. Since the target person may not be visible in all viewpoints throughout the active-sequence, we specify a *cost threshold*, $C = 0.5$, such that if the assignment cost $c^{j,l}$ of the target is above it (i.e. $c^{j,l} > C$), we consider the person to not be visible in the view. In that case the associated pose is not fused into the final estimate.

4 Deep Reinforcement Learning Model

We now introduce our Pose-DRL agent for solving the active human pose estimation task and first explain the agent's state representation and actions, then define the reward signal for training an agent which selects views that yield an accurate fused pose estimate while keeping down the processing time.

4.1 Overview of the Pose-DRL Agent

The agent is initiated at a randomly selected view v_1^1 in the first active-view $\mathcal{V}^1$ of an active-sequence $\mathcal{S}^{1:T}$. Within the current active-view $\mathcal{V}^t$, the agent issues *viewpoint selection* actions to progressively select a sequence of views $v_2^t, \ldots, v_k^t$, the number of which may vary between active-views. At each view v_i^t the underlying pose estimator predicts the pose $\boldsymbol{x}_i^t$. As seen in Figure 5.2 the cameras are approximately located on a partial sphere, so a viewpoint can be specified by the azimuth and elevation angles (referred to as spherical angles). Thus for viewpoint selection the Pose-DRL agent predicts spherical angles relative to its current location and selects the camera closest to those angles.

Once the agent is done exploring viewpoints associated to a particular time-freeze it issues the *continue* action and switches to the next active-view $\mathcal{V}^{t+1}$, at which time the collection of individual pose estimates $\boldsymbol{x}_i^t$ from the different viewpoints are fused together with the estimate $\boldsymbol{x}_\star^{t-1}$ from the previous active-view $\mathcal{V}^{t-1}$:

$$\boldsymbol{x}_\star^t = f(\boldsymbol{x}_\star^{t-1}, \boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_k^t). \tag{5.2}$$

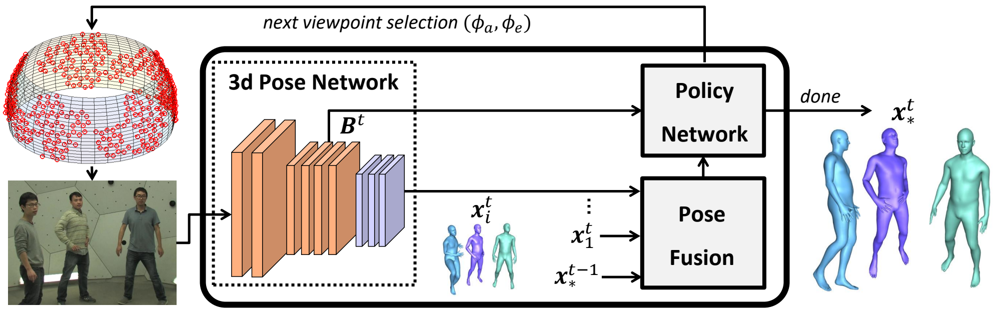


Figure 5.2: Overview of our Pose-DRL agent for active human pose estimation. The agent initially observes the scene from a randomly given camera on the rig. In each visited viewpoint, the associated image is processed by a 3d pose estimation network, producing the base state $\boldsymbol{B}^t$ of the agent and pose estimate(s) $\boldsymbol{x}_i^t$. The pose estimate is fused together with estimates from previous viewpoints $\boldsymbol{x}_1^t, \ldots, \boldsymbol{x}_{i-1}^t$ and the previous temporal step $\boldsymbol{x}_\star^{t-1}$. Both the current and fused estimate are fed as additional features to the agent. At each step the policy network outputs the next viewpoint, until it decides it is done and continues to next active-view at time $t+1$. The viewpoint selection action predicts spherical angles relative to the agent's current location on the camera rig, and the closest camera associated with the predicted angles is visited next. When the agent is done it outputs $\boldsymbol{x}_\star^t$, the per-joint fusion of the individual pose estimates seen during the current active-view and the fused pose estimate from the previous active-view, cf. (5.2). Pose-DRL can be used either to reconstruct a target person, or to reconstruct all people in a scene. The underlying pose estimator is exchangeable – we show strong results using two different ones in Section 5.1.

Including the previous time step estimate $\boldsymbol{x}_\star^{t-1}$ in the pose fusion as in (5.2) often improves results (see Section 5.2). After returning the fused pose estimate $\boldsymbol{x}_\star^t$ for the current active-view, the agent continues to the next active-view $\mathcal{V}^{t+1}$. The initial view v_1^{t+1} for $\mathcal{V}^{t+1}$ is set to the final view v_k^t of $\mathcal{V}^t$, i.e. $v_1^{t+1} = v_k^t$. The process repeats until the end of the active-sequence. Figure 5.2 and 5.3 show model overviews for active-views and active-sequences, respectively.

4.2 State-Action Representation

To simplify notation we here describe how the agent operates in a given time-freeze and in this context we will use t to index actions within the active-view, as opposed to temporal structures. The state at step t is the tuple $s^t = (\boldsymbol{B}^t, \boldsymbol{X}^t, \boldsymbol{C}^t, \boldsymbol{u}^t)$. Here $\boldsymbol{B}^t \in \mathbb{R}^{H\times W\times C}$ is a deep feature map associated with the underlying 3d pose estimation architecture, and $\boldsymbol{X}^t = \{\boldsymbol{x}_t, \tilde{\boldsymbol{x}}, \boldsymbol{x}_\star^{\text{hist}}\}$, where $\boldsymbol{x}_t$ is the current individual pose estimate, $\tilde{\boldsymbol{x}} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$ is the current partially fused pose estimate, and $\boldsymbol{x}_\star^{\text{hist}}$ is a history of fused predictions from 4 previous active-views. The two-channel matrix $\boldsymbol{C}^t \in \mathbb{N}^{w\times h\times 2}$ consists of an *angle canvas*, a discretized encoding[2] of the previously visited regions on the camera rig, as well as a similar

[2]The camera sphere is discretized into w bins in the azimuth direction and h bins in the elevation direction, appropriately wrapped to account for periodicity. We set $w = 9$ and $h = 5$.

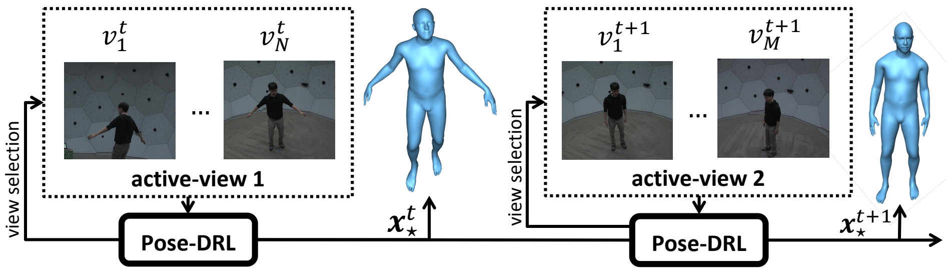


Figure 5.3: Illustration of how Pose-DRL operates on an active-sequence, here for a single-person scenario. Fused pose estimates are fed to subsequent active-views within the active-sequence, both as additional state representation for action selection, and for fusing poses temporally.

encoding of the camera distribution over the rig. Finally, $\boldsymbol{u}^t \in \mathbb{R}^2$ is an auxiliary vector holding the number of actions taken and the number of people detected.

For action selection we use a deep stochastic policy $\pi_{\boldsymbol{\theta}}(a^t|s^t)$ parametrized by $\boldsymbol{\theta}$ which predicts the action $a^t = \{\phi_a^t, \phi_e^t, c^t\}$. Here (ϕ_a^t, ϕ_e^t) is the azimuth-elevation angle pair, jointly referred to as *viewpoint selection*, and c^t is a Bernoulli variable indicating whether to continue to the next active-view (occurs if $c^t = 1$), referred to as the *continue* action. To produce action probabilities, the base feature map $\boldsymbol{B}^t$ is fed through two convolutional blocks which are shared between the viewpoint selection and continue actions. The output of the second convolutional block is then concatenated with $\boldsymbol{X}^t$, $\boldsymbol{C}^t$ and $\boldsymbol{u}^t$ and fed to viewpoint selection- and continue-branches with individual parameters. Both action branches consist of three fully connected layers with $\tanh$ activations. The probability of issuing the *continue* action is computed using a sigmoid layer:

$$\pi_{\boldsymbol{\theta}}(c^t = 1|s^t) = \sigma\left[\boldsymbol{w}_c^\top \boldsymbol{z}_c^t + b_c\right], \tag{5.3}$$

where $\boldsymbol{w}_c$ and b_c are trainable weights and bias, and $\boldsymbol{z}_c^t$ is the output from the penultimate fully connected layer of the *continue* action branch.

Due to the periodic nature of the viewpoint prediction task we rely on von Mises distributions for sampling the spherical angles. We use individual distributions for the azimuth and elevation angles. The probability density function for the azimuth is given by:

$$\pi_{\boldsymbol{\theta}}\left(\phi_a^t|s^t\right) = \frac{1}{2\pi I_0(m_a)} \exp\{m_a \cos(\phi_a^t - \tilde{\phi}_a(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a))\}, \tag{5.4}$$

where I_0 is the zeroth-order Bessel function, normalizing (5.4) to a proper probability distribution over the unit circle $[-\pi, \pi]$. Here $\tilde{\phi}^a$ is the mean of the distribution (parametrized by the neural network), m_a is the precision parameter,[3] $\boldsymbol{w}_a$ and b_a are trainable weights

[3]We treat the precision parameters as constants but increase them over training to focus the policy on high-reward viewpoints.

and bias, respectively, and $\boldsymbol{z}_a^t$ comes from the penultimate fully connected layer of the viewpoint selection action branch. The support for the azimuth angle should be on a full circle $[-\pi, \pi]$, and hence we set

$$\tilde{\phi}_a(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a) = \pi \tanh(\boldsymbol{w}_a^\top \boldsymbol{z}_a^t + b_a). \tag{5.5}$$

The probability density function for the elevation angle has the same form (5.4) as that for the azimuth. However, as seen in Figure 5.2, the range of elevation angles is more limited than for the azimuth angles. We denote this range $[-\kappa, \kappa]$ and the mean elevation angle thus becomes

$$\tilde{\phi}_e(\boldsymbol{w}_e^\top \boldsymbol{z}_e^t + b_e) = \kappa \tanh(\boldsymbol{w}_e^\top \boldsymbol{z}_e^t + b_e), \tag{5.6}$$

with notation analogous to that of the azimuth angle, cf. (5.5). In practice, when sampling elevation angles from the von Mises, we reject samples outside the range $[-\kappa, \kappa]$.

4.3 Reward Signal for Policy Gradient Objective

The agent should strike a balance between choosing sufficiently many cameras so that the resulting 3d pose estimate becomes as accurate as possible, while ensuring that not too many cameras are visited, to save processing time. As described earlier, the two types of actions are *viewpoint selection* and *continue*. We will next cover the reward functions for them.

Viewpoint selection reward. At the end of an active-view we give a reward which is inversely proportional to the ratio between the final and initial reconstruction errors within the active-view. We also give a penalty $\epsilon = 2.5$ each time the agent moves to an already visited viewpoint. Thus the viewpoint selection reward is:

$$r_v^t = \begin{cases} 0, & \text{if } c^t = 0 \text{ and view not visited} \\ -\epsilon, & \text{if } c^t = 0 \text{ and view visited before} \\ 1 - \frac{\varepsilon^k}{\varepsilon^1}, & \text{if } c^t = 1, \end{cases} \tag{5.7}$$

where k is the number of views visited prior to the agent issuing the *continue* action ($c^t = 1$), ε^1 is the reconstruction error associated with the initial viewpoint, and ε^k denotes the final reconstruction error, i.e. $\varepsilon^k = \|\boldsymbol{x}_\star - \boldsymbol{x}_{\text{gt}}\|_2^2$. Here $\boldsymbol{x}_\star$ is the final fused pose estimate for the active-view, cf. (5.2), and $\boldsymbol{x}_{\text{gt}}$ is the ground truth 3d pose for the time-freeze.

Continue action reward. The *continue* action has two purposes: i) ensure that not too many viewpoints are visited, to reduce computation time, and ii) stop before suboptimal viewpoints are explored, which could happen if the agent is forced to visit a preset number of viewpoints. Therefore, the *continue* action reward is:

$$r_c^t = \begin{cases} 1 - \frac{\min\limits_{j \in \{t+1,\ldots,k\}} \varepsilon^j}{\varepsilon^t} - \tau, & \text{if } c^t = 0 \\ 1 - \frac{\varepsilon^k}{\varepsilon^1}, & \text{if } c^t = 1. \end{cases} \tag{5.8}$$

At each step that the agent decides *not* to continue to the next active-view ($c^t = 0$), the agent is rewarded relative to the ratio between the error at the best future stopping point within the active-view (with lowest reconstruction error) and the error at the current step. If in the future the agent selects viewpoints that yield lower reconstruction error the agent is rewarded, and vice versa if the best future error is higher. In addition, the agent gets a penalty τ at each step, which acts as an improvement threshold, causing the reward to become negative unless the ratio is above the specified threshold τ. This encourages the agent not to visit many viewpoints in the current active-view unless the improvement is above the given threshold. On the validation set we found $\tau = 0.07$ to provide a good balance.

Policy gradient objective. We train the Pose-DRL network in a policy gradient framework, maximizing expected cumulative reward on the training set with objective

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=1}^{|\boldsymbol{s}|} r^t\right], \tag{5.9}$$

where $\boldsymbol{s}$ denotes state-action trajectories, and the reward signal $r^t = r_v^t + r_c^t$, cf. (5.7) - (5.8). We approximate the gradient of the objective (5.9) using REINFORCE [34].

4.4 Active Pose Estimation of Multiple People

So far we have explained the Pose-DRL system that estimates the pose of a target person, assuming it is equipped with a detection-based single person estimator. This system can in principle estimate multiple people by generating active-sequences for each person individually. However, to find a *single* active-sequence that reconstructs *all* persons, one can equip Pose-DRL with an image-level multi-people estimator instead. The state representation is then modified to use the image level feature blob from the multi-people estimator ($\boldsymbol{B}_t$ in Figure 5.2). The reward signal used when learning to reconstruct all people is identical to (5.7) - (5.8), except that the rewards are averaged over the individual pose estimates. Thus Pose-DRL is very adaptable in that the underlying pose estimator can easily be changed.

