Klemens Kasseroller, BSc

Reinforcement Learning for Multi-Landmark Localization in Medical Applications

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Institute of Computer Graphics and Vision, Graz University of Technology

Advisor

Dipl.-Ing. Dr.techn. Darko Štern
Department of Biophysics, Medical University of Graz

Graz, Austria, Oct. 2020

Abstract

Automatic localization of anatomical landmarks is an important step for a wide range of applications in Medical Image Processing (MIP). Many *MIP* algorithms such as segmentation or registration benefit from the reliable localization of anatomical landmarks as a preprocessing step. When working with large 3D medical images, current state-of-the-art methods that are mainly based on Convolutional Neural Networks (CNNs) using heat map regression suffer from two major limitations. Either they require large amounts of memory to store the whole image inside the *GPU* or, using patch-based approaches, depend on an additional model for global guidance.

This thesis focuses on the automated localization of multiple landmarks using an artificial Reinforcement Learning (RL) agent, which learns to find the optimal path to the target landmark by interacting with the environment, in our case 2D X-ray images. *RL* agents are able to move through the environment by observing local image patches and learning a representation of the anatomy. We present an algorithm for single landmark localization based on Deep Deterministic Policy Gradients (DDPG). With the *DDPG* algorithm we are able to save computational resources, compared to existing *RL* based algorithms for landmark localization by reducing the number of predictions needed to reach the target landmark. Building upon this algorithm, we provide two methods for multi-landmark localization: (1) a multi-task approach with a novel sample efficient training method and (2) an approach for the simultaneous localization of multiple landmarks by using multiple agents inside the environment, which are controlled by a single neural network. The latter, called Simultaneous Acting and Localizing (SAL) furthermore allows learning spatial relations between landmarks during training.

We compare the proposed methods to an existing Deep Q-Network (DQN) based *RL* algorithm and a state-of-the-art *CNN* approach for landmark localization. We show that our algorithm is able to reach comparable performance and, at the same time, reduces computational cost for both single- and multi-landmark localization. Furthermore, we show that *SAL* is able to learn spatial configurations, which allows localizing hidden or occluded landmarks.

Kurzfassung

Die automatische Lokalisierung anatomischer Strukturen ist ein wichtiger Schritt für eine Vielzahl von Anwendungen in der medizinischen Bildverarbeitung. Viele Algorithmen wie Segmentierung oder Bildregistrierung benötigen die zuverlässige Detektierung anatomischer Strukturen als Vorverarbeitungsschritt. Derzeitige Methoden verwenden hauptsächlich Heatmap-Regression auf Basis von Convolutional Neural Networks (CNNs) die zwei bedeutende Nachteile mit sich bringen: Bildbasierte Methoden benötigen große Mengen an *GPU*-Speicher und Patch-basierte Methoden erfordern das zusätzliche Erlernen eines globalen Models.

Diese Arbeit konzentriert sich auf die automatische Lokalisierung anatomischer Strukturen mit Hilfe eines künstlichen Reinforcement Learning (RL)-Agenten, welcher durch Interaktion mit der Umwelt, in unserem Fall 2D-Röntgenbildern, lernt, den optimalen Weg zur Zielstruktur zu finden. *RL*-Agenten sind in der Lage, sich durch die Umwelt zu bewegen, indem sie lokale Bildausschnitte beobachten und eine Repräsentation der Anatomie erlernen. Wir stellen einen Algorithmus für die Lokalisierung einzelner Strukturen auf der Basis von Deep Deterministic Policy Gradients (DDPG) vor. Mit dem *DDPG*-Algorithmus sind wir in der Lage, im Vergleich zu bestehenden *RL*-basierten Algorithmen Rechenressourcen einzusparen, indem wir die Anzahl der Schritte, die zum Erreichen der Zielstruktur erforderlich sind, reduzieren. Aufbauend auf diesem Algorithmus entwickeln wir zwei Methoden für die Lokalisierung mehrerer anatomischer Strukturen in einem Röntgenbild. Wir verwenden einen Multitasking-Ansatz mit einer neuartigen, effizienten Trainingsmethode, die es dem *RL*-agenten ermöglicht gleichzeitig die optimalen Pfade zu verschiedenen Zielstrukturen zu erlernen. Weiters, für die gleichzeitige Lokalisierung mehrerer Strukturen verwenden wir mehrere Agenten, die von einem gemeinsamen Neuronalen Netzwerk gesteuert werden. Diese Methode ermöglicht uns außerdem, eine räumliche Konfiguration anatomischer Strukturen zu erlernen, was uns in der praktischen Anwendung des erlernten Algorithmus ermöglicht, verdeckte anatomische Strukturen zu lokalisieren.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First and foremost I would like to thank Dr. Darko Štern for giving me the opportunity to do my master thesis in this group and for the guidance, the constructive feedback and the review of this thesis. I would also like to thank the members of the research group, especially DI Christian Payer and DI Franz Thaler for constant support and helpful suggestions throughout this work. Lastly, I thank my family and friends for their moral support during my whole studies.

Contents

List of Figures

List of Tables

1 Introduction

Contents

Medical imaging is a field of research that deals with creating a visual representation of the interior of the body as well as the visualization of physiological processes to aid the physician in diagnosis and surgery. Structures of interest are recorded with different imaging techniques, depending on the desired acquisition time, contrast, spatial resolution and other criteria. The most common medical imaging methods are *Computed Tomography (CT)*, *Magnetic Resonance Imaging (MRI)*, *Ultrasound Imaging (USI)* and radiography. With the help of these images, the physician is able to detect anomalies and diseases and is able to plan surgical interventions in a more sophisticated way. However, analyzing medical images can be tedious and complex. To support medical professionals and to reduce their workload, the field of Medical Image Processing (MIP) has gained research interest in the last decades.

MIP has strong parallels to the field of computer vision and deals with similar tasks. *MIP* includes basic operations such as denoising and sharpening, but also more advanced problems such as segmentation or image registration. While the algorithms used in *MIP* share a lot of theory with the computer vision domain, the problems in medical image processing pose additional challenges. Data-driven methods are limited by low availability of proper training data, due to privacy concerns and high cost of labeling through expert annotators. Often datasets include systematic error due to negligence caused by repetitiveness. While large computer vision datasets usually have sizes in the order of millions, a large medical image dataset has a size of a few thousand images, depending on the task. Another challenge arises with the use of volumetric data since medical images

are often 3-dimensional (3D), which increases computational and algorithmic complexity. Also, medical images can be acquired with a myriad of different techniques, each coming with its own characteristics. Finally, algorithms are usually tailored to a specific task and include a priori knowledge, which needs strong collaboration between medical experts and engineers.

1.1 Landmark Localization

Landmark localization, especially facial landmark localization is a well-studied problem, due to its rising importance in commercial applications for mobile devices. Landmark localization is the fundamental building block of many computer vision applications such as face recognition, facial expression recognition, eye gaze tracking, and many more. Landmarks are points of interest that occur at a specific structure within images. Examples of facial landmarks are the corners of the mouth or the eyes. Landmarks can be further specialized to medical applications, and consequently are called medical or anatomical landmarks. Anatomical landmarks are prominent features at structures of interest inside or on the surface of the human body and include among others, joints, teeth or commissures. Medical image analysis algorithms are often based on the reliable detection of such anatomical landmarks. An example is image registration, where two images are aligned and transformed, i.e. *registered* to each other. For landmark-based image registration algorithms, it is important, that the landmarks are always located at the exact same anatomical position. Furthermore, many structures of interest contain multiple landmarks, which can be used for diagnosis. An example is cephalometric image analysis, which deals with the measurement of the skull or age assessment, where often ossification is used as an indicator, which is essentially expressed by the relative size between different bones.

Manual landmark localization is time-consuming and suffers from high observer variability. Therefore, automatic landmark localization deals with developing computer algorithms that are able to localize landmarks without or with minimal human interaction. Developing such algorithms is challenging for the following reasons. The great variety in image intensities, contrast, image quality and other properties make it challenging for algorithms to generalize. Also the structure of interest's shape, size and orientation can have a great degree of variability. Data-driven methods suffer from low availability and expensive annotation of training data. Furthermore, training and inference on 3D images is computationally expensive, making real-time applications challenging. Finally, if multiple landmarks are to be localized, spatial configuration between the landmarks and ambiguity should be considered.

Current state-of-the-art methods are mainly based on Convolutional Neural Networks (CNNs). *CNN* based methods can be divided into two categories. The first category processes the image as a whole, which puts high demands on the hardware, especially when working with 3D images. The second category uses local image patches as training samples to reduce inference complexity. These methods however need additional global

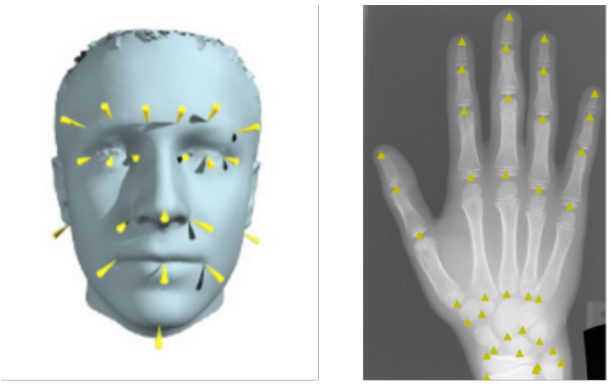


Figure 1.1: Examples of facial landmarks (left) and joint landmarks of the human hand (right). Left image borrowed from [40].

guidance, which in many cases requires learning an additional model.

1.2 Reinforcement Learning for Landmark Localization

Reinforcement Learning (RL) is a machine learning paradigm that originated from the psychology of animal learning. In the late 80s, the theory of *RL* was formulated combining elements from psychology, control theory and dynamic programming. Although initially mainly theoretical in nature, *RL* started to make its way into real-world applications since the advent of more efficient algorithms in recent years, especially in the field of robotics, self-driving vehicles and autonomous Unmanned Aerial Vehicles (UAVs). Since the ground-breaking publication in 2015 [56], where for the first time deep neural networks and *RL* have been combined successfully, the research field has gained a lot of attention.

In *RL* a goal-oriented software agent takes actions in an environment and learns by trial and error, based on rewards or punishments. These rewards or punishments are controlled by a task-specific reward function, which is the only supervision an *RL* algorithm needs. In this thesis, we adopt an *RL* strategy to the problem of landmark localization. The *RL* agent navigates through the environment, the medical image, and decides at each time step in which direction to move, by observing a local image patch around its current position. The *RL* agent aims to learn the optimal path to the target landmark. To exploit the representative power of *CNNs*, we use a *CNN* to process image patches and map to the agent's actions. The combination of *CNNs* and *RL* is known as Deep Reinforcement Learning (DRL).

Using *RL* has the advantage, that the agent is able to internally keep a representation of the anatomy, while benefiting in terms of computational requirements from processing image patches. This internal model of the anatomy allows the *RL* agent to localize the landmark from any arbitrary starting position inside the image, without the need of an additional model for global guidance. By learning from patches *RL* eliminates the need of

storing the whole image in *GPU* memory, which is a challenge for *CNN*-based methods, especially when working with large 3D volumes. Furthermore, patch-based methods require less training images, because by splitting the images into smaller patches artificially generates a large amount of training data. Moreover, the process of localizing anatomical landmarks based on the perception of local image information is similar to the way a physician localizes anatomical structures in a medical image. Indeed, the physician, based on his learned knowledge, can estimate the position of an anatomical structure relative to another structure in the image. The closer the initial observation was, the more accurately the physician is able to estimate the sought landmark.

1.3 Contributions

RL for anatomical landmark localization was first introduced by Ghesu et al. [33], where a Deep Q-Network (DQN) agent was used as a foundation. Based on this publication, we implement a baseline approach for single landmark localization and tackle specific problems that arise with this method. Furthermore, we develop methods for multi-landmark localization. Our research led to the following contributions:

1. We present a multi-task strategy based on the *DQN* algorithm, where a single agent is trained for multiple landmarks, which allows training multiple landmarks simultaneously. This method allows training an agent for multiple landmarks in almost the same time needed for a single landmark.

2. We implement a method for single landmark localization, based on the algorithm Deep Deterministic Policy Gradients (DDPG). This algorithm allows predicting a variable step-size, which makes the agent more versatile. This allows reducing the number of steps the agent needs to localize the target landmark, which leads to reduced inference time while keeping the same accuracy.

3. We also extend the *DDPG* method for single-landmark localization to the multi-landmark case. Similar to the previous item, we use a multi-task strategy and present a sample-efficient training method for this algorithm.

4. We present an algorithm inspired from Multi-Agent Reinforcement Learning (MARL), where multiple agents are used to localize multiple landmarks simultaneously. This method allows learning spatial configurations between landmarks. We show that we are able to localize missing or occluded landmarks during inference with this method.

2 Related Work

Contents

This chapter provides a literature review of the topics covered in this thesis. The first part of the chapter will summarize the state-of-the-art in landmark localization, including general methods from the computer vision domain and more specific methods from the medical image processing domain. The second part of the chapter will provide an overview to recent applications of reinforcement learning to the field of medical image processing.

2.1 Landmark Localization in Computer Vision

The earliest methods for landmark localization used handcrafted low-level descriptors of the landmark to automate the process of feature extraction. These descriptors often made use of mathematical operators such as image gradients, geometrical descriptions like curvature, line-crossings or saddle points or used the intensities of the images directly [65]. Feature matching algorithms could then be applied to match the descriptor with the image. These methods however were very specific and could not be easily generalized to other datasets.

Statistical Shape Models (SSMs) were introduced, where the descriptors for the features in the dataset were built by automated statistical analysis of the dataset. The first method to use shape models is called Active Shape Model (ASM) and was introduced by Cootes and Taylor [18]. *ASMs* generate a model of the dataset by computing the mean landmark positions of the labeled data and applying Principal Component Analysis (PCA) to the data, which is represented by the landmarks. The eigenvectors generated by *PCA*

are called the modes and are added to the mean landmark positions weighted with a factor w. To predict landmarks, these weights are altered, by minimizing a certain energy such that the control points fit the shape in the new image. Cootes et al. [19] further refined the method by including a multi-resolution framework which increases robustness if the initial guess was far off the true position of the landmarks.

An improvement of the *ASM* is the *Active Appearance Model (AAM)* [15], developed by the same research group, which includes textural information to the shape model. *AAMs* were mainly developed for face modeling which in turn were used for face recognition [27] or to interpret face images [28]. A comparison including applications for both *ASMs* and *AAMs* is given in [16].

Similar to *AAMs*, *Constrained Local Models (CLMs)* [22] use texture information to match an instance of the shape model to the image. While *AAMs* generate a texture model of the complete image by *PCA*, *CLMs* only use image patches centered around the landmarks.