5 Experiments

Dataset. We use diverse scenes for demonstrating our active pose estimation system, considering complex scenes with multiple people (*Mafia*, *Ultimatum*) as well as single person ones (*Pose*). The motions range from basic poses to various social games. Panoptic provides data as 30 FPS-videos which we sample to 2 FPS, which makes the data more manageable in size. It also increases the change in pose between consecutive frames.

The data we use consists of the same 20 scenes as in [26]. The scenes are randomly split into training, validation and test sets with 10, 4 and 6 scenes, respectively. Since we split the data over the scenes the agent needs to learn a general look-around-policy which adapts to various circumstances (scenarios and people differ between scenes). All model selection is performed exclusively on the training and validation sets; final evaluations are performed on the test set. The data consists of 343k images, where 140k are single-person and 203k are multi-person scenes.

Implementation details. We attach Pose-DRL on top of the DMHS monocular pose estimation system [4]. In the multi-people setting (cf. Section 4.4) we instead use MubyNet [35]. Both estimators were trained on Human3.6M [6]. To avoid overfitting we do not to fine-tune these on Panoptic, and instead emphasize how Pose-DRL can select good views with respect to the underlying estimation system (but joint training is possible). We use an *identical set of hyperparameters* when using DMHS and MubyNet, except the improvement threshold τ (-0.07 for DMHS and -0.04 for MubyNet). Thus Pose-DRL is robust with respect to the pose estimator used. We use median averaging for fusing poses, cf. (5.2).

Training. We use 5 active-sequences, each of length 10, to approximate the policy gradient, and update the policy parameters using Adam [36]. As standard, to reduce variance we normalize cumulative rewards for each episode to zero mean and unit variance over the batch. The maximum trajectory length is set to 8 views including the initial one (10 in the multi-target mode, as it may require more views to reconstruct all people). The *viewpoint selection* and *continue* actions are trained jointly for 80k episodes. The learning rate is initially set to $5 \cdot 10^{-7}$ and is halved at 720k and 1440k training steps. We linearly increase the precision parameters m_a and m_e of the von Mises distributions from $(1, 10)$ to $(25, 50)$ in training, making the viewpoint selection increasingly focused on high-rewarding regions as training proceeds.

Baselines. To evaluate our active human pose estimation system we compare it to several baselines, similar to [26]. For fair comparisons, the baselines use the same pose estimator, detector and matching approach. All methods also obtain the same initial random view as the agent at the start of the active-sequence. We design the following baselines: i) *Random:* Selects k different random views; ii) *Max-Azim:* Selects k different views equidistantly with respect to the azimuth angle. At each azimuth angle it selects a random elevation angle; iii) *Oracle:* Selects as next viewpoint the one that minimizes the fused 3d pose reconstruction when combined with pose estimates from all viewpoints observed so far (averaged over all people in the multi-target setting). This baseline cheats by extensively using ground truth information, and thus it shown as a lower bound with respect to reconstruction error. In addition to cheating during viewpoint selection, the oracle is also impractically slow since it requires computing pose estimates for *all* available viewpoints and exhaustively computing errors for all cameras in each step.

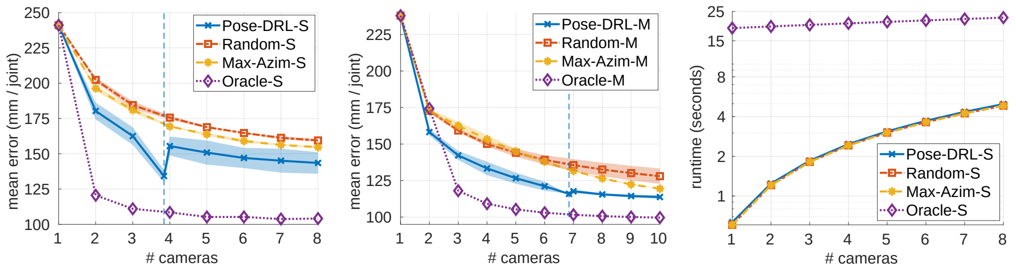


Figure 5.4: How the number of views affects pose estimation error and runtimes of Pose-DRL and baselines on multi-people data (union of *Mafia* and *Ultimatum* test sets). We show mean and 95% confidence intervals over 5 seeds. Left: Reconstructing a single target person. Estimation error reduces with added viewpoints, and the agent consistently outperforms the non-oracle baselines. The automatic *continue* action (dashed line at 3.8 views on average) yields significantly lower reconstruction errors than any fixed viewpoint schemes. Hence the auto-model clearly provides the best speed-accuracy trade-off. Middle: Simultaneously reconstructing all persons. The agent outperforms the heuristic baselines in this setting too. Adaptively determining when to continue to the next active-view (6.8 views on average) yields better results than fusing from 7 cameras all the time. The gain is not as pronounced as in the single-target case, since inspecting more viewpoints typically leads to increased estimation accuracy for some of the persons. Right: Runtime of the Pose-DRL agent and baselines vs. number of views (log scale). The oracle always needs to evaluate the deep pose estimation system and detector for all cameras due to its need to sort from best to worst, independently of the number of views, which explains its high runtime. Our agent is as fast as the heuristic baselines.

5.1 Quantitative Results

We report results both for the Pose-DRL agent that tracks and reconstructs a single target person (possibly in crowded scenes) and for the Pose-DRL model which actively estimates poses for all persons in the scene, cf. Section 4.4. Pose-DRL is trained over 5 different random initializations of the policy network, and we report average results. In each case, training the model 80k steps gave best results on the validation set, so we use that. Also, for the heuristic baselines we report average results over 5 seeds (the oracle is deterministic). When computing errors, we first hip-align pose estimates with their respective ground truths. Our agent is compared to the baselines on the Panoptic test set on active-sequences consisting of 10 active-views. Table 5.1 presents reconstruction errors. Figure 5.4 shows how the the number of selected views affects accuracy and runtimes. For visualizations of Pose-DRL, see Figure 5.5 - 5.7 (in these visualizations we use SMPL [5] for the 3d shape models).

Table 5.1: Reconstruction error (mm/joint) for Pose-DRL and baselines on active-sequences on the selected Panoptic test splits. Results are shown both for the setting where the agent decides the number of views (auto), and when using a fixed number of views. In the latter case, the number of views is set to the closest integer corresponding to the average in auto-mode, rounded up. The baselines are also evaluated at this preset number of views. The average number of views are shown in parentheses. Pose-DRL models which automatically select the number of views outperform the heuristic baselines and fixed Pose-DRL models on all data splits, despite fusing estimates from fewer views on average. Left: Single-target mode (S), using DMHS as pose estimator. The agent significantly outperforms the baselines (e.g. 35 mm/joint improvement over *Max-Azim* on multi-people data *Maf* + *Ult*). Right: Multi-target mode (M), using MubyNet as pose estimator. MubyNet is a more recent and accurate estimator, so the average errors are typically lower than the DMHS-counterparts. Automatic termination is useful in the multi-target setting as well, although it does not provide as drastic gains as in the single-target setup.

Model	# Views	Maf	Ult	Pose	Maf + Ult	All
Pose-DRL-S	**auto**	130.3 (4.6)	135.4 (3.4)	135.3 (3.7)	134.2 (3.8)	135.0 (3.7)
	fixed	144.7 (5.0)	157.5 (4.0)	135.1 (4.0)	155.5 (4.0)	140.4 (4.0)
Rand-S	**fixed**	160.2 (5.0)	178.3 (4.0)	145.7 (4.0)	175.6 (4.0)	157.1 (4.0)
Max-Azim-S	**fixed**	156.3 (5.0)	171.4 (4.0)	139.9 (4.0)	169.4 (4.0)	150.3 (4.0)
Oracle-S	**fixed**	103.4 (5.0)	108.9 (4.0)	106.5 (4.0)	108.5 (4.0)	105.4 (4.0)

Model	# Views	Maf	Ult	Pose	Maf + Ult	All
Pose-DRL-M	**auto**	114.8 (7.5)	116.4 (6.6)	104.6 (2.1)	115.9 (6.8)	110.7 (4.5)
	fixed	114.8 (8.0)	118.0 (7.0)	106.7 (3.0)	117.6 (7.0)	112.8 (5.0)
Rand-M	**fixed**	128.8 (8.0)	134.9 (7.0)	115.9 (3.0)	131.4 (7.0)	126.0 (5.0)
Max-Azim-M	**fixed**	123.5 (8.0)	131.2 (7.0)	116.3 (3.0)	131.6 (7.0)	126.4 (5.0)
Oracle-M	**fixed**	98.6 (8.0)	102.4 (7.0)	90.2 (3.0)	101.6 (7.0)	92.6 (5.0)

Single-target estimation. It is clear from Table 5.1 (left) and Figure 5.4 (left) that Pose-DRL outperforms the heuristic baselines, which is particularly pronounced for multi-people data. In such scenes the view selection process is more delicate, as it requires avoiding cameras where the target is occluded. We note that the automatically stopping agent yields by far the most accurate estimates, which shows that it is able to continue to the next active-view when it is likely that the current one does not provide any more good views. Thus it is often better to fuse a few accurate estimates than including a larger set of poorer ones.

Multi-target estimation. From Table 5.1 (right) and Figure 5.4 (middle) we see that the agent outperforms the heuristic baselines as in the case with a single target. Automatic view selection termination does not yield as big improvements in accuracy as in the single-target case. In the single-target setting the agent stops early to avoid occluded and bad views, but when reconstructing all people there is more reason to keep selecting additional views to find some views which provide reasonable estimates for each person. This also explains the decreased gaps between the various methods – there may be many sets of cameras which together provide a fairly similar result when averaged over all people in the scene (a future improvement of Pose-DRL could include selectively fusing a subset of estimates in each view). Running in auto-mode still yields more accurate estimates than fixed schemes which use a larger number of views.

Runtimes. The runtimes[4] of Pose-DRL and baselines are shown in Figure 5.4 (right). DMHS and Faster R-CNN require 0.50 and 0.11 seconds per viewpoint, respectively, which constitutes the bulk of the processing time. The policy network has a negligible overhead of about 0.01 seconds per action.

5.2 Ablation Studies

In this section we compare the full agent to versions lacking key parts of the model: i) providing only the base feature map $\boldsymbol{B}^t$, and ii) not propagating the fused reconstruction $\boldsymbol{x}_\star^t$ to the next active-view (*reset*), cf. (5.2). The results are given in Table 5.2 and show that the full model outperforms the stripped-down versions for multi-people data (*Mafia*, *Ultimatum*), while simpler single-people data (*Pose*) is not sensitive to removing some parts of the model. There is significantly more room for intelligent decision making for complex multi-people data where the model has to avoid occlusions, and thus it requires a stronger state description and fusion approach. In contrast, selecting viewpoints in single-people scenes is less fragile to the particular camera choices as there is no risk of choosing views where the target is occluded.

[4]Shown for DMHS-based systems. Using MubyNet (which requires 1.01 seconds per image) gives runtime curves which look qualitatively similar.

Table 5.2: Ablations on the test sets, showing the effect of removing certain components of the DMHS-based Pose-DRL system. Results (errors, mm/joint) are for models that select a fixed number of views (shown in parentheses), where the number of views are the same as in Table 5.1. Providing more information than the base feature map $\boldsymbol{B}^t$ is crucial for crowded scenes with multiple people (*Mafia*, *Ultimatum*), as is including previous pose estimates in the current pose fusion.

Model	Settings	Mafia	Ultimatum	Pose
Pose-DRL	**full model**	144.7 (5)	157.5 (4)	135.1 (4)
	$\boldsymbol{B}^t$ **only**	153.5 (5)	166.9 (4)	134.4 (4)
	reset	152.5 (5)	160.8 (4)	133.4 (4)

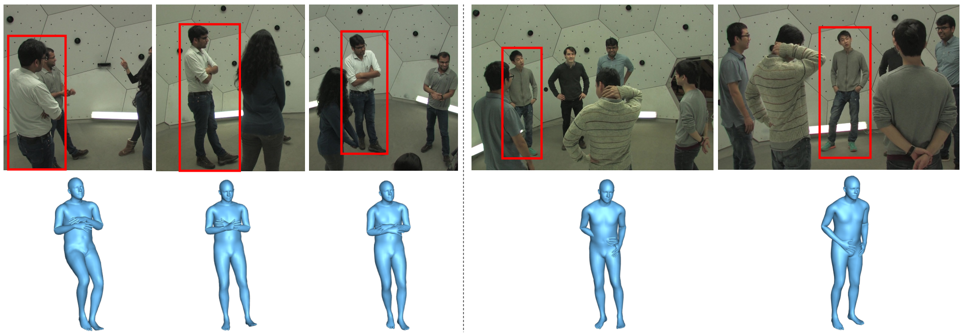


Figure 5.5: Visualizations of Pose-DRL reconstructing a given target person (red bounding box) in scenes with multiple people. Left: A *Mafia* test scene. The target is viewed from behind and is partially visible in the first view, producing the poor first estimate. As the agent moves to the next view, the person becomes more clearly visible, significantly improving the estimate. The last view from the front further increases accuracy. The agent decides to terminate after three views with error decreasing from 200.1 to 120.9 mm/joint. Right: An *Ultimatum* test scene where the agent only requires two viewpoints prior to automatically continuing to the next active-view. The target person is only partially visible in the initial viewpoint, and the right arm that is not visible results in a non-plausible configuration in the associated estimate. As the agent moves to the next viewpoint the person becomes fully visible, and the final fused estimate is both physically plausible and accurate. The reconstruction error reduces from 160 to 104 mm/joint.

6 Conclusions

In this paper we have presented *Pose-DRL*, a fully trainable deep reinforcement-learning based active vision model for human pose estimation. The agent has the freedom to move and explore the scene spatially and temporally, by selecting informative views that improve its accuracy. The model learns automatic stopping conditions for each moment in time, and transition functions to the next temporal processing step in video. We showed in extensive experiments – designed around the dense Panoptic multi-camera setup, and for complex scenes with multiple people – that Pose-DRL produces accurate estimates, and that it is robust with respect to the underlying pose estimator used. Moreover, the results show that our model learns to select an adaptively determined number of informative viewpoints which result in considerably more accurate pose estimates compared to strong multi-view baselines.

Practical developments of our methodology would include e.g. real-time intelligent processing of multi-camera video feeds or controlling a drone observer. In the latter case the model would further benefit from being extended to account for physical constraints, e.g. a single camera and limited speed. Our paper is a key step since it presents fundamental methodology required for future applied research.

Acknowledgments: This work was supported by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535 and PCCF-2016-0180, the EU Horizon 2020 Grant DE-ENIGMA, Swedish Foundation for Strategic Research (SSF) Smart Systems Program, as well as the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Finally, we would like to thank Alin Popa, Andrei Zanfir, Mihai Zanfir and Elisabeta Oneata for helpful discussions and support.

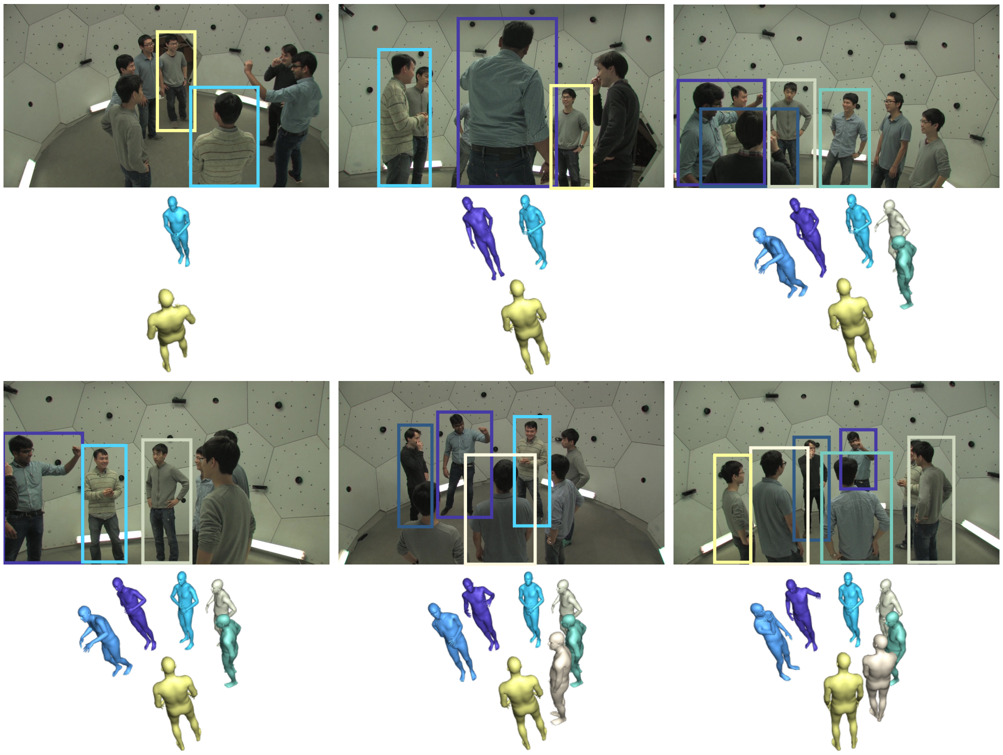


Figure 5.6: Visualization of how Pose-DRL performs multi-target pose estimation in an *Ultimatum* test scene. In this example the agent sees six viewpoints prior to automatically continuing to the next active-view. The mean error decreases from 358.9 to 114.6 mm/joint. Only two people are detected in the initial viewpoint, but the number of people detected increases as the agent inspects more views. Also, the estimates of already detected people improve as they get fused from multiple viewpoints.

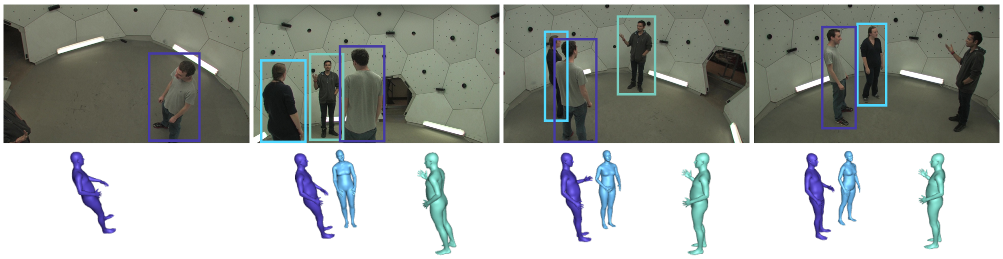


Figure 5.7: Visualization of how Pose-DRL performs multi-target pose estimation in an *Ultimatum* validation scene. The agent chooses four viewpoints prior to automatically continuing to the next active-view. The mean error decreases from 334.8 to 100.9 mm/joint. Only one of the persons is visible in the initial viewpoint, and from a poor angle. This produces the first, incorrectly tilted pose estimate, but the estimate improves as the agent inspects more viewpoints. The two remaining people are successfully reconstructed in subsequent viewpoints.

References

[1] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *ICCV*, 2015.

[2] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[3] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy, "Towards accurate multi-person pose estimation in the wild," in *CVPR*, 2017.

[4] A.-I. Popa, M. Zanfir, and C. Sminchisescu, "Deep multitask architecture for integrated 2d and 3d human sensing," in *CVPR*, 2017.

[5] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *SIGGRAPH*, vol. 34, no. 6, 2015.

[6] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *PAMI*, vol. 36, no. 7, 2014.

[7] T. von Marcard, R. Henschel, M. Black, B. Rosenhahn, and G. Pons-Moll, "Recovering accurate 3d human pose in the wild using imus and a moving camera," in *ECCV*, 2018.

[8] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it SMPL: Automatic estimation of 3d human pose and shape from a single image," in *ECCV*, 2016.

[9] H. Rhodin, N. Robertini, D. Casas, C. Richardt, H.-P. Seidel, and C. Theobalt, "General automatic human shape and motion capture using volumetric contour cues," in *ECCV*, 2016.

[10] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, "Coarse-to-fine volumetric prediction for single-image 3d human pose," in *CVPR*, 2017.

[11] G. Rogez, P. Weinzaepfel, and C. Schmid, "Lcr-net: Localization-classification-regression for human pose," in *CVPR*, 2017.

[12] A. Zanfir, E. Marinoiu, and C. Sminchisescu, "Monocular 3d pose and shape estimation of multiple people in natural scenes–the importance of multiple scene constraints," in *CVPR*, 2018.

[13] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, “Vnect: Real-time 3d human pose estimation with a single rgb camera,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, 2017.

[14] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, “End-to-end recovery of human shape and pose,” in *CVPR*, 2018.

[15] H. Joo, T. Simon, and Y. Sheikh, “Total capture: A 3d deformation model for tracking faces, hands, and bodies,” in *CVPR*, 2018.

[16] G. Pavlakos, X. Zhou, and K. Daniilidis, “Ordinal depth supervision for 3D human pose estimation,” in *CVPR*, 2018.

[17] P. Ammirato, P. Poirson, E. Park, J. Košecká, and A. C. Berg, “A dataset for developing and benchmarking active vision,” in *ICRA*, 2017.

[18] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, “Embodied question answering,” in *CVPR*, 2018.

[19] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, “Gibson env: Real-world perception for embodied agents,” in *CVPR*, 2018.

[20] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *ICRA*, 2017.

[21] R. Cheng, A. Agarwal, and K. Fragkiadaki, “Reinforcement learning of active vision for manipulating objects under occlusions,” in *CoRL*, 2018.

[22] R. Cheng, Z. Wang, and K. Fragkiadaki, “Geometry-aware recurrent neural networks for active visual recognition,” in *NeurIPS*, 2018.

[23] D. Jayaraman and K. Grauman, “Learning to look around: Intelligently exploring unseen environments for unknown tasks,” in *CVPR*, 2018.

[24] B. Xiong and K. Grauman, “Snap angle prediction for 360 panoramas,” in *ECCV*, 2018.

[25] E. Johns, S. Leutenegger, and A. J. Davison, “Pairwise decomposition of image sequences for active multi-view recognition,” in *CVPR*, 2016.

[26] A. Pirinen, E. Gärtner, and C. Sminchisescu, “Domes to drones: Self-supervised active triangulation for 3d human pose reconstruction,” in *NeurIPS*, 2019.

[27] J. Caicedo and S. Lazebnik, “Active object localization with deep reinforcement learning,” in *ICCV*, 2015.

[28] A. Pirinen and C. Sminchisescu, "Deep reinforcement learning of region proposal networks for object detection," *CVPR*, 2018.

[29] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang, "Deep reinforcement learning for visual object tracking in videos," *arXiv preprint arXiv:1701.08936*, 2017.

[30] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Y. Choi, "Action-driven visual object tracking with deep reinforcement learning," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, 2018.

[31] A. Das, S. Kottur, J. M. Moura, S. Lee, and D. Batra, "Learning cooperative visual dialog agents with deep reinforcement learning," in *CVPR*, 2017.

[32] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[34] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992.

[35] A. Zanfir, E. Marinoiu, M. Zanfir, A.-I. Popa, and C. Sminchisescu, "Deep network for the integrated 3d sensing of multiple people in natural images," in *NeurIPS*, 2018.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

A Supplementary Material

In this supplemental we provide additional insights to our Pose-DRL model. Details of the network architecture are provided in Section $A.1$. Further model insights are provided in Section $A.2$. A description of how we handle missed detections or failed matchings are given in Section $A.3$. Additional visualizations are shown in Section $A.4$. Finally, in Section $A.5$ we show proof-of-concept results when using Pose-DRL to control a smartphone camera in the context of active human pose estimation.

A.1 Model Architecture

See Figure 5.8 for a description of the Pose-DRL architecture. The underlying pose estimation networks, DMHS [1] and MubyNet [2], as well as our agent were implemented in Caffe [3] and MATLAB. For the Faster R-CNN detector [4] we used a publicly available Tensorflow [5] implementation,[5] with ResNet-101 [6] as base feature extractor.

A.2 Additional Insights and Details

Table 5.3: Pose-DRL agent's selection statistics of good / bad viewpoints on the test set splits. The agent consistently chooses a high percentage of good cameras while avoiding bad cameras. Note that randomly choosing cameras would result in always having 10% chosen among the 10% best cameras, and similar for the 10% worst cameras.

	10% best	**10% worst**	**Rest**
Mafia	52 %	2%	46%
Ultimatum	67%	1%	32%
Pose	24%	2%	74%
All	43%	2%	55%

More about runtimes. All experiments reported in this supplementary material and in the main paper were performed using an Ubuntu workstation using a single Titan V100. Training the Pose-DRL policy from scratch took about 70 hours after having pre-computed all DMHS / MubyNet features, Faster R-CNN bounding boxes and instance features. When presenting the runtimes (see Figure 5.4 in the main paper) we include the time needed to compute these detections and features.

Quality of selected viewpoints. To obtain further insights to which types of cameras the agent selects on average, we tracked how often it selects good vs bad viewpoints for the DMHS-based model. Specifically, for each selected camera in the various test set splits, we

[5] `https://github.com/smallcorgi/Faster-RCNN_TF`

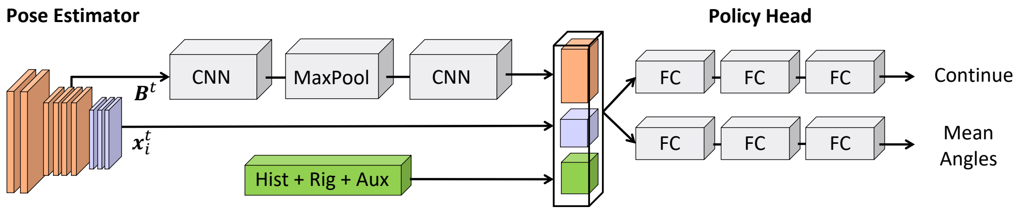


Figure 5.8: Pose-DRL network architecture. The pose estimator is shown to the left (we have shown results for two different pose estimators, DMHS and MubyNet, but any other moncoular pose estimator would work). When using the single person pose estimator DMHS, the input is a bounding box containing the target person, and its convolutional feature map $\boldsymbol{B}_t$ forms the base state of the agent. For the multi-person estimator MubyNet, the full image is instead fed as input and the associated feature map $\boldsymbol{B}_t$ is used as the base state. Next, $\boldsymbol{B}_t$ is processed by two convolutional layers with ReLU-activations (first conv: 3×3 kernel, stride 1, output dimension $21 \times 21 \times 8$; max pool: 2×2 kernel, stride 2, output dimension $11 \times 11 \times 8$; second conv: 3×3 kernel, stride 1, output dimension $9 \times 9 \times 4$). It is then concatenated with the pose prediction information $\boldsymbol{x}_i^t$ for the current active-view, a history of the last 4 fused pose estimates from previous active-views (*Hist*), camera rig information (*Rig*), containing both a description of the camera rig as well as the agent's current and previously visited viewpoints within the rig, and auxiliary information (*Aux*) with the number of actions taken and number of people detected. Note that pose information is used in the single-target mode only; for the multi-person setting we omit pose information in the state space as there may be a variable number of persons per scene. However, in this setting the agent instead has access to image level information. See more about the state space in Section 4.2 of the main paper. The concatenated state is subsequently fed to the two action branches: the *continue* branch (top) and the *viewpoint selection* branch (bottom). Both branches use tanh-activations for the hidden fully connected (FC) layers. For the *continue* action branch, the output is turned into a *continue* probability through a sigmoid-layer, cf. (5.3) in the main paper. For the *viewpoint selection* action branch, the azimuth and elevation mean angles are produced by a scaled tanh-layer, cf. (5.5) - (5.6) in the main paper. In the *continue* action branch the three FC-layers have 512, 512, and 1 output neurons each respectively, while the *viewpoint selection* action branch's three FC-layers have 1024, 512, and 2 output neurons, respectively.

sorted it into being in either the 10% best or worst cameras based on associated individual reconstruction error. The results are shown in Table 5.3. It can be seen that the agent typically selects among the best while avoiding the worst viewpoints. The viewpoint errors are more uniform for the single-people *Pose* scenes, since there are no viewpoints where the target is occluded; hence the camera selection statistics are also more uniform for *Pose*.

A.3 Handling Missed Detections or Matchings

For an overview of how we detect and match multiple people, refer to Section 3.2 in the main paper. In this section we describe what happens in case some persons are not detected or matched. For the detection-based DMHS-variant of Pose-DRL, if in a viewpoint there are no detections, or if no detection has a matching cost below the threshold C, the underlying pose estimator is computed on the entire input image to obtain a base state descriptor $\boldsymbol{B}_t$ for decision making (no associated pose is fused in this case).