Many modern approaches are based on more advanced machine learning methods such as *Random Forests (RFs)* [6, 35] and *Convolutional Neural Networks (CNNs)* [44]. Random forests are an ensemble method based on multiple decision trees, which are trained independently on random samples of the data. A decision tree consists of leaf nodes and split nodes, where each split node contains a weak learner. Random forests can be classified into classification or regression forests, whereby in landmark localization regression forests dominate. In a regression forest [20] each leaf node stores a distribution over a continuous variable. The final prediction is the union over the distributions from the predictions of the individual trees that build up the regression forest. A common method to use regression forests for landmark localization is called the voting approach. In the voting approach a feature vector of an image patch around each pixel in the image is generated (i.e. using Haar-like features) and evaluated by the regression forest. The regression forest casts a vote for a displacement relative to it, where the landmark is assumed. These votes are accumulated and the region with the most votes is chosen as the landmarks position. Fanelli et al. [30] use the voting approach to localize facial features for head-pose estimation. Lindner et. al. [49] propose an *RF* regression-voting model for landmark localization, where votes are accumulated on a grid.

CNN based approaches for landmark localization have been investigated extensively since the ImageNet classification breakthrough in 2012 [42], especially for facial landmark localization and human pose estimation. The authors of [79] were the first, to introduce *CNNs* to the field of pose estimation, by regressing the coordinates of joints, a problem known as coordinate regression. Other methods use *CNNs* to regress heatmaps with the same size as the original input [57, 61, 74, 77, 78, 86]. The heatmap approach has been shown to be more robust and is used in general. Heatmap regressors also have the advantage over coordinate regressors, that through the use of encoder-decoder structures such as U-Net [67] no densely connected layers are needed, which are prone to overfitting and usually introduce a lot of additional parameters, which make training more expensive. The

authors of [13] use an approach, where they first generate candidate key points and then refine the prediction iteratively by predicting a correction vector for the key points. The problem of human pose estimation is even more challenging if the pose of multiple persons inside a single image has to be estimated. The authors of [11, 12, 37, 62] use a bottom-up approach, where first the key points of every person inside the image are localized, followed by assigning the individual key points to distinct individuals. The authors of [31] use a top-down approach, where first the bounding boxes of the individuals are detected, followed by single-person pose estimation using a stacked hourglass network [57]. Similar to the problem of pose estimation, recent facial landmark localization methods make heavy use of *CNNs*. The authors of [51, 63, 75, 90] address the problem using coordinate regression. Heatmap regression is used in the publications [7, 25, 46].

2.2 Landmark Localization in Medical Image Processing

The authors in [64] describe corner detection methods, based on image gradients and Hessian matrices to locate landmarks in human brain images from Magnetic Resonance (MR) and Computed Tomography (CT) volumes. The same differential methods for 3D corner detection as in [64] are used in [66] for landmark localization as a preliminary step for registration of medical images. Since these corner detectors are based on gradients i.e. high variations in the image, these algorithms lead to a lot of false-positive detections. An advantage of these algorithms is, that they do not involve optimization, which makes them very fast compared to more advanced algorithms. The authors of [5] use a template-based approach to detect landmarks for lung- and nodule registration. Štern et al. [73] propose a edge detection based method for 3D spinal centerline detection, where the centerline is used in a second step to localize the center of the vertebral bodies by analyzing the image intensities and gradient magnitudes along the centerline.

Some approaches in Medical Image Processing (MIP) use atlas-based registration methods [29]. An atlas is a single image or a set of images, which represent a dataset. These methods use the landmarks of the atlas and apply a transformation to the new image to match (i.e. register) these landmarks. Due to variation in the images, the registered image usually needs to undergo a non-rigid transformation, which is not a trivial task. The opposite problem is the use of deformable models, where a model (e.g. the atlas) is registered to the image and is used in [32].

The *ASM* is used by Cootes et al. in [17], where they apply their method from [18] to medical images, by fitting the *ASM* to the images of organs. Another application of the *ASM* is cephalometric analysis [36], which deals with the measurement of the head, by localizing cephalometric landmarks. De Bruijne et al. [24] use the *ASM* for segmentation of tubular structures in medical images. As the previous examples show, the *ASM* in medical applications is used mainly for segmentation, where the landmarks only act as control points for the segmentation boundary and are often a disregarded byproduct. The same applies to the *AAM*, which is used for cardiac segmentation of *MR* and Ultrasound

(US) images in [54]. A more detailed overview of *AAMs* in Medical Image Analysis (MIA) is given in [4]. The *CLM* is used in [22] to localize points of interest in *MR* images of the brain and in dental panoramic radiographs.

A stronger impact on medical landmark localization had recent developments in *RFs* and *CNNs*. Criminisi et al. [21] use regression forests to estimate location and size of organs in 3D *CT* images, where the coordinates describing location and size are predicted as continuous variables. Cheng et al. [14] use a discriminative random forest approach to locate the Dent-landmark, which is one of the key landmarks to construct the midsagittal reference plane. Ebner et al. [26] use a weighted regression voting approach, which comprises two random forests. The first forest is used for coarse global landmark localization, and the second to refine the landmark predictions locally. Urschler et al. [81] extend the *RF* regression approach by first generating candidate regions and then iteratively refining the joint landmark predictions of multiple landmarks using the coordinate descent algorithm [88]. In the refinement stage, they integrate the geometric configuration of multiple landmarks into the *RF* optimization problem.

The use of *CNNs* has seen a lot of attention in recent years, due to its success in computer vision tasks. Payer et al. [60] propose a regressing heatmap approach, with a special architecture that takes into account the spatial configuration between multiple landmarks. The application of *CNNs* to 3D volumes is particularly challenging due to increased computational complexity. Different approaches have been investigated to tackle this problem. The method of representing 3D volumes with 2D planes is commonly known as the 2.5D representation. Roth et al. [68] use such a 2.5D approach, where they sample n viewing planes of a volume for the detection of lymph nodes. Despite increased efficiency, these representations omit a lot of relevant information. Zheng et al. [91] tackle the 3D landmark detection problem as a two-stage classification problem, where first a shallow network generates multiple landmark candidates using a sliding window approach. A second deep network classifies image patches around the proposed landmark positions. To increase *CNN* efficiency, they additionally decompose 3D convolutions into three 1D convolutions. The authors of [58] propose a patch method where a *CNN* is used to simultaneously regress a displacement vector from each patch to the desired landmark and to classify if the patch contains the landmark. Yang et al. [89] use *CNNs* for classification of the individual slices of a 3D image. The three orthogonal slices with the best classification result have then been intersected to obtain the location of the landmark.

2.3 Reinforcement Learning in Medical Image Processing

In 2015, with the introduction of Deep Q-Network (DQN) [56] the technology Reinforcement Learning (RL) had a major breakthrough and has since been used successfully in many areas of computer vision and robotics. While other machine learning methods have been used extensively in *MIP*, *RL* is a fairly unexplored research field. Also, some attempts have successfully introduced *RL* to the *MIP* domain, which will be introduced in

this section. The theory of *RL* is explained in more detail in Chapter 3.