It is possible that one or several persons are not detected in a single viewpoint in an active-view. In this case the pose estimate is set to the fused estimate from the previous active-view as a backup. In case a previous estimate also does not exist (could happen e.g. in the initial active-view of an active-sequence), to be able to compute a reconstruction error we set a placeholder pose estimate where each joint is equal to the center hip location of the ground truth. Naturally, this is an extremely poor and implausible estimate, but it is used only to be able to compute an error (another option would be to not include such an estimate when computing average errors, but that would not penalize the fact that a person was never detected and reconstructed).

A.4 Additional Visualizations of Pose-DRL

In Figure 5.9 - 5.10 we show additional visualizations of how Pose-DRL performs single-target pose estimation in active-views from the Panoptic [7] test set we have used in this work. We use SMPL [8] for the 3d shape models (here and in the main paper), and per-joint median averaging for fusing poses. As it is referenced in the visualizations, we repeat the equation for a partially fused pose (for the first j steps) within an active-view[6] below:

$$\tilde{\boldsymbol{x}} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i). \tag{5.10}$$

A.5 Using Pose-DRL in the Wild

The dense CMU Panoptic studio provides a useful environment for training and evaluating our proposed model. However, it is also interesting to test the model's applicability in the the real world. To this end we captured data with an off-the-shelf smartphone and used internal sensors to estimate the camera pose matrix for each image. This simple process of walking around subjects while they stand still emulates the *time-freeze* setup in Panoptic and allows us to test our model in the real world. Note that neither the 3d pose estimation network nor the policy was re-trained; only the instance detector was refined to produce accurate appearance models for the detected people. See Figure 5.11 for resulting visualizations. Finally, note that we obtained consent from the people shown.

[6]For active-sequence processing, the agent also fuses temporally by adding the previous fused estimate; see (5.2) in the main paper

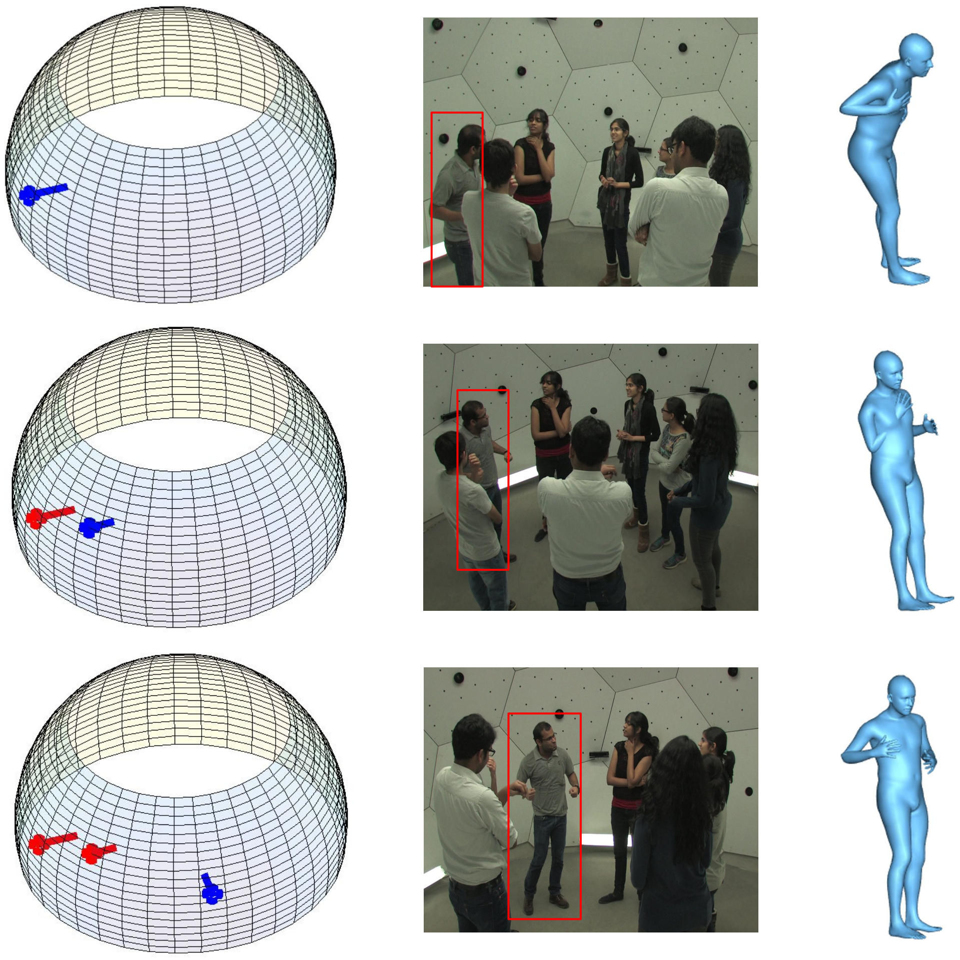


Figure 5.9: Visualization of how Pose-DRL performs single-target reconstruction on an active-view (set of viewpoints for a time-freeze) in a *Mafia* test scene. In this case the agent sees three viewpoints prior to automatically continuing to the next active-view. The reconstruction error reduces from 168 to 107 mm/joint. Left: Viewpoints seen by the agent, where blue marks the current viewpoint (camera) and red marks previous viewpoints. Note that the initial camera was given randomly. Middle: Input images associated to the viewpoints, also showing the detection bounding box of the target person in red – detections for the other people are left out to avoid visual clutter. Right: SMPL visualizations of the partially fused poses, cf. (5.10). The target person is only partially visible in the initial viewpoint, and the associated pose estimate is inaccurate with the reconstruction incorrectly tilting forward. As the agent visits more viewpoints, the stance of the reconstruction becomes straighter and more correct. The person is fully visible in the final viewpoint, and the associated final fused estimate is plausible and accurate.

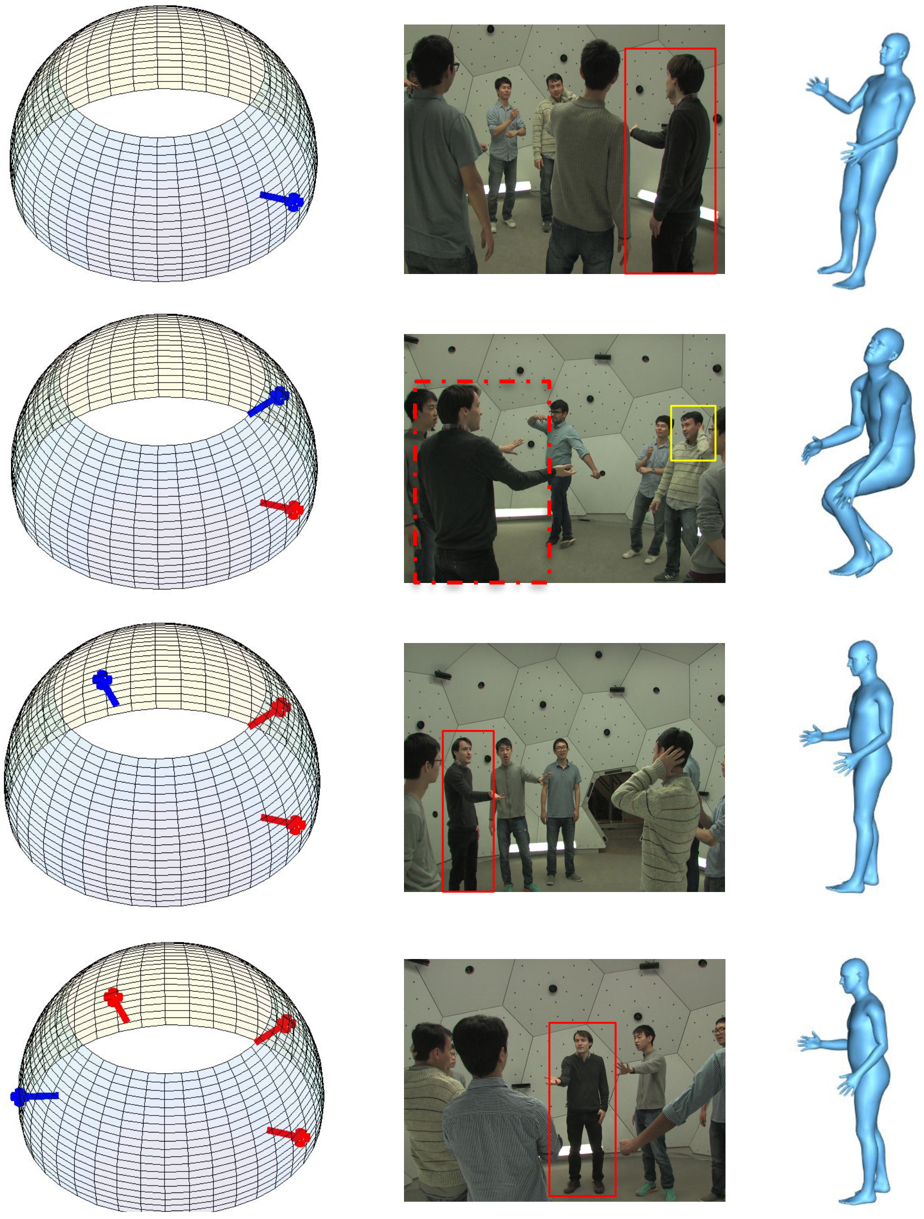


Figure 5.10: Visualization of how Pose-DRL performs single-target reconstruction on an active-view in an *Ultimatum* test scene. The target person is viewed from a suboptimal direction in the first viewpoint, which causes the associated pose estimate to be incorrectly tilted. As the agent moves to the next viewpoint to get a better view of the person, the underlying detection and matching system suggests an incorrect detection to feed the pose estimator, which causes the fused estimate to deteriorate severely. However, the agent is able to remedy this by selecting two more good and diverse viewpoints where the target is clearly visible, which yields a considerably better fused pose estimate. In this example the agent sees four viewpoints prior to automatically continuing to the next active-view. The reconstruction error reduces from 149 to 119 mm/joint.

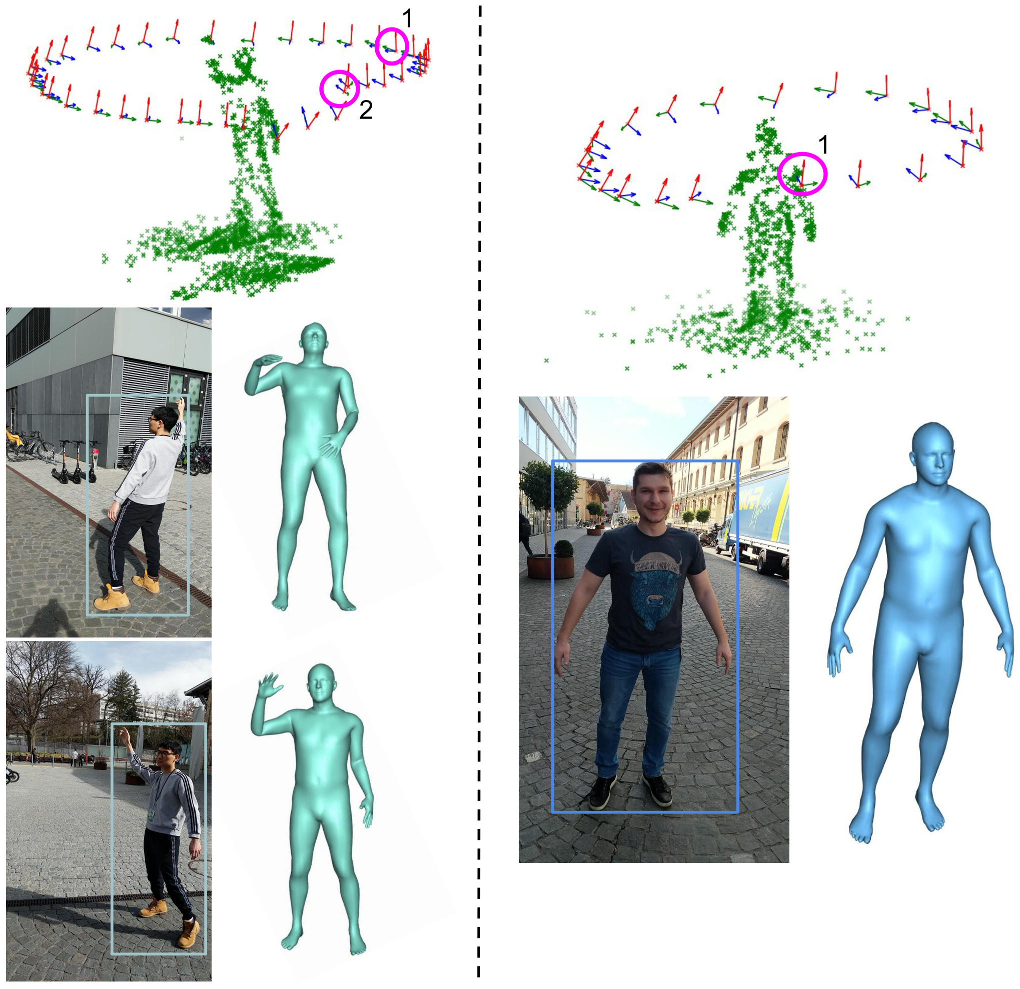


Figure 5.11: People standing in various poses, captured with a smartphone camera from different viewpoints. Note that this data is significantly different from that obtained from Panoptic, with more challenging outdoor lighting conditions, human-imposed errors from holding and directing the smartphone camera, and so on. We show two visualization of how Pose-DRL operates in different scenarios. Pose-DRL was *not* re-trained on this data; we use the same model weights as for producing the results in the main paper. In each scenario we also show the 3d configuration of the scene, as well as which viewpoints are selected by the agent and in what order (pink circles). Left: In this example the agent sees two views before terminating viewpoint selection. The initial randomly given viewpoint produces a pose estimate where the arms are not accurate, which is corrected for in the second and final viewpoint. Right: The agent receives a very good initial viewpoint and decides to terminate viewpoint selection immediately, producing an accurate pose estimate. See Section $A.5$ for more details about these visualizations.

References

[1] A.-I. Popa, M. Zanfir, and C. Sminchisescu, "Deep multitask architecture for integrated 2d and 3d human sensing," in *CVPR*, 2017.

[2] A. Zanfir, E. Marinoiu, M. Zanfir, A.-I. Popa, and C. Sminchisescu, "Deep network for the integrated 3d sensing of multiple people in natural images," in *NeurIPS*, 2018.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[7] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *ICCV*, 2015.

[8] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *SIGGRAPH*, vol. 34, no. 6, 2015.

Paper V

Embodied Visual Active Learning for Semantic Segmentation

David Nilsson[1,2] **Aleksis Pirinen**[1] **Erik Gärtner**[1,2] **Cristian Sminchisescu**[1,2]
[1]Centre for Mathematical Sciences, Lund University
[2]Google Research

Abstract

We study the task of *embodied visual active learning*, where an agent is set to explore a 3d environment with the goal to acquire visual scene understanding by actively selecting views for which to request annotation. While accurate on some benchmarks, today's deep visual recognition pipelines tend to not generalize well to certain real-world scenarios, or to unusual viewpoints. Robotic perception, in turn, requires the capability to refine the recognition capabilities for the conditions where the mobile system operates, including cluttered indoor environments or poor illumination. This motivates the proposed task, where an agent is placed in a novel environment with the objective of improving its visual recognition capability. To study embodied visual active learning, we develop a battery of agents – both learnt and pre-specified – and with different levels of knowledge of the environment. The agents are equipped with a semantic segmentation network and seek to acquire informative views, move and explore in order to propagate annotations in the neighborhood of those views, then refine the underlying segmentation network by online retraining. The trainable method uses deep reinforcement learning with a reward function that balances two competing objectives: *task performance, represented as visual recognition accuracy*, which requires exploring the environment, and the necessary *amount of annotated data* requested during active exploration. We extensively evaluate the proposed models using the photorealistic Matterport3D simulator and show that a fully learnt method outperforms comparable pre-specified counterparts, even when requesting fewer annotations.

1 Introduction

Imagine a household robot in a home in which it has never been before and equipped with a visual sensing module to perceive its environment and localize objects. If the robot fails to recognize some objects, or to adapt to changes in the environment, over time, it may not be able to properly perform its tasks. Much of the recent success of visual perception has been achieved by deep CNNs, e.g. in image classification [1, 2, 3], semantic segmentation [4, 5] and object detection [6, 7]. Such systems may however be challenged by unusual viewpoints or domains, as noted e.g. by [8] and [9]. Moreover, a mobile household robot should ideally operate with lightweight, re-trainable and task-specific perception models, rather than large and comprehensive ones, which could be demanding computationally and not tailored to the needs of a specific house.

In practice, even in closed but large environments, developing robust scene understanding by exhaustive approaches may be difficult, as looking everywhere requires an excessive amount of annotation labor. All views are however not equally informative, as a view containing many diverse objects is likely more useful than one covering a single semantic class, e.g. a wall. This suggests that in learning visual perception one does not have to label exhaustively. As new, potentially difficult arrangements appear in an evolving environment, it would be useful to identify those automatically, based on the task and demand, rather than programmatically, by periodically re-training a complete model. Moreover, the agent could make the most out of its embodiment by propagating a given ground truth annotation using motion – as measured by the perceived optical flow – in that neighborhood. The agent can then self-train, online, for increased performance. The key questions are how should one explore the environment, how to select the most informative views to annotate, and how to make the most out of them. We analyze these questions in an *embodied visual active learning* framework, illustrated in Figure 6.1.

To ground the embodied visual active learning task, in this work we measure visual perception ability as semantic segmentation accuracy. The agent is equipped with a semantic segmentation system and must move around and request annotations in order to refine it. After exploring the scene the agent should be able to accurately segment all views in the explored area. This requires an exploration policy covering different objects from diverse viewpoints and selecting sufficiently many annotations to train the perception model. The agent can also propagate annotations to different nearby viewpoints using optical flow and then self-train. We develop a battery of methods, ranging from pre-specified ones to a fully trainable deep reinforcement learning-based agent, which we evaluate extensively in the photorealistic Matterport3D environment [10].

In summary, our main contributions are:

- We study the task of *embodied visual active learning*, where an agent should explore a 3d environment to acquire visual scene understanding by actively selecting views for which to request annotation. The agent then propagates information by moving in the neighborhood of those views and self-trains.

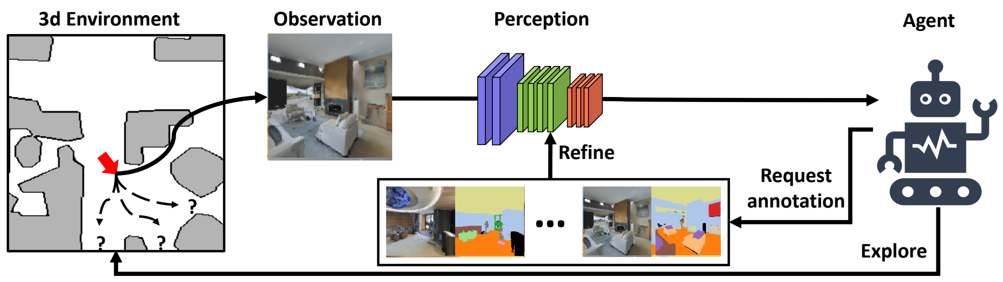


Figure 6.1: Embodied visual active learning. An agent in a 3d environment must explore and occasionally request annotation to efficiently refine its visual perception. The navigation component makes this task significantly more complex than traditional active learning, where the data pool over which the agent queries annotations, either in the form of image collections or pre-recorded video streams, is static and given.

- In our setup, visual learning and exploration can inform and guide one another since the recognition system is selectively and gradually refined during exploration, instead of being trained at the end of a trajectory on a full set of densely annotated views.
- We develop a variety of methods, both learnt and pre-specified, to tackle our task in the context of semantic segmentation.
- We perform extensive evaluation in a photorealistic 3d environment and show that a fully learnt method outperforms comparable pre-specified ones.

2 Related Work

The embodied visual active learning setup leverages several computer vision and machine learning concepts, such as embodied navigation, active learning and active vision. There is substantial recent literature on embodied agents navigating in real or simulated 3d environments, especially given the recent emergence of large-scale simulators [11, 12, 13, 14, 15]. In Embodied Question Answering [16, 17, 18], an agent is given a question, such as "What color is the car?". The agent must then explore the environment, often quite extensively, in order to be able to answer. Zhu et al. [19] and Mousavian et al. [20] task agents with reaching a target viewpoint in as few steps as possible. The agents receive the current view and the target as inputs in each step. In point-goal navigation [21, 22, 11, 23] the agent is given coordinates of a target to reach using visual information and ego-motion. In visual exploration [24, 25, 26, 27, 28, 29] the task is to explore an unknown environment as quickly as possible, by covering the whole scene area. In [8, 9], an agent is tasked to navigate an environment to increase the accuracy of a pre-trained recognition model, for example by moving around occluded objects. This is in contrast to our work where the goal is to collect views for *training* a perception model. Whereas in [8, 9] the agent is spawned close to the target object, we cannot make such assumptions, as our task is not only to accurately recognize a single object or view, but to do so for *all* views in the potentially large area explored by the agent.

There are relations also to curiosity-driven learning [30, 31], in that we also seek an agent which visits novel views (states). In [30], exploration is aided by giving rewards based on the prediction error of a self-supervised inverse-dynamics model. This is a task-independent exploration strategy useful to search 2d or 3d environments during training. In our setup, exploration is task-specific in that it is aimed specifically at refining a visual recognition system in a novel environment. Moreover, we use semi-dense rewards for both visual learning and for exploration. Hence we are not operating using entirely sparse rewards where curiosity approaches often outperform other methods.

Our work is also related to [32, 33, 34, 35]. Different from us, [32] uses hand-crafted annotation and exploration strategies, aiming to label all voxels in a 3d reconstruction by selecting a subset of frames covering all voxels. This is a form of exhaustive annotation and a visual perception system is not trained. Hence the system can only analyze objects in annotated voxels. In our setup the agent is instead tasked with both exploration and the selection of views to annotate, and we learn a perception module aiming to generalize to unseen views. In contrast to us, [33, 34, 35] do not consider an agent choosing where to move in the environment, nor which parts to label. Instead they use all views seen when following a pre-specified path for training a visual recognition system. Pot et al. [33] use an object detector obtained by self-supervised learning and clustering, while Zhong et al. [34] and Wang et al. [35] use constraints from SLAM to improve a given segmentation model. The latter approaches could in principle complement our label propagation, and are orthogonal to our main contributions.

Next-best-view (NBV) prediction [36, 37, 38, 39, 40, 41] is superficially similar to our task. In [36] an agent is trained to reveal parts of a panorama and a model is built to complete all views of the panorama. Our setup allows free movement in an environment, hence it features a navigation component which makes our task more comprehensive. While NBV typically integrates information from all predicted views, our task requires the adaptive selection of only a subset of the views encountered during the agent's navigation trajectory.

Active learning [42, 43, 44, 45, 46, 47] can be seen as the static version of our setup, as it considers approaches for learning what parts of a larger *pre-existing* and *static* training set should be fed into the training procedure, and in what order. We instead consider the active learning problem in an embodied setup, where an agent can move and actively select views for which to request annotation. Embodiment makes it possible to use motion to propagate annotations, hence effectively generate new ones at no additional annotation cost. In essence, our work lays groundwork towards marrying the active vision and active learning paradigms.

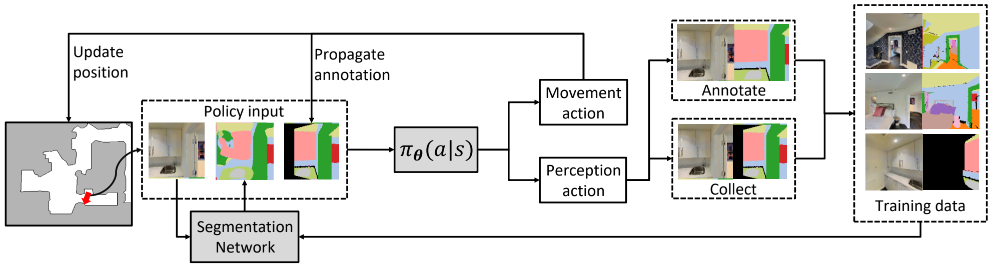


Figure 6.2: Embodied visual active learning for semantic segmentation. A first-person agent is placed in a room and a deep network predicts the semantic segmentation of the agent's view. Based on the view and its segmentation, the agent can either select a *movement action* to change position and viewpoint, or select a *perception action* (`Annotate` or `Collect`). `Annotate` adds the current view and its ground truth segmentation to the pool of training data for the segmentation network, while `Collect` is a cheaper version (no additional supervision required) where the current view and the last annotated view – propagated to the agent's current position using optical flow – is added to the training set. The propagated annotation is also a policy input for the learnt agent described in Section 3.3. After a perception action, the segmentation network is refined on the current training set. The embodied visual active learning process is considered successful if, after selecting a limited number of `Annotate` actions or an exploration budget is exhausted, the segmentation network can accurately segment any other view in the environment where the agent operates. Note that the map (left) is *not* provided as input to the learnt agent in Section 3.3.

3 Embodied Visual Active Learning

Embodied visual active learning is an interplay between a first-person agent, a 3d environment and a trainable perception module. See Figure 6.1 for a high-level abstraction and Figure 6.2 for details of the particular task considered in this paper. The perception module processes images (views) observed by the agent in the environment. The agent can request annotations for views in order to refine the perception module. It should ideally request few annotations as these are costly. The agent can also generate more annotations for free by neighborhood exploration using label propagation, such that when trained on that data the perception module becomes more accurate in the explored environment. To assess how successful an agent is on the task, we test how accurate the perception module is on multiple random viewpoints selected uniformly in the area explored by the agent.

Task overview. The agent begins each episode randomly positioned and rotated in a 3d environment, with a randomly initialized semantic segmentation network. The ground truth segmentation mask for the first view is given for the initial training of the segmentation network. The agent can choose *movement actions* (`MoveForward`, `MoveLeft`, `MoveRight`, `RotateLeft`, `RotateRight`, with 25 cm movements and 15 degree rotations), or *perception actions* (`Annotate`, `Collect`). If the agent moves or rotates, the ground truth mask is propagated using optical flow. The agent may at any time choose to insert the propagated annotation into its training set with the `Collect` action, or to ask for a new ground truth mask with the `Annotate` action. After an `Annotate` action the propagated annotation mask is re-initialized to the ground truth annotation. After each perception action, the segmentation network $\mathcal{S}$ is refined on the training set, which is first expanded with the newly acquired data point.

The agent's performance is evaluated at the end of the episode. The goal is to maximize the mIoU and mean accuracy of the segmentation network on the views in the area explored by the agent. Specifically, a set of *reference views* are randomly sampled within a disc of radius r centered at the starting location, and the segmentation network is evaluated on these. Hence to perform well the agent is required to explore its surroundings, and it should refine its perception module in regions of high uncertainty.

3.1 Methods for the Proposed Task

We develop several methods to evaluate and study the embodied visual active learning task. All methods except the RL-agent issue the `Collect` action when 30% of the propagated labels are unknown and `Annotate` when 85% are unknown. The intuition is that the pre-specified methods should request annotation when most pixels are unlabeled. The specific hyperparameters of all models were set based on a validation set.

Random. Selects uniform random movement actions. This baseline is thus a lower bound in terms of embodied exploration performance for this task.

Rotate. Continually rotates left. This method is useful in comparing with agents that move and explore, i.e. to monitor what improvements can be expected from embodiment.

Bounce. Explores by walking straight forward until it hits a wall, then samples a new random direction and moves forward until it collides with a new wall, and so on. This agent quickly explores the environment.

Frontier exploration. This method builds a map, online, by using using depth and motion from the simulator [48]. All pixels with depth within a 4m threshold are back-projected in 3d and then classified as either obstacles or navigable, based on height relative to the ground plane. This agent is confined to move within the reference view radius r, which is a choice to its advantage,[1] as annotated views will more likely be similar to reference views that reside within that same radius.

RL-agent. This fully trainable method that we develop jointly learns exploration and perception actions in a reinforcement learning framework. See full description in Section 3.3.

[1]This ensures it is evaluated under ideal conditions, in contrast to the RL-agent in Section 3.3.

Space filler. Follows a shortest space filling curve within the reference view radius r, and as r increases the entire environment is explored. This baseline makes strong and somewhat less general (or depending on the application, altogether unrealistic) assumptions in order to create a path: knowing the floor plan in advance, as well as which locations are reachable from the start. Also, it only moves within the reference view radius, and knows the shortest geodesic paths to take on the curve. Hence this method can be considered an upper bound to which other methods are compared.