The problem of landmark localization was first formulated as a reinforcement learning problem by Ghesu et al. in [33]. In this approach, a *DQN* agent is used, which observes a sub-image or a sub-volume for 2D images and 3D volumes respectively. The agent moves with a fixed step size on the four principal directions through the complete image or on the six principal directions through the volume to localize the landmark. Based on this work, Alansary et al. [3] evaluated different improvements over the *DQN* algorithm, such as Double DQN (DDQN) [82] or Dueling *DQN* [84]. In [83] the authors extend the idea of using *DQN* for landmark localization to the multi-landmark case. They formulate the problem as a multi-agent problem, where weight sharing between convolutional layers is supposed to establish communication between agents. During the work in this thesis, a further extension to this method has been published, where additional to weight sharing in the convolution layers, explicit communication signals are learned by sharing communication channels in the fully connected layers [45].

Besides landmark localization, *RL* has been investigated for several other problems in *MIP*. Maicas et al. [52] use *RL* for breast lesion detection. They define the agent as a subvolume in 3D DCE-MRI volumes where additionally to the discrete translational actions, as they are used in landmark localization, they add actions for scaling and a trigger action to signalize detection of the desired area. Sahba et al. [69] use an *RL* agent for prostate segmentation. They divide the ultrasound images into smaller sub-images and use thresholding on the sub-images to do the segmentation. The agent's actions are to lower or raise the segmentation threshold. The authors of [47] use an artificial agent for rigid image registration. The agent's observation is raw image data and its goal is to find the best sequence of motion actions that lead to image alignment. Following a similar approach, the authors of [41] apply *RL* to non-rigid registration. In [2] automated view planning is performed by applying a *DQN* agent to cardiac- and brain Magnetic Resonance Imaging (MRI) images. The algorithm is assessed on the midsagittal and anterior-posterior commissure planes of the brain images and the 4-chamber long-axis plane in the cardiac images.

3 Background

Contents

3.1 Reinforcement Learning

This chapter summarizes the theory of Reinforcement Learning (RL). The first section of the chapter presents the fundamentals of *RL*, the second section focuses on Deep Reinforcement Learning (DRL) and the third section provides a short introduction to multi-agent *RL*. This whole chapter is based on the well-known book *Reinforcement Learning - An Introduction* from *Sutton and Barto* [76] and on the lecture of David Silver [72], the *RL* research group leader of the company DeepMind Technologies.

3.1.1 Elements of Reinforcement Learning

This section describes the basic elements of *RL*. The terms explained in this section will appear extensively in upcoming sections and are essential for a profound understanding of the topic.

- **Agent**

  The agent is the most important element in the *RL* setup. It is the component that makes the decision of what action to take. To infer the right actions in certain states, the agent learns by reward or punishment. An agent can be a physical object like a self-driving vehicle or a more abstract construct like a software-based controller that plays Atari games.

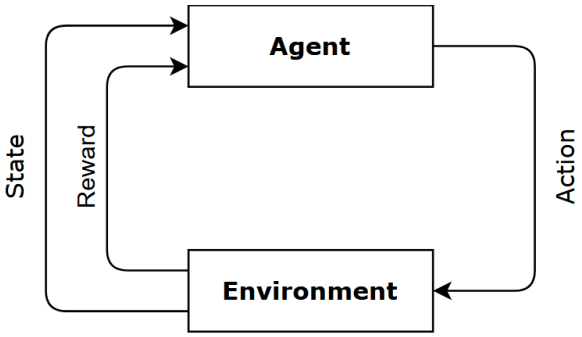


Figure 3.1: The agent-environment loop represents the basic idea behind *RL*. The agent takes an action, transitions to a new state and receives a reward. With this information, the agent learns to infer the right actions from states.

- **Environment**

  The environment is the playground of the agent and includes two basic functions. First, it needs to provide a set of states, which can be visited by the agent. The second necessary property of the environment is a reward function, which can be thought of like another entity inside the environment, which observes the agent and tells him, whether his actions are good or bad. The reward function can be represented in multiple ways. For example in a robot control problem, the reward can be defined by a person. In other scenarios, a mathematical function may suit the problem better.

  In abstract words, one could describe the environment as a stochastic finite state machine with inputs (actions sent from the agent) and outputs (observations and rewards sent to the agent).

- **Action**

  Actions are the agent's way of interacting with the environment and thus transfer between states. The set of actions available to an agent is called the action space, which can be either discrete or continuous. An example of a discrete action space is a robot with the possibility to move in its four principal directions: *left*, *right*, *forward* and *backward*. A continuous action space on the other hand could be an agent with a steering gearbox, where the steering angle can be any real-valued angle inside a certain interval.

- **State and Observation**

  A state is the situation in the environment the agent is in. The way the agent is able to observe its state in the environment is an important topic in *RL* and deals with the correct description of the situation. How a state is represented is called an observation. Proper state representation is important to uniquely describe the agent's position in the environment. A robot for example, additionally to a camera,

might also require acceleration sensors to describe its situation. For an agent playing Atari games instead the raw pixel data should be enough.

- **Reward**

  A reward is given to the agent after every action it executes and is usually represented by a scalar value. The reward can be positive if the agent's action is considered good in a particular state, negative otherwise or zero if it has no particular influence. The agent's sole objective is to maximize the total reward it receives in the long run. The reward however does not indicate if the agent is moving towards a more favorable state since it might be advantageous to sacrifice immediate rewards for larger future rewards. How favorable a state is, is represented by the value function, which is an important concept in *RL* and is described in more detail in Section 3.1.3. The amount of reward an agent receives for taking an action in a specific state is computed by a problem-specific *reward function*.

- **Episode**

  An episode describes one full simulation of the agent-environment loop, at the end of which the agent ends in a terminal state. A terminal state is reached if the agent accomplishes its designated goal. A task is called *episodic* if there exists a terminal state. Otherwise, it is a *continuing* task, which is comparable to a control problem, where the agent keeps making decisions until it is turned off.

- **Policy**

  A policy is the strategy an agent follows and can be described as a probability distribution over actions given states. That is the likelihood of every action when an agent is in a particular state. In the simplest case, the policy is a function, that takes a state as input and returns an action. In this case, the policy π is deterministic and can be described as a map from a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$:

  $$\pi : \mathcal{S} \to \mathcal{A} \tag{3.1}$$

  In other cases, the policy might be stochastic, which means, the policy can be described as a map from a state-action pair to a probability as in the following equation:

  $$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1] \tag{3.2}$$

- **Model**

  A model is a representation of the environment's underlying Markov Decision Process (MDP). During training, the agent tries to build an explicit model of the *MDP*, which can later be used for *planning*, which is the process of decision-making in advance, i.e. predicting a set of actions, which should lead to the desired goal. The model is

optional in *RL*. In model-free *RL* the agent only decides based on the current state and makes a new decision after it transitioned to the next state. The majority of real-world *RL* problems, including the landmark localization problem, which will be explained in the next chapter are solved model-free.

3.1.2 Markov Decision Processes

In general, the *RL* problem is formulated as an *MDP*, which can be described completely as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_a, \mathcal{R}_a \rangle$, where:

- $\mathcal{S}$ is a set of states called the *state-space* and $s_t \in \mathcal{S}$ describes an individual state at time step t,
- $\mathcal{A}$ is a set of actions called the *action-space*and $a_t \in \mathcal{A}$ describes an individual action executed at time step t,
- $\mathcal{P}_a(s, s')$ is the probability, that action a leads to state s' from state s and
- $\mathcal{R}_a(s, s')$ is the reward for transitioning from state s to s' after taking action a.