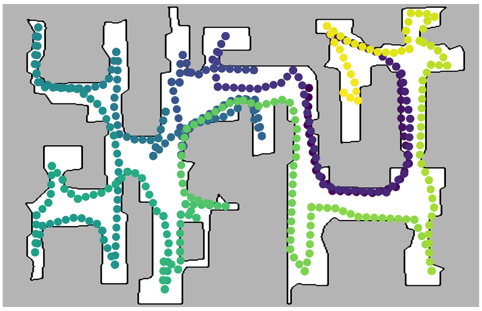


Figure 6.3: An example of a space filling curve in a Matterport3D floor plan. The space filler baseline assumes complete spatial knowledge of the environment.

The space filling curve is computed by placing a grid of nodes onto the floor plan (1m resolution, using a sampling and reachability heuristic), and then finding the shortest path around it with an approximate traveling salesman solver. Figure 6.3 shows a space filling curve in a Matterport3D floor plan.

3.2 Semantic Segmentation Network

Each method uses the same FCN-inspired deep network [4] for semantic segmentation. The network consists of 3 blocks of convolutional layers, each containing 3 convolutional layers with kernels of size 3×3. The first convolutional layer in each block uses a stride of 2, which halves the resolution. For each block the number of channels doubles, using 64, 128 and 256 channels respectively. Multiple predictions are made using the final convolutional layers of each block. The multi-scale predictions are resized to the original image resolution using bilinear interpolation and are then summed up, resulting in the final segmentation estimate. Note that we have deliberately chosen to make the network small so that it can be efficiently refined on new data.

At the beginning of each episode the parameters are initialized randomly, and we train the network on the very first view, for which we always supply the ground truth segmentation. Each time `Annotate` or `Collect` is selected, we refine the network. Minibatches of size 8, which always include the latest added labeled image, are used in training. We use random cropping and scaling for data augmentation. The network is refined either until it has trained for $1,000$ iterations or until the accuracy of a minibatch exceeds 95%. We use a standard cross-entropy loss averaged over all pixels. The segmentation network is trained using stochastic gradient descent with learning rate 0.01, weight decay 10^{-4} and momentum 0.9.

To propagate semantic labels we compute optical flow between consecutive viewpoints using PWC-Net [49]. The optical flow is computed bidirectionally and only pixels where the difference between the forward and backward displacements is less than 2 pixels are propagated [50] (we found that labels were reliably tracked over several frames when using 2 pixels as a threshold).

3.3 Reinforcement Learning Agent

To present the reinforcement learning agent for our task, we begin with an explanation of the state-action representation and policy network, followed by the reward structure and finally policy training.

Actions, states and policy. The agent is represented as a deep stochastic policy $\pi_{\boldsymbol{\theta}}(a_t|s_t)$ which samples an action a_t in state s_t at time t. The actions are `MoveForward`, `MoveLeft`, `MoveRight`, `RotateLeft`, `RotateRight`, `Annotate` and `Collect`. The full state is $s_t = \{\boldsymbol{I}_t, \boldsymbol{S}_t, \boldsymbol{P}_t, \boldsymbol{F}_t\}$ where $\boldsymbol{I}_t \in \mathbb{R}^{127\times127\times3}$ is the image, $\boldsymbol{S}_t = \mathcal{S}_t(\boldsymbol{I}_t) \in \mathbb{R}^{127\times127\times3}$ is the semantic segmentation mask predicted by the deep network $\mathcal{S}_t$ (this network is refined over an episode; t indexes the network parameters at time t), $\boldsymbol{P}_t \in \mathbb{R}^{127\times127\times3}$ is the propagated annotation, and $\boldsymbol{F}_t \in \mathbb{R}^{7\times7\times2048}$ is a deep representation of $\boldsymbol{I}_t$ (a ResNet-50 backbone feature map).

The policy consists of a *base processor*, a *recurrent module* and a *policy head*. The base processor consists of two learnable components: ϕ_{img} and ϕ_{res}. The 4-layer convolutional network ϕ_{img} takes as input the depth-wise concatenated triplet $\{\boldsymbol{I}_t, \boldsymbol{S}_t, \boldsymbol{P}_t\}$, producing $\phi_{img}(\boldsymbol{I}_t, \boldsymbol{S}_t, \boldsymbol{P}_t) \in \mathbb{R}^{512}$. Similarly, the 2-layer convolutional network ϕ_{res} yields an embedding $\phi_{res}(\boldsymbol{F}_t) \in \mathbb{R}^{512}$ of the ResNet features $\boldsymbol{F}_t$. An LSTM [51] with 256 cells constitutes the recurrent module, which takes as input $\phi_{img}(\boldsymbol{I}_t, \boldsymbol{S}_t, \boldsymbol{P}_t)$ and $\phi_{res}(\boldsymbol{F}_t)$. The input has length 1024. The hidden LSTM state is fed to the policy head, consisting of a fully-connected layer followed by a 7-way softmax which produces action probabilities.

Rewards. In training, the main reward is related to the mIoU improvement of the final segmentation network $\mathcal{S}_T$ over the initial $\mathcal{S}_0$ on a reference set $\mathcal{R}$. The set $\mathcal{R}$ is constructed at the beginning of each episode by randomly selecting views within a geodesic distance r from the agent's starting location, and contains views with corresponding ground truth semantic segmentation masks. At the end of an episode of length T, the underlying perception module is evaluated on $\mathcal{R}$. Specifically, after an episode of T steps the agent receives as final reward:

$$R_T = \text{mIoU}(\mathcal{S}_T, \mathcal{R}) - \text{mIoU}(\mathcal{S}_0, \mathcal{R}). \tag{6.1}$$

To obtain a denser signal, which is tightly coupled with the final objective, we also give a reward proportional to the improvement of $\mathcal{S}$ on the reference set $\mathcal{R}$ after each `Annotate` (*ann*) and `Collect` (*col*) action:

$$R_t^{ann} = \text{mIoU}(\mathcal{S}_t, \mathcal{R}) - \text{mIoU}(\mathcal{S}_{t-1}, \mathcal{R}) - \epsilon^{ann}, \tag{6.2}$$

$$R_t^{col} = \text{mIoU}(\mathcal{S}_t, \mathcal{R}) - \text{mIoU}(\mathcal{S}_{t-1}, \mathcal{R}). \tag{6.3}$$

To ensure that the agent does not request costly annotations too frequently, each `Annotate` action is penalized with a negative reward $-\epsilon^{ann}$ (we set $\epsilon^{ann} = 0.01$), as seen in (6.2). Such a penalty is *not* given for the free `Collect` action. Moreover, the dataset we use has 40 different semantic classes, but some are very rare and apply only to small objects, and some might not even be present in certain houses. We address this imbalance by computing the mIoU using only the 10 largest classes, ranked by the number of pixels in the set of reference views for the current episode.

While the rewards (6.1) - (6.3) should implicitly encourage the agent to explore the environment in order to request annotations for distinct, informative views, we empirically found it useful to include an additional explicit exploration reward. Denote by $\{\boldsymbol{x}_i\}_{i=1}^{t-1} = \{(x_i, y_i)\}_{i=1}^{t-1}$ the positions the agent has visited up to time $t-1$ in its current episode, and let $\boldsymbol{x}_t = (x_t, y_t)$ denote its current position. We define the exploration (*exp*) reward based on a kernel density estimate of the agent's visited locations:

$$R_t^{exp} = a - bp_t(\boldsymbol{x}_t) := a - \frac{b}{t-1}\sum_{i=1}^{t-1} k(\boldsymbol{x}, \boldsymbol{x}_i), \tag{6.4}$$

where a and b are hyperparameters (both set to 0.003). Here $p_t(\boldsymbol{x}_t)$ is a Gaussian kernel estimate of the density with bandwidth 0.3m. It is large for previously visited positions and small for unvisited positions, thereby encouraging the agent's expansion towards new places in the environment. The exploration reward is only given for movement actions. Note that the pose $\boldsymbol{x}_i$ is only used to compute the reward R_t^{exp} and is not available to the policy via the state space.

Policy training. The policy network is trained using PPO [52] based on the RLlib reinforcement learning package [53], as well as OpenAI Gym [54]. For optimization we use Adam [55] with batch size 512, learning rate 10^{-4} and discount rate 0.99. During training, each episode consists of 256 actions. The agent is trained for 4k episodes, which totals 1024k steps.

Our system is implemented in TensorFlow [56], and it takes about 3 days to train an agent using 4 Nvidia Titan X GPUs. An episode of length 256 took on average about 3 minutes using a single GPU, and during training we used 4 workers with one GPU each, collecting rollouts independently. The runtime per episode varies depending on how frequently the agent decides to annotate, as training the segmentation network is the bottleneck and accounts for approximately 90% of the runtime. We use optical flow from the simulator to speed up policy training. For evaluation, the RL-agent and all other methods use PWC-Net to compute optical flow. The ResNet-50 feature extractor is pre-trained on ImageNet [57] with weights frozen during policy training.

4 Experiments

In this section we provide empirical evaluations of various methods. The primary metrics are mIoU and segmentation accuracy but we emphasize that we test the exploration and annotation selection capability of *policies* – the mIoU and accuracy measure how well agents explore in order to refine their perception. Different from accuracy, the mIoU does not become excessively high by simply segmenting large background regions (such as walls), and hence it is more representative of overall semantic segmentation quality.

Experimental setup. We evaluate the methods on the Matterport3D dataset [10] using the embodied agent framework Habitat [11]. This setup allows the agent to freely explore photorealistic 3d models of large houses, that have ground truth annotations with 40 diverse semantic classes. Hence it is a suitable environment for evaluation. To assess the generalization capability of the RL-agent we train and test it in *different* houses. We use the same 61, 11 and 18 houses for training, validation and testing as [10]. The RL-agent and all pre-specified methods except the space filler are *comparable* in terms of assumptions, cf. Section 3.1. The space filler assumes full spatial knowledge of the environment (ground truth map) and hence has inherent advantages over the other methods.

During RL-agent training we randomly sample starting positions and rotations from the training houses at the start of each episode. An episode ends after 256 actions. Hyperparameters of the learnt and pre-specified agents are tuned on the validation set. For validation and testing we use 3 and 4 starting positions per scene, respectively, so each agent is tested for a total of 33 episodes in validation and 72 episodes in testing. The reported metrics are the mean over all these runs. All methods are evaluated on the same starting positions in the same houses. The reference views used to evaluate the semantic segmentation performance are obtained by sampling 32 random views within a 5m geodesic distance of the agent's starting position at the beginning of each episode. In training the reference views are sampled randomly. During validation and testing, for fairness, we sample the same views for a given starting position when we test different agents. Note that there is no overlap between reference views during policy training and testing, since training, validation and testing houses are non-overlapping.

Recall that the RL-agent's policy parameters are denoted $\boldsymbol{\theta}$. We now let $\boldsymbol{\theta}_{seg}$ denote the parameters of the underlying semantic segmentation network, in order to clarify when we reset, freeze and refine $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{seg}$, respectively. In RL-training we refine $\boldsymbol{\theta}$ during policy estimation in the training houses. When we evaluate the policy on the validation or test houses we freeze $\boldsymbol{\theta}$ and only use the policy for inference. The parameters of the segmentation network $\boldsymbol{\theta}_{seg}$ are always reset at the beginning of an episode, regardless of which house we deploy the agent in and whether the policy network is training or not. During an episode we refine $\boldsymbol{\theta}_{seg}$ exactly when the agent selects the `Annotate` or `Collect` actions (this applies to all methods described in Section 3.1). Thus annotated views in an episode are used to refine $\boldsymbol{\theta}_{seg}$ in that episode only, and are not used in any other episodes.

Table 6.1: Comparison of different agents for a fixed episode length of 256 actions on the Matterport3D test scenes. The RL-agent gets higher mIoU using far fewer annotations than comparable pre-specified methods, implying that the RL-agent selects more informative views to annotate.

Method	mIoU	Accuracy	# Annotations	# Collects
Space filler	0.439	0.769	24.7	23.9
RL-agent	0.394	0.727	16.7	15.2
Frontier exploration	0.385	0.735	24.2	21.6
Bounce	0.357	0.708	29.6	26.0
Rotate	0.295	0.661	34.3	32.7
Random	0.204	0.566	29.1	19.5

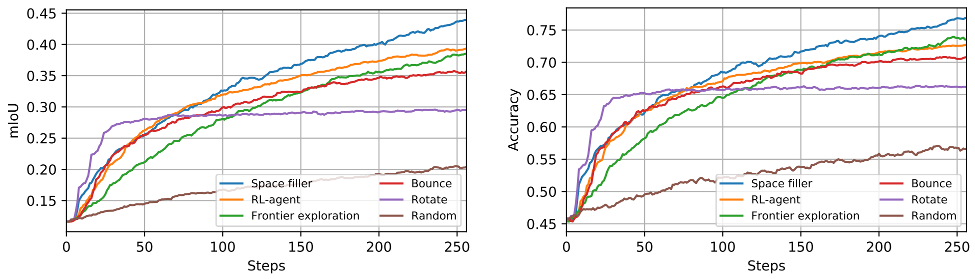


Figure 6.4: Mean segmentation accuracy and mIoU versus number of actions (steps), evaluated on the Matterport3D test scenes. The RL-agent was trained on 256-step episodes. This agent fairly quickly outperforms all other comparable pre-specified agents. *Rotate* is strong initially since it quickly gathers many annotations in a 360 degree arc, but is eventually outperformed by most other methods that move around in the houses. Frontier exploration yields similar accuracy as the RL-agent after about 170 steps, but uses significantly more annotations (cf. Table 6.1) and assumes perfect pose and depth information. The space filler, which assumes full knowledge of the environment, yields the best results after about 100 steps.

4.1 Main Results

We measure the performances of the agents in two settings: i) with unlimited annotations but limited total actions (max 256, as during RL-training), or ii) for a limited annotation budget (max 100) but unlimited total actions. All methods were tuned on the validation set in a setup similar to i) with 256-step episodes. Note however that the number of annotations can differ for different methods in a 256-step episode. The setup ii) is used to assess how the different methods compare for a fixed number of annotations.

Fixed episode length. Table 6.1 and Figure 6.4 show results on the test scenes for episodes of length 256. The RL-agent outperforms the comparable pre-specified methods in mIoU and accuracy, although frontier exploration – which uses perfect pose and depth information, and is idealized to always move within the reference view radius – yields similar accuracy after about 170 steps. The RL-agent uses much fewer annotations than other methods,

Table 6.2: Comparison of different agents for a fixed budget of 100 annotations on Matterport3D test scenes. The RL-agent gets a higher mIoU than comparable pre-specified agents, despite not being trained in this setting.