The *MDP* is an extension to the Markov Reward Process (MRP), which itself is an extension to the Markov chain. The Markov chain models an environment by assigning transition probabilities from each possible state to their subsequent states (Figure 3.2). The *MRP* adds a reward to the transition (Figure 3.3) and the *MDP* further adds the possibility of decision-making to the *MRP* (Figure 3.4). *Decision-making* in this context means, that a policy is introduced to the process, by which an agent decides which action to take in a certain state. The policy is modeled with a probability distribution over actions given a state.

$$\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a | \mathcal{S}_t = s] \tag{3.3}$$

A deterministic policy is a policy, where the agent would always execute the same action in the same state, which means, that the probability is always 1 for a single action and 0 for all others. Additionally to the policy also the environment can be stochastic, which means, even though the agent takes a specific action does not ensure that the agent will transition to the desired state. A frequent example of a stochastic environment in real-world application is a self-driving vehicle, where the properties of the underground (mainly friction) provide a source of uncertainty. The example *MDP* in Figure 3.4 includes both stochastic policy and stochastic environment. The black dot shows a case, where the action can lead to multiple states.

The *MDP* satisfies the Markov assumption. The Markov assumption assumes independence of past and future states. This means, that the state and the behavior of the environment at time step t are not influenced by past agent-environment interactions. This assumption is formulated in Equation 3.4.

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t] \tag{3.4}$$

If the agent does not receive all the information required to uniquely describe its state, then the *MDP* becomes a Partially Observable Markov Decision Process (POMDP) [38], which is a special research field in *RL*.

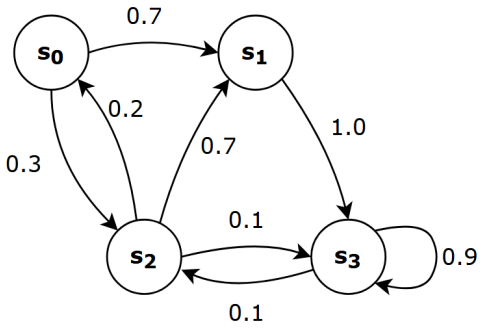


Figure 3.2: A Markov chain with transition probabilities.

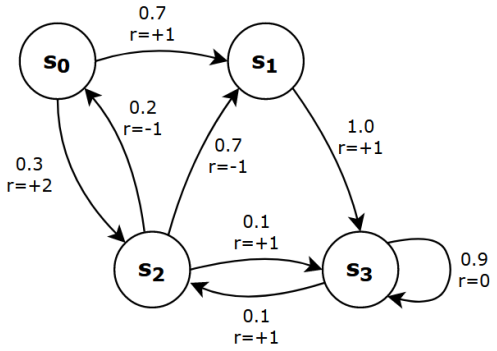


Figure 3.3: A Markov reward process, which extends the Markov chain by adding rewards to the transitions.

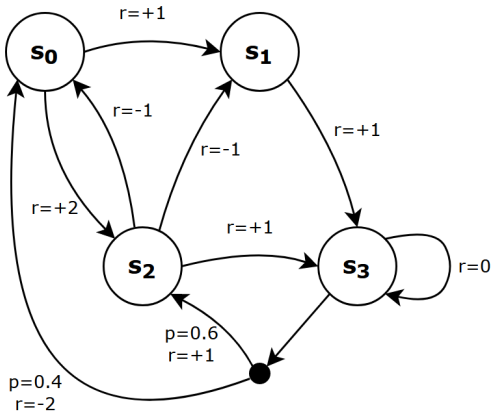


Figure 3.4: A Markov decision process. In an MDP the agent has the power over which actions to take and therefore is able to maximize the reward by taking the optimal sequence of actions. However, the environment can still be stochastic, as can be seen in state s_3. If the agent decides to take the action to the black dot, it has a 60% chance to land in state s_2 and a 40% chance to land in state s_0.

3.1.3 The Value Function

Model-free algorithms can be separated into policy-based and value-based algorithms. These two classes do not have to be strictly separated, which will be shown in Section 3.2.3.1. In policy-based methods, an explicit representation of the policy is built, which is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ from a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$. In value-based methods, no

explicit policy is stored, but a value function (or state-value function). A value function is a function, that estimates how good it is for an agent to be in a particular state and is usually denoted as $V(s)$. The policy is here implicit and can be derived from the value function. The value $V_\pi(s)$ is the expected cumulative reward R_t starting from state $s_t = s$ at time t, following a policy π:

$$V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} r_{t+k+1}|s_t = s] \tag{3.5}$$

The immediate reward, i.e. the reward received for a transition between two states is denoted as r and the cumulative reward, i.e. the sum of all rewards along the agent's trajectory is denoted as R.

In non-terminating episodes this series would sum up to infinity, therefore a *discount factor* is introduced to stabilize the training process. The discount factor is an important hyper-parameter and essential for convergence of most *RL* algorithms. To appreciate the importance of the discount factor, often the analogy to finance is brought up, where immediate returns to an investment are more important than future returns, due to inflation and uncertainty in the markets.

$$V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s] \qquad \gamma \in [0, 1] \tag{3.6}$$

Equation 3.6 can be written recursively. The recursive form is called the *Bellman equation for* V_π.

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_t|s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1}|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{s,s'}[\mathcal{R}^a_{s,s'} + \gamma \mathbb{E}_\pi[R_{t+1}|s_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{s,s'}[\mathcal{R}^a_{s,s'} + \gamma V_\pi(s')] \end{aligned} \tag{3.7}$$

$\mathcal{P}^a_{s,s'}$ describes the probability of transitioning to state s' from state s after taking action a and $\mathcal{R}^a_{s,s'}$ denotes the associated reward.

The **optimal value function** V^* is achieved if it has a higher value than any other value function for every state s.

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S}, \tag{3.8}$$

where $\mathcal{S}$ is the space of all states and π is the policy, i.e. the function that maps a state to an action.

3.1.4 The Q-Function

Closely related to the value function is the Q-function or *action-value* function. The Q-value can be understood as a quality measure of taking a certain action in a certain state to gain future rewards, hence the name "Q"-value and answers the question, which action gives the most future reward given the current state. Like the value-function, the Q-function predicts the expected discounted reward for a given state, but for every possible action. Assume an agent with a set of four actions. The value function would consist of a single value per state and the Q-function would consist of four values per state.

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_t|s_t = s, a_t = a] \\ &= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \end{aligned} \tag{3.9}$$

For finite and small state spaces, the Q-function can be represented by a table, which can be used as a look-up table during inference. Since the Q-function predicts the expected reward for taking a certain action given a state, the optimal policy in Q-learning is the policy that maximizes the Q-function for every state. In other words, the optimal policy is obtained by choosing the action with the largest Q-value in every state. Table 3.1 shows a very simple example of a Q-table. Although the difference between value-function and Q-function seems insignificant, the slight modification, which includes the agent's action makes it possible to implicitly derive the policy from the Q-function. A model-free prediction would not be possible from the value function alone.