Method	mIoU	Accuracy	# Steps	# Collects
Space filler	0.600	0.863	1048	91
RL-agent	0.507	0.796	1541	94
Frontier exploration	0.485	0.796	998	84
Bounce	0.464	0.776	861	87
Rotate	0.303	0.668	752	96
Random	0.242	0.595	910	64

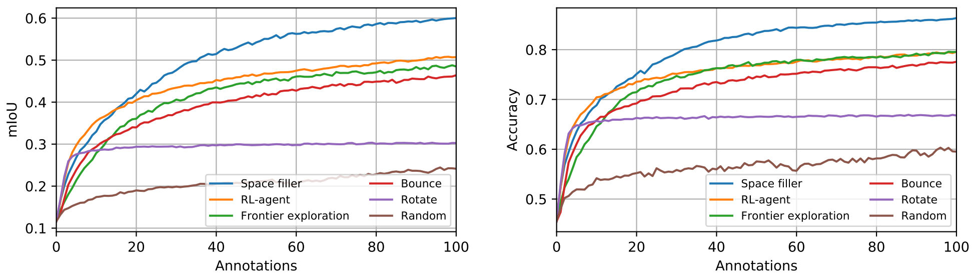


Figure 6.5: Mean segmentation accuracy and mIoU vs. number of requested annotations, evaluated on the Matterport3D test scenes. The RL-agent outperforms all comparable pre-specified methods (although frontier exploration matches it in accuracy after about 40 annotations), indicating that it has learnt an exploration policy which generalizes to novel scenes. The space filler outperforms the RL-agent as expected, except for less than 15 annotations. Thus the RL-agent is best before and around its training regime, where it on average annotates 16.7 times per episode, cf. Table 6.1.

hence those annotated views are more informative. The space filler, which assumes perfect knowledge of the map, outperforms the RL-agent but uses significantly more annotations. Note that the *Rotate* baseline saturates, supporting the intuition that an agent has to move around in order to increase performance in complex environments.

Fixed annotation budget. In Table 6.2 and Figure 6.5 we show test results when the annotation budget is limited to 100 images per episode. As expected, the space filler yields the best results, although the RL-agent gets comparable performance when using up to 15 annotations. The RL-agent outperforms comparable pre-specified methods in mIoU and accuracy. Frontier exploration obtains similar accuracy. We also see that the episodes of the RL-agent are longer.

Qualitative examples. Figure 6.6 shows examples of views for which the RL-agent requests annotations. The agent explores large parts of the space and the annotated views are diverse, both in their spatial locations and in the types of semantic classes they contain. Figure 6.7

Table 6.3: Ablation study of different RL-based model variants for 256-step episodes on the validation set. The full RL-agent outperforms all ablated models at a comparable or lower number of requested annotations.

Variant	mIoU	Accuracy	# Annotations	# Collects
Full model	0.427	0.732	16.4	16.4
No collect nor $\boldsymbol{P}_t$	0.415	0.727	17.9	0.0
Only exploration	0.411	0.727	16.1	14.4
$R_t^{exp} = 0$	0.401	0.719	17.7	47.4
No ϕ_{img}	0.378	0.696	14.3	3.8
No ResNet	0.375	0.705	23.3	0.3

shows how the segmentation network's performance on two reference views improves during an episode. The two views are initially poorly segmented, but as the agent explores and acquires annotations for novel views, the accuracy on the reference views increases.

4.2 Ablation Studies of the RL-agent

Ablation results for the RL-agent on the validation set are in Table 6.3. We compare to the following versions: i) policy without visual features ϕ_{img}; ii) policy without ResNet features ϕ_{res}; iii) no additional exploration reward (6.4), i.e. $R_t^{exp} = 0$; iv) no `Collect` action and $\boldsymbol{P}_t$ is not an input to ϕ_{img}; and v) only exploration trained, using the heuristic strategy for annotations. We trained the ablated models for 4k episodes as for the full model.

Both the validation accuracy and mIoU are higher for the full RL-model compared to all ablated variants, justifying design choices. The model not relying on propagating annotations and using the `Collect` action performs somewhat worse than the full model despite a comparable amount of annotations. The learnt annotation strategy yields higher mIoU and accuracy compared to the heuristic one, at a comparable number of annotations. The exploration reward is important in encouraging the agent to navigate to unvisited positions – without it performance is worse, despite a comparable number of annotations. Moreover, the agent trained without the exploration reward uses an excessive number of `Collect` actions, so this agent often stands still instead of moving. Finally, omitting either visual or ResNet features from the policy significantly harms accuracy for the resulting perception system.

4.3 Analysis of Annotation Strategies

In this section we examine how different annotation strategies affect the task performance on the validation set for the space filler and bounce methods, on episodes of length 256. Specifically, the annotation strategies are:

- **Threshold perception.** This is the variant evaluated in Section 4.1, i.e. it issues the `Collect` action when 30% of the propagated labels are unknown and `Annotate` when 85% are unknown.

- **Learnt perception.** We train a simplified RL-agent where the movement actions are restricted to follow the exploration trajectory of the baseline method (space filler and bounce, respectively). This model has three actions: move along the baseline exploration path, `Annotate` and `Collect`. All other training settings are identical to the full RL-agent.

- **Random perception.** In each step, this variant follows the baseline exploration trajectory with 80% probability, while annotating views and collecting propagated labels with 10% probability each.

As can be seen in Table 6.4, the best results for the space filler are obtained by using the threshold strategy, which also annotates slightly less frequently than other variants. Using learnt perception actions yields better results compared to random perception actions, and uses slightly fewer annotations per episode. Similar results carry over to the bounce method in Table 6.5, i.e. the best results are again obtained by the threshold variant. The model with a learnt annotation strategy fails to converge to anything better than heuristic perception strategies. In fact, it converges to selecting `Collect` almost 40% of the time, which indicates a lack of movement for this variant.

In Table 6.3 we saw that a learnt exploration method with a heuristic annotation strategy yields worse results than a fully learnt model. Conversely, the results from Table 6.4 - 6.5 show that a heuristic exploration method using a learnt annotation strategy yields worse results than an entirely heuristic model. Together these results indicate that it is necessary to learn how to annotate and explore jointly to provide the best results, given comparable environment knowledge.

Table 6.4: Results on the validation set for different model variants of the space filler method. The threshold perception strategy – which is the one used in the main evaluations in Section 4.1 – yields the best results.

Variant	mIoU	Accuracy	# Annotations	# Collects
Threshold perception	0.472	0.770	20.8	19.9
Learnt perception	0.454	0.755	22.8	37.4
Random perception	0.446	0.747	24.2	24.4

Table 6.5: Results on the validation set for different model variants of the bounce method. The threshold perception strategy – which is the one used in the main evaluations in Section 4.1 – yields the best results, but also uses the largest amount of annotations on average.

Variant	mIoU	Accuracy	# Annotations	# Collects
Threshold perception	0.388	0.706	27.4	24.5
Learnt perception	0.375	0.699	14.6	98.8
Random perception	0.379	0.698	25.9	24.6

Table 6.6: Results on the validation set of different training regimes for the semantic segmentation network. A pre-trained segmentation network generalizes poorly to unseen environments (first row), and there is relatively little gain for the RL-agent by having a pre-trained segmentation network (third row). Note that pre-training uses over 1000x more annotations compared to performing embodied active visual learning from scratch.

Variant	mIoU	Accuracy	# Annotations	# Colllects
Pre-train, no RL	0.208	0.549	20k	0.0
No pre-train, RL	0.427	0.732	16.4	16.4
Pre-train, RL	0.461	0.780	20k+14.4	13.3

4.4 Pre-training the Segmentation Network

Recall that our semantic segmentation network is randomly initialized at the beginning of each episode. In this section we evaluate the effect of instead pre-training the segmentation network[2] on the 61 training houses using about $20,000$ random views. In Table 6.6 we compare using this pre-trained segmentation network as initialization for the RL-agent with the case of random initialization, on episodes of length 256. We also show results when not further fine-tuning the pre-trained segmentation network, i.e. when not performing any embodied visual active learning.

The weak result obtained when not fine-tuning (first row) indicates significant appearance differences between the houses. This is further suggested by the fact that the RL-agent gets a surprisingly modest boost from pre-training the segmentation network (third row vs second row). Note the different number of annotated views used here – the agent without pre-training uses only 16.4 views on average, while the other one uses about $20,000 + 14.4$ annotated views, if we count all the images used for pre-training. Due to the relatively marginal gains for a large number of annotated images, we decided to evaluate all agents without pre-training the segmentation network.

5 Conclusions

In this paper we have explored the *embodied visual active learning* task for semantic segmentation and developed a diverse set of methods, both pre-designed and learning-based, in order to address it. The agents can explore a 3d environment and improve the accuracy of their semantic segmentation networks by requesting annotations for informative viewpoints, propagating annotations via optical flow at no additional cost by moving in the neighborhood of those views, and self-training. We have introduced multiple baselines as well as a more sophisticated fully learnt model, each exposing different assumptions and knowledge of the environment. Through extensive experiments in the photorealistic Matterport3D environment we have thoroughly investigated the various methods and shown

[2]In this pre-training experiment, we use the same architecture and hyperparameters for the segmentation network as when it is trained and deployed in the embodied visual active learning task.

that the fully learning-based method outperforms comparable non-learnt approaches, both in terms of accuracy and mIoU, while relying on fewer annotations.

Acknowledgments: This work was supported in part by the European Research Council Consolidator grant SEED, CNCS-UEFISCDI PN-III-P4-ID-PCE-2016-0535 and PCCF-2016-0180, the EU Horizon 2020 Grant DE-ENIGMA, Swedish Foundation for Strategic Research (SSF) Smart Systems Program, as well as the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

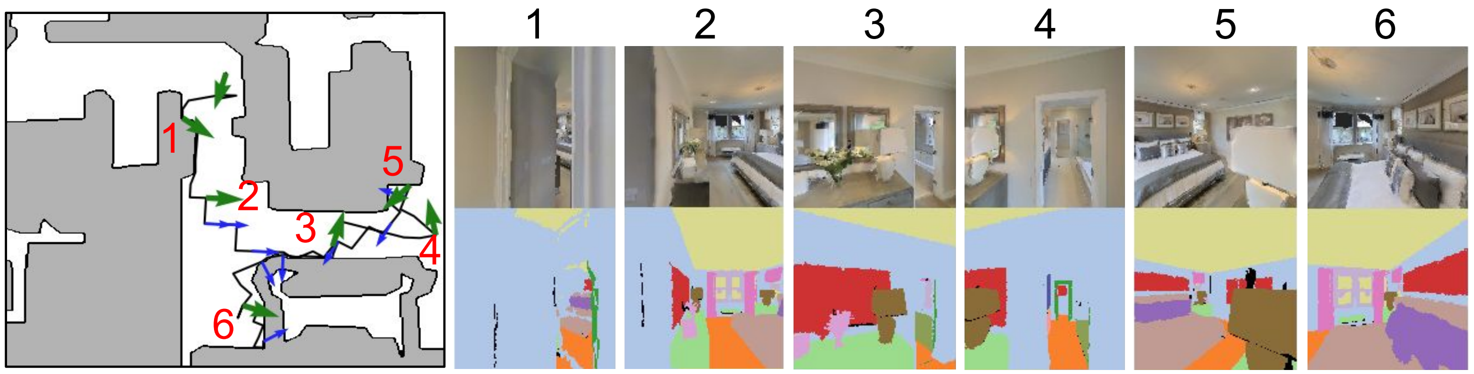


Figure 6.6: The first six requested annotations by the RL-agent in a room from the test set. Left: Map showing the agent's trajectory and the six first requested annotations (green arrows). The initially given annotation is not indicated with a number. Blue arrows indicate `Collect` actions. Right: For each annotation (numbered 1 - 6) the figures show the image seen by the agent and the ground truth received when the agent requested annotations. As can be seen, the agent quickly explores the room and requests annotations containing diverse semantic classes.

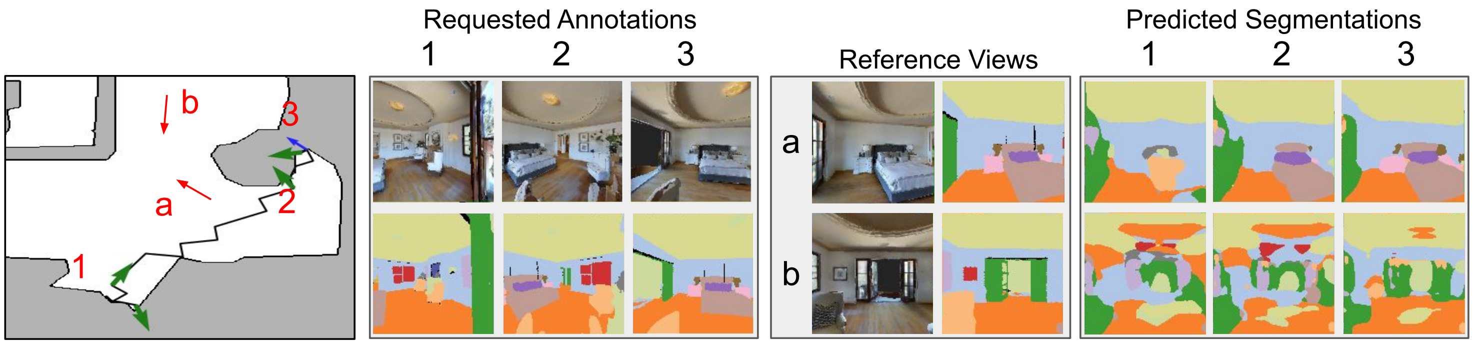


Figure 6.7: Example of the RL-agent's viewpoint selection and how its perception improves over time. We show results of two reference views after the first three annotations of the RL-agent. Left: Agent's movement path is drawn in black on the map. The annotations (green arrows) are numbered 1 - 3, and the associated views are shown immediately right of the map (the initially given annotation is not shown). Red arrows labeled *a* - *b* indicate the reference views. Right: Reference views and ground truth masks, followed by predicted segmentation after one, two and three annotations. Note the clear segmentation improvements as the agent requests more annotations. Specifically, note how reference view *a* improves drastically with annotation #2 as the bed is visible in that view, and with annotation #3 where the drawer is seen. Also note how the segmentation improves for reference view *b* after the door is seen in annotation #3.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *PAMI*, vol. 40, no. 4, 2017.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[8] P. Ammirato, P. Poirson, E. Park, J. Košecká, and A. C. Berg, "A dataset for developing and benchmarking active vision," in *ICRA*, 2017.

[9] J. Yang, Z. Ren, M. Xu, X. Chen, D. J. Crandall, D. Parikh, and D. Batra, "Embodied amodal recognition: Learning to move to perceive objects," in *ICCV*, 2019.

[10] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision*, 2017.

[11] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, *et al.*, "Habitat: A platform for embodied ai research," in *ICCV*, 2019.

[12] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv preprint arXiv:1712.05474*, 2017.

[13] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: real-world perception for embodied agents," in *CVPR*, 2018.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *CoRL*, 2017.

[15] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "MINOS: Multimodal indoor simulator for navigation in complex environments," *arXiv:1712.03931*, 2017.

[16] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, "Embodied question answering," in *CVPR*, 2018.

[17] E. Wijmans, S. Datta, O. Maksymets, A. Das, G. Gkioxari, S. Lee, I. Essa, D. Parikh, and D. Batra, "Embodied Question Answering in Photorealistic Environments with Point Cloud Perception," in *CVPR*, 2019.

[18] L. Yu, X. Chen, G. Gkioxari, M. Bansal, T. L. Berg, and D. Batra, "Multi-target embodied question answering," in *CVPR*, 2019.

[19] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*, 2017.

[20] A. Mousavian, A. Toshev, M. Fišer, J. Košecká, A. Wahid, and J. Davidson, "Visual representations for semantic target driven navigation," in *ICRA*, 2019.

[21] D. Mishkin, A. Dosovitskiy, and V. Koltun, "Benchmarking classic and learned navigation in complex 3d environments," *arXiv preprint arXiv:1901.10915*, 2019.

[22] A. Sax, B. Emi, A. R. Zamir, L. J. Guibas, S. Savarese, and J. Malik, "Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies.," in *CoRL*, 2019.

[23] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *CVPR*, 2017.

[24] S. K. Ramakrishnan, D. Jayaraman, and K. Grauman, "An exploration of embodied visual exploration," *IJCV*, 2021.

[25] K. Fang, A. Toshev, L. Fei-Fei, and S. Savarese, "Scene memory transformer for embodied agents in long-horizon tasks," in *CVPR*, 2019.

[26] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," in *ICLR*, 2019.

[27] W. Qi, R. T. Mullapudi, S. Gupta, and D. Ramanan, "Learning to move with affordance maps," in *ICLR*, 2020.

[28] L. Zheng, C. Zhu, J. Zhang, H. Zhao, H. Huang, M. Niessner, and K. Xu, "Active scene understanding via online semantic reconstruction," in *Computer Graphics Forum*, vol. 38, 2019.

[29] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," in *ICLR*, 2020.

[30] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[31] H.-K. Yang, P.-H. Chiang, K.-W. Ho, M.-F. Hong, and C.-Y. Lee, "Never forget: Balancing exploration and exploitation via learning optical flow," *arXiv preprint arXiv:1901.08486*, 2019.

[32] S. Song, L. Zhang, and J. Xiao, "Robot in a room: Toward perfect object recognition in closed environments," *CoRR, abs/1507.02703*, 2015.

[33] E. Pot, A. Toshev, and J. Kosecka, "Self-supervisory signals for object discovery and detection," *arXiv preprint arXiv:1806.03370*, 2018.

[34] F. Zhong, S. Wang, Z. Zhang, and Y. Wang, "Detect-slam: Making object detection and slam mutually beneficial," in *WACV*, pp. 1001–1010, IEEE, 2018.

[35] K. Wang, Y. Lin, L. Wang, L. Han, M. Hua, X. Wang, S. Lian, and B. Huang, "A unified framework for mutual improvement of slam and semantic segmentation," in *ICRA*, 2019.

[36] D. Jayaraman and K. Grauman, "Learning to look around: Intelligently exploring unseen environments for unknown tasks," in *CVPR*, 2018.

[37] B. Xiong and K. Grauman, "Snap angle prediction for 360 panoramas," in *ECCV*, 2018.

[38] E. Johns, S. Leutenegger, and A. J. Davison, "Pairwise decomposition of image sequences for active multi-view recognition," in *CVPR*, 2016.

[39] D. Jayaraman and K. Grauman, "Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion," in *ECCV*, 2016.

[40] S. Song, A. Zeng, A. X. Chang, M. Savva, S. Savarese, and T. Funkhouser, "Im2pano3d: Extrapolating 360 structure and semantics beyond the field of view," in *CVPR*, 2018.

[41] E. Gärtner, A. Pirinen, and C. Sminchisescu, "Deep reinforcement learning for active human pose estimation.," in *AAAI*, 2020.

[42] B. Settles, "Active learning literature survey," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2009.

[43] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in *EMNLP*, 2017.

[44] E. Lughofer, "Single-pass active learning with conflict and ignorance," *Evolving Systems*, vol. 3, no. 4, 2012.

[45] M. Woodward and C. Finn, "Active one-shot learning," *NeurIPS Deep RL Workshop*, 2016.

[46] A. Pardo, M. Xu, A. Thabet, P. Arbelaez, and B. Ghanem, "Baod: Budget-aware object detection," *arXiv preprint arXiv:1904.05443*, 2019.

[47] D. Feng, X. Wei, L. Rosenbaum, A. Maki, and K. Dietmayer, "Deep active learning for efficient training of a lidar 3d object detector," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[48] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *CIRA*, 1997.

[49] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *CVPR*, 2018.

[50] N. Sundaram, T. Brox, and K. Keutzer, "Dense point trajectories by gpu-accelerated large displacement optical flow," in *ECCV*, 2010.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[53] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *ICML*, 2018.

[54] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[56] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[57] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009.

A Supplementary Material

This supplementary material provides additional insights and results for our proposed embodied visual active learning framework.

A.1 Model Architectures

Figure 6.8 and Figure 6.9 contain details of the semantic segmentation network and the RL-agent's policy network, respectively.

A.2 Pseudo Code

The full procedure of our RL-model for embodied visual active learning for semantic segmentation is described in Algorithm 1.

A.3 Additional Variants of Bounce

We tried three additional annotation strategies of the bounce baseline on the validation set and in Table 6.7 present the results. The three variants are:

- **New direction 1.** Recall that the bounce method samples a random rotation after bouncing in a wall and then moves in that direction. This version annotates prior to walking in a new direction. It issues no `Collect` actions.
- **New direction 2.** Issues `Annotate` after rotating towards a new direction, and issues `Collect` four steps (1m) after that.
- **New direction 3.** Issues `Annotate` after rotating towards a new direction, and issues `Annotate` and `Collect` with 10% probability each when walking forward.

We see that the third version – with more frequent annotations and collects compared to versions 1 and 2 – obtains the best performance in terms of accuracy and mIoU on the validation set for 256-step episodes. It does not outperform the threshold strategy, although it annotates significantly less freqently.

Table 6.7: Results for different model variants of the bounce method (mean on the validation scenes). Threshold perception (used in the main paper) yields the highest mIoU and accuracy.

Variant	mIoU	Accuracy	# Annotations	# Collects
Threshold perception	0.388	0.706	27.4	24.5
New direction 1	0.353	0.677	11.4	0.0
New direction 2	0.372	0.695	9.8	9.6
New direction 3	0.381	0.701	20.0	10.4

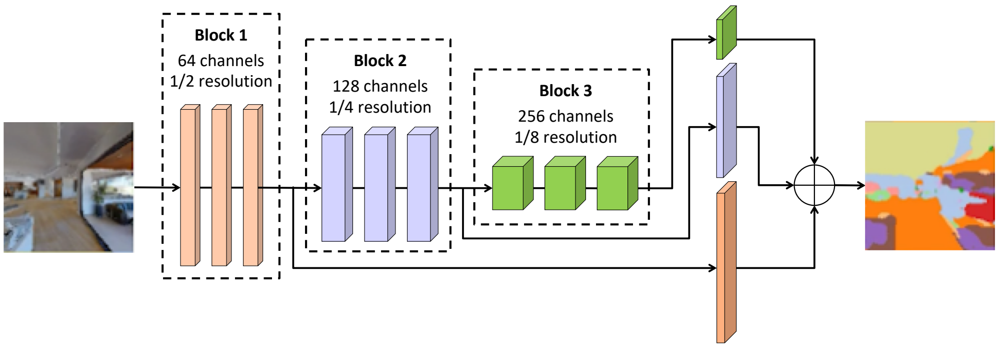


Figure 6.8: Architecture of the deep network we use for semantic segmentation. The input image is processed sequentially through three blocks, each containing three convolutional layers. The first convolutional layer in each block uses a stride of 2, which halves the resolution for each block, and at the same time the number of channels is doubled. The semantic segmentation is predicted for multiple resolutions and are summed together to predict the semantic segmentation.

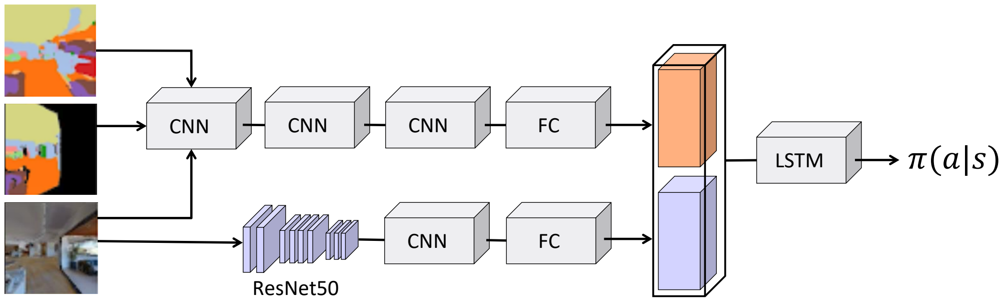


Figure 6.9: The policy network architecture for the RL-agent. The network has three inputs: the current RGB image $\boldsymbol{I} \in \mathbb{R}^{127\times127\times3}$ (bottom), the current segmentation prediction $\boldsymbol{S} \in \mathbb{R}^{127\times127\times3}$ (top), and the current optical flow propagated ground truth segmentation $\boldsymbol{P} \in \mathbb{R}^{127\times127\times3}$ (middle). All three inputs are stacked depth-wise and then processed by three convolutional layers and a fully connected layer (this processing subnetwork is called ϕ_{img} in Section 3.3 of the main paper). The first layer has 32 filters, kernel size 8×8, and stride 4. The second layer has 64 filters, kernel size 4×4, and stride 2. The third layer has 64 filters, kernel size 3×3, and stride 1. Finally, the fully connected layer has 512 outputs. In addition, the RGB image is passed through an image feature extractor (ResNet-50), called ϕ_{res} with output $\boldsymbol{F}_t$ in the main paper. The deep features $\boldsymbol{F}_t$ are subsequently passed through a convolutional layer with 128 filters, kernel size 2×2 and stride 2. Finally, these features are processed by a fully connected layer with 512 outputs. These two input branches are then concatenated and fed to an LSTM with 256 cells. The hidden state of the LSTM is finally passed to a softmax layer to produce the action distribution.

Algorithm 1 Procedural code for the RL-agent in the embodied visual active learning for semantic segmentation task.

```
1: Initialize parameters of the segmentation network $\mathcal{S}$
2: Initialize location $(x_1, y_1)$ and rotation $\phi_1$ randomly and let $\boldsymbol{x}_1 = (x_1, y_1, \phi_1)$
3: Extract image $\boldsymbol{I}_1$ and receive associated annotation mask $\boldsymbol{A}_1$ at $\boldsymbol{x}_1$; initialize training set $\mathcal{D} = \{(\boldsymbol{I}_1, \boldsymbol{A}_1)\}$
4: Perform initial training of $\mathcal{S}$ on $\mathcal{D}$
5: Initialize propagated annotation $\boldsymbol{P}_1 = \boldsymbol{A}_1$
6: Compute segmentation $\boldsymbol{S}_1 = \mathcal{S}(\boldsymbol{I}_1)$ and deep features $\boldsymbol{F}_1$
7: Initialize agent state $s_1 = (\boldsymbol{I}_1, \boldsymbol{S}_1, \boldsymbol{P}_1, \boldsymbol{F}_1)$
8: for $t = 1, \dots, T$ do
9:     Sample action $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$
10:    if $a_t \in \{\texttt{MoveForward}, \texttt{MoveLeft}, \texttt{MoveRight}, \texttt{RotateLeft}, \texttt{RotateRight}\}$ then
11:        Set $\boldsymbol{x}_{t+1}$ according to movement
12:        Propagate annotation $\boldsymbol{P}_{t+1} = \text{flow}(\boldsymbol{P}_t)$
13:    else
14:        Set $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t$
15:        Set $\boldsymbol{P}_{t+1} = \boldsymbol{P}_t$
16:    end if
17:    Obtain view $\boldsymbol{I}_{t+1}$ associated to $\boldsymbol{x}_{t+1}$
18:    if $a_t = \texttt{Annotate}$ then
19:        Obtain annotation mask $\boldsymbol{A}_{t+1}$ at $\boldsymbol{x}_{t+1}$
20:        Update training set $\mathcal{D} = \mathcal{D} \cup \{(\boldsymbol{I}_{t+1}, \boldsymbol{A}_{t+1})\}$
21:        Refine $\mathcal{S}$ on $\mathcal{D}$
22:        Reset propagated annotation $\boldsymbol{P}_{t+1} = \boldsymbol{A}_{t+1}$
23:    else if $a_t = \texttt{Collect}$ then
24:        Update training set $\mathcal{D} = \mathcal{D} \cup \{(\boldsymbol{I}_{t+1}, \boldsymbol{P}_{t+1})\}$
25:        Refine $\mathcal{S}$ on $\mathcal{D}$
26:    end if
27:    Compute segmentation $\boldsymbol{S}_{t+1} = \mathcal{S}(\boldsymbol{I}_{t+1})$ and deep features $\boldsymbol{F}_{t+1}$
28:    Update agent state $s_{t+1} = (\boldsymbol{I}_{t+1}, \boldsymbol{S}_{t+1}, \boldsymbol{P}_{t+1}, \boldsymbol{F}_{t+1})$
29: end for
30: return $\mathcal{S}_\star$ (trained segmentation network)
```