The Q-table can be computed with dynamic programming methods by arbitrarily initializing all elements of the table (a Q-table is usually initialized with zeros). During training, every Q-value is updated sequentially by the methods explained in the next section. Convergence is achieved if the Q-values do not change anymore. If training has been successful, the Q-table can be used as a look-up table. Consider the example in Table 3.1. If the agent is located in state four, the maximum Q-value of $Q = 1.0$ corresponds to the action "right". This action can be considered optimal if the Q-function is optimal. Taking the action with the largest Q-value results in a deterministic policy, which means, the agent would always choose the same action in the same state. In this case, the connection between Q-function and value function is as follows:

$$V_\pi(s) = \max_a Q_\pi(s,a) \tag{3.10}$$

Equivalent to Equation 3.7, the Q-function can be written recursively, which is called the *Bellman equation for the Q-function.*

s \ a	↑	↓	→	←
0	0.2	-0.5	-0.21	-0.6
1	0.3	-0.4	0.4	-0.1
2	1.0	-0.2	-0.3	-0.1
3	-0.22	-0.04	0.5	-0.31
4	-0.3	-0.02	1.0	0.0
5	-0.6	0.15	0.21	0.05

Table 3.1: Example of a Q-table. The corresponding environment has six states and four actions - up, down, right and left. The state is indicated as s and the action as a. Assuming the Q-table to be optimal, the optimal policy is choosing the action with the largest Q-value for a given action. As an example consider the agent to be in state 4. The maximum Q-value is 1.0 and the corresponding action is "right".

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[R_t|s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1}|s_t = s, a_t = a] \\ &= \sum_{s'} \mathcal{P}^a_{s,s'}[\mathcal{R}^a_{s,s'} + \gamma \mathbb{E}_\pi[R_{t+1}|s_{t+1} = s']] \\ &= \sum_{s'} \mathcal{P}^a_{s,s'}[\mathcal{R}^a_{s,s'} + \gamma V_\pi(s')] \end{aligned} \tag{3.11}$$

By Equation 3.10, we can rewrite Equation 3.11 as

$$Q_\pi(s,a) = \sum_{s'} \mathcal{P}^a_{s,s'}[\mathcal{R}^a_{s,s'} + \gamma \max_a Q_\pi(s',a)] \tag{3.12}$$

3.1.5 Learning Methods

The goal of *RL* is to solve an *MDP* by taking samples from the environment. Solving an *MDP* means determining the value function or the Q-function of the *MDP* that satisfies the Bellman equation. This section introduces the most important techniques to solve an *RL* problem for the value function or the Q-function.

Monte Carlo Learning

Monte Carlo (MC) methods are a broad class of algorithms, which rely on random sampling to obtain numerical results. They are often used for physical simulations, where analytical solutions are infeasible. In *RL*, *MC* methods sample an entire trajectory and update the policy only when the agent reaches its terminal state. In other words, the agent has to play an entire episode before it can update his policy. The update rule for *MC* methods is

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)). \tag{3.13}$$

As defined in Equation 3.5, R_t denotes the cumulative reward for the entire trajectory starting at state s_t.

Temporal Difference (TD) Learning

A disadvantage of *MC* methods is, that they are not suitable for online training. Also in many problems, Monte Carlo methods are infeasible, for example in episodic environments, where the agent does not necessarily terminate. An example is the landmark localization problem: Assuming the agent is inexperienced and therefore only takes random actions, the episode only terminates if the agent coincidentally lands on one specific pixel in an environment of e.g. $512 \times 340 = 174.080$ pixels, which is rather unlikely. Therefore, Temporal Difference (TD) methods have been introduced, which determine the value function by taking only samples from single transitions, consisting of the elements *state*, *action*, *reward* and *next state*. These four elements are written as a tuple $\langle s, a, r, s' \rangle$, which is called the *experience tuple*. *TD*-methods typically make use of the Bellman equation (Equation 3.7), since the Bellman equation is only dependent on the reward and on the state-value of the next state. The training procedure starts by initializing the value function to arbitrary values (e.g. all zeros) followed by updating each value individually until the values stop changing. Given enough training time, *TD*-sampling converges to the optimal value function. Intuitively one can think, that due to the reward in every transition, a small piece of information is added to the value function. The process of sampling only experience tuples is called *bootstrapping*. The same process can be applied to determine the Q-function with the Bellman equation for the Q-function (Equation 3.11). Experiments showed that *TD*-methods converge more quickly than Monte Carlo methods because *MC* methods have to deal with much higher variance in the samples. The most basic algorithm of the *TD* class is called SARSA, short for *state, action, reward, next state, next action*. SARSA will be explained in Section 3.1.7.

3.1.6 Exploration and Exploitation

The exploration-exploitation dilemma is a problem that arises with learning from trial and error where an agent has to repeatedly make a decision with uncertain pay-off. The agent's goal is to maximize utility. The question the agent faces is, should it choose the best actions to his current knowledge (exploit) or should it try different actions that might lead to even greater utility (explore)? There has been extensive research concerning this question that brought up many strategies. The simple yet effective method ϵ-greedy is explained in the next section. Even though other methods such as *Upper Confidence Bound* or *Boltzmann Exploration* have been shown to be slightly more effective, the ϵ-greedy strategy is the most widely used one because of its simplicity.

ϵ-greedy

The ϵ-greedy approach introduces some exploration by choosing random actions with probability ϵ and exploiting actions with probability $1 - \epsilon$. Usually during training ϵ is set to 1 and is decreased periodically until a minimum threshold (e.g. 0.1) is reached. In *RL* the term ϵ-greedy policy is used, whereby the policy usually is only used for training and in some rare cases during inference to prevent overfitting. The purely greedy approach would be to exploit the current knowledge without ever making new decisions and is usually used during inference.

3.1.7 Algorithms

SARSA

SARSA is an online algorithm to determine the Q-function. SARSA [76] is short for *state, action, reward, next state, next action*. Being an online algorithm means, that the agent during training always selects actions based on his current knowledge, which does not allow the agent to explore new actions. The SARSA algorithm works as follows. The agent is located in state s. Following the current policy, it takes action a. The environment transitions the agent to state s' and returns reward r. The agent, again based on the current policy takes action a'. Now enough information is available to perform an update for $Q(s, a)$. This makes the algorithm *on-policy*, which means, that the policy used for bootstrapping is updated continuously. The Q-function is updated in the following way, where α is the learning rate:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma Q(s', a')) \tag{3.14}$$

The term

$$r + \gamma Q(s', a') \tag{3.15}$$

is called the TD-target and the difference between TD-target and the current approximation

$$r + \gamma Q(s', a') - Q(s, a) \tag{3.16}$$

is called the TD-error.

Q-learning

Q-learning [85] is an important algorithm because it builds the basis for many modern algorithms. The algorithm is an improvement over SARSA, because it allows *off-policy* learning. Instead of directly sampling the action from the next state, Q-learning queries the Q-function from the next state for all actions and takes the maximum Q-value for the

TD-target, which is equivalent to the state-value of the next state by Equation 3.10. The pseudocode for the algorithm is shown in Algorithm 1.

Algorithm 1 Q-learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$, arbitrarily and $Q(\text{terminal-state}, \cdot) = 0$
foreach *Episode* **do**
  Initialize s
  foreach *step of episode* **do**
    Choose $a = \pi(s)$ using policy π derived from Q
    Take action a, observe r, s'
    $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
    $s \leftarrow s'$
  end foreach
end foreach

3.2 Deep Reinforcement Learning

Early *RL* algorithms were almost purely theoretical in nature and a lot of the theory has been developed with Q-tables. The success of *RL* algorithms in recent years is owed to the introduction of *neural networks* to the field. The field of *RL* that uses neural networks is called Deep Reinforcement Learning (DRL). In most cases, neural networks are used to approximate the Q-function, the value function or the policy.

3.2.1 Value-Based Methods

Previous chapters focused on the theoretical formulation of the *RL* problem and some basic algorithms. These algorithms are based on the existence of a value-function or a Q-function. The exact representation of these functions is a table, where the value-function has as many entries as there are states in the *RL* problem and the Q-function has as many entries as the state-value function multiplied with the number of actions. This becomes intractable for most *RL* applications since the state-space grows very large or even infinitely large in many real-world applications. Table 3.2 shows the state-space complexity for some board games.

In value-based *DRL* function approximators are used to estimate the Q-function. There are several possible basis functions to approximate the Q-function, like *linear combination of features* or *decision trees*. In practice, however, as in many other problems, *neural networks* are used to approximate the Q-function, because, according to the *Universal Approximation Theorem* [23], a neural network can approximate any function, if it has enough neurons. The Q-function is parameterized by a weight vector θ and can be represented in two ways as shown in Figure 3.5. For continuous action-spaces, the function

Game	State-space complexity
Tic-Tac-Toe	10^3
Backgammon	10^{20}
Chess	10^{47}
Go	10^{170}

Table 3.2: State-space complexity of a number of board games

approximator uses both state and action as an input to predict the Q-value, for discrete actions the approximator uses only the state as input and predicts a Q-value for every possible action.

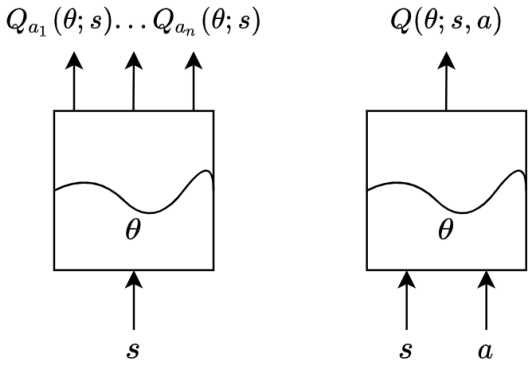


Figure 3.5: In discrete action spaces, the Q-function is approximated as a vector-valued function with only the state as an argument and the Q-value for every action as function output (right image). For continuous action spaces the action is used as a function argument and the Q-function output is a single value (left image).

3.2.1.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) [56] is an algorithm that combines Q-learning with Deep Neural Networks (DNNs) to allow *RL* to work in high-dimensional environments. The name of this algorithm implies, that a *DNN* is used to approximate the Q-function. The algorithm was introduced by DeepMind Technologies in 2015 and posed a major breakthrough in the field of *RL*. Today, it is one of the most widely used algorithms in the field. While most previous algorithms used hand-crafted features, the original *DQN* algorithm used raw sensory image data. Through the use of convolutional filters inside a Convolutional Neural Network (CNN), the network is able to extract the relevant features by itself. Besides using a *CNN*, the algorithm includes two techniques, that essentially led to the success of this algorithm, which are *experience replay* and the introduction of a *target network*. These methods are explained in the upcoming paragraphs. The pseudocode of the algorithm is shown in Algorithm 2.

Algorithm 2 DQN

Initialize empty experience replay buffer D to capacity N
Initialize Q-function Q with random weights θ
Initialize target Q-function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode=1,M* **do**
  Initialize state s_1
  for *t=1,T* **do**
    With probability ϵ select a random action a_t,
    otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and image s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\theta^-; s_{j+1}, a') & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\theta; s_j, a_j))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

Target Network

In (tabular) Q-learning an update only affects the Q-value of a single state-action pair. In *DQN* however, caused by the introduction of a function approximator, the update of a single value also affects all other Q-values and as a consequence also the value of the very next state. This causes network updates to be very unstable since the targets change with every update. To tackle this problem, the *DQN* algorithm uses a target network, which is a frozen copy of the policy network and is updated every C steps. C is an arbitrary hyper-parameter and is usually set to 10,000 as in the original publication. The loss function we are trying to minimize now has the following form:

$$Loss = \mathbb{E}_{s,a,r,s' \sim D}[(r + \gamma \max_{a'} Q(\theta^-; s', a') - Q(\theta; s, a))^2] \tag{3.17}$$

where θ are the parameters of the policy network and the θ^- the currently frozen parameters of the target network. The expectation is used to indicate the use of mini-batches for stochastic gradient descent, which are sampled from the replay buffer D as explained in the next section.

Experience Replay

Experience replay is a biologically inspired method, where samples of past experiences are reused, instead of the most recent ones. The method stems from the idea, that humans

process the information they collect while they sleep and not directly after the information is incurred. In the implementation of experience replay, tuples of information are stored in a so-called replay buffer D of size N in the form $\langle s, a, r, s' \rangle$, where s stands for state, a stands for action, r for reward and s' for next state.

In *DQN* a replay buffer is used to ensure *independent and identically distributed (i.i.d.) data*. During training, a buffer is filled up to a minimum before the Q-network is updated. After this minimum is reached, the Q-network is updated via *stochastic gradient descent* after every training step by sampling a mini-batch of typically 32 to 64 experience tuples.

This is possible because learning phase and experiencing phase are logically independent. However, we interleave the two processes acting and learning by using an ϵ-greedy strategy because improving the policy over time leads to a different behavior of the agent, that should explore actions closer to the optimal actions.

Improvements of DQN

Since the advent of *DQN*, a lot of research has been done concerning this algorithm. The most important improvements Double DQN (DDQN), Dueling *DQN* and Prioritized Experience Replay (PER) are explained in the following chapters. Alansary et. al. [3] however showed, that these improvements to the *RL* problem, did not show particular improvements to the problem of medical landmark detection. Therefore, we did not include these methods in our experiments, but due to the importance of these concepts, they are briefly explained hereinafter.

3.2.1.2 Double DQN

DQN has the problem of overestimating Q-values due to the max function in the TD-target as shown in Equation 3.17. This introduces strong positive bias, also called the *maximization bias*. To tackle this problem, Double DQN (DDQN) [82] was introduced. This method uses two separate networks to compute the *TD*-target. As *DQN* already uses a target network, there is no need to hold additional networks, as was done in early implementations of *DDQN*. Instead, the loss function from Equation 3.17 is modified to Equation 3.18. The online network is used to predict the action for the next state and the target network is used to predict the Q-value of the next state.

$$Loss = \mathbb{E}_{s,a,r,s' \sim D}[(r + \gamma Q(\theta^-; s', \arg\max_{a'} Q(\theta; s', a')) - Q(\theta; s, a))^2] \tag{3.18}$$

3.2.1.3 Dueling DQN

Dueling DQN [84] is mainly a change in the network structure. The Dueling network architecture comprises two separate paths, one of which computes the value function $V(s)$

and the other a so-called *advantage function* $A(s, a)$. At the final layer both value and advantage are combined to the Q-value, as in the following equation.

$$Q(s, a) = V(s) + A(s, a) \tag{3.19}$$

The advantage function is a measure of how much better an action is to the action chosen by the current policy. Equation 3.19 is unidentifiable, therefore a trick is used, where instead of simply adding the two functions, the following forward mapping is implemented, which makes the equation identifiable. This allows to uniquely learn $V(s)$ and $A(s, a)$, without posing special constraints on the loss function.

$$Q(s, a) = V(s) + A(s, a) - \max_{a'} A(s, a') \tag{3.20}$$

This type of structure allows the network to better differentiate actions and to speed up training. With this structure, the value function is updated in every time step and therefore converges much faster. In *DQN*, only one Q-value is updated at a time, which leads to slow convergence in large action spaces. The network structure is shown in Figure 3.6.

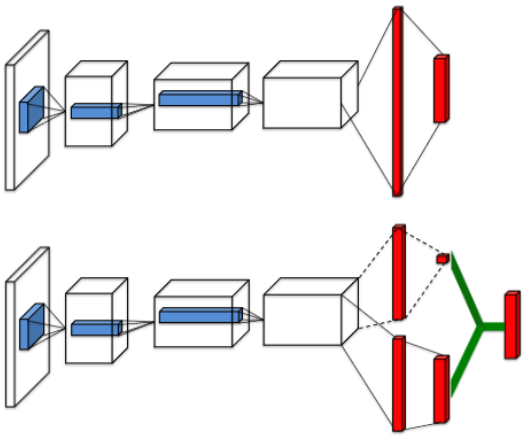


Figure 3.6: A normal Q-network with a single path (top) and a dueling architecture, where state-value and advantage-value are computed in separate paths and recombined at the output layer (bottom). Graphic borrowed from [84].

3.2.1.4 Prioritized Experience Replay

The original *DQN* algorithm stores experience samples in the replay buffer and samples uniformly random to update the Q-network. Prioritized experience replay [70] is based on the idea, that some experience samples include more information than others and should therefore be *prioritized* in the sampling procedure to execute more gradient updates on important samples and fewer updates on less informative samples. The original paper

used the *TD* error as a measure of importance. The *TD* error can be seen as an indicator of how "surprised" the agent was with the outcome of a certain action.

3.2.2 Policy Gradient Methods

Value-function based methods solve the *MDP* by approximating the Q-function, from which a usually *greedy* policy is derived. While it is possible to add some randomness to the policy by e.g. using a ϵ-greedy policy (Section 3.1.6) for inference, it is not possible to build a deliberately stochastic policy as it would break the assumptions on which Q-learning is based, which is Equation 3.10. In some scenarios, however, we want the agent to act with a stochastic policy. Consider the case where we want to teach an agent how to play *rock-paper-scissors*. The optimal policy would be uniformly random. If the agent is value-based, however, due to the deterministic character of value-based methods, the agent would always play the same hand, making it very easy for the opponent to figure out the agent's strategy. Another, more realistic example would be a robot in a room, which gets stuck in a certain position by bumping into the wall. A deterministic policy would enforce the agent to keep on bumping into the wall, whereas a stochastic policy at some point would execute a different action to escape the locked position.

A second major limitation of value-based methods is their application to continuous action spaces. Although possible in theory, it is very difficult to implement, because the Bellmann equation (Equation 3.11), which is part of the loss function, includes the max operator. This means, for every gradient update, we would have to maximize the (now continuous) Q-function, which is an optimization problem in itself and requires more complex methods, which are called *actor-critic* methods.

To solve these two problems, a class of algorithms called *policy gradient methods* are used. Instead of learning the value function, policy gradient methods learn the policy directly. This means, the policy, represented by some function approximator (usually a neural network) directly predicts the probability distribution over the actions or the continuous action instead of learning the underlying value function. A positive side-effect of learning the policy is, that policy-based algorithms can be more stable during training. Since value-based methods are deterministic, a small change in the value function during training can lead to a strongly different policy and can therefore cause oscillations during training. In policy gradient methods instead, a small update in the parameters only affects the policy by a small amount. Equation 3.21 shows the parameterized policy as a probability distribution over the actions.

$$\pi_\theta(s, a) = \mathbb{P}[a|s, \theta] \tag{3.21}$$

As in all other *RL* algorithms, the agent's objective is to maximize the expected reward. The expectation of the reward, shown in Equation 3.22 is called the objective function $J(\theta)$, where θ parameterizes the policy and R_t is the cumulative reward starting at state t.

$$J(\theta) = \mathbb{E}_{\pi}[R_t] \tag{3.22}$$

We are trying to find the parameters θ that maximize the objective function, which can be done using gradient ascent, as in Equation 3.23:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t). \tag{3.23}$$

Policy Gradient Theorem

Consider a class of one-step *MDPs*. The agent starts in state s and terminates after one-time step with reward $r = \mathcal{R}_{s,a}$, where $\mathcal{R}$ is the reward function. Then the objective function 3.22 can be written as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{a \in A} \pi_\theta(s,a) \mathcal{R}_{s,a} \end{aligned} \tag{3.24}$$

and the gradient can be computed as

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{a \in A} \nabla_\theta \pi_\theta(s,a) \mathcal{R}_{s,a} \\ &= \sum_{a \in A} \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \mathcal{R}_{s,a} \\ &= \sum_{a \in A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]. \end{aligned} \tag{3.25}$$

We can generalize this to multi-step *MDPs* by replacing the single-step reward with the Q-function, since the Q-function equals the discounted sum of future rewards. The result is called the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)] \tag{3.26}$$

This result is very satisfying because it applies to any probabilistic policy. In practice, usually, a Softmax policy is used for stochastic action prediction and a Gaussian policy is used for continuous action spaces. The most basic algorithm in the class of policy gradients is called the REINFORCE [87] algorithm and is explained in Algorithm 3. In REINFORCE, the policy gradient is used to updated the policy parameters. Instead of the real Q-function, the return v_t is used as an unbiased sample of $Q^{\pi_\theta}(s,a)$. This means, that instead of keeping track of an approximated (biased) Q-function, a trajectory is sampled

before an update is performed, which is a sample of the real Q-function for the current policy. The parameters are updated with

$$\Delta\theta_t = \alpha\nabla_\theta \log \pi_\theta(s_t, a_t)v_t, \tag{3.27}$$

where α is the learning rate. Another, more sophisticated policy gradient algorithm, which is used in practice is called *Proximal Policy Optimization*[71].

Algorithm 3 REINFORCE

Using $v_t = r_t + r_{t+1} + ... + r_T$ as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$
Initialize θ arbitrarily
foreach *episode* $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T \sim \pi_\theta\}$ **do**
  for $t = 1$ *to* $T - 1$ **do**
    $\theta \leftarrow \theta + \alpha\nabla_\theta \log \pi_\theta(s_t, a_t)v_t$
  end for
end foreach

3.2.3 Actor-Critic Methods

Although policy gradient methods eliminate some limitations of value-based methods, they introduce new problems. One problem is a high variance in the gradients, which comes from the fact, that during training a number of trajectories are sampled before the network is updated. Due to the stochastic nature of the policy, these trajectories can deviate from each other to a great degree, which causes a high variance in the gradients. Some improvements over the REINFORCE algorithm have been made to tackle its problems, but most modern algorithms used in practice use actor-critic methods.

Actor-critic methods combine the two concepts of value-based *RL* and policy-based *RL*. A *critic*, which is another function approximator, is used to estimate the Q-function $Q_{\theta^Q}(s, a)$ and an *actor* is used to approximate the policy π_{θ^π}. Therefore actor-critic methods maintain two sets of parameters, the critic's parameters θ^Q, which approximate the Q-function and the actor's parameters θ^π which approximate the policy. Actor-critic algorithms follow an approximate policy gradient.

$$Q_{\theta^Q}(s, a) \approx Q^{\pi_{\theta^\pi}}(s, a) \tag{3.28}$$

$$\nabla_{\theta^\pi} J(\theta^\pi) \approx \mathbb{E}_{\pi_{\theta^\pi}}[\nabla_\theta \log \pi_{\theta^\pi}(s, a)Q_{\theta^Q}(s, a)] \tag{3.29}$$

The critic is solving the problem of policy evaluation, which makes it possible to train the agent in *TD* fashion, as is done in *DQN*. Policy evaluation means, that the current policy is evaluated by sampling and accumulating the reward of a trajectory until the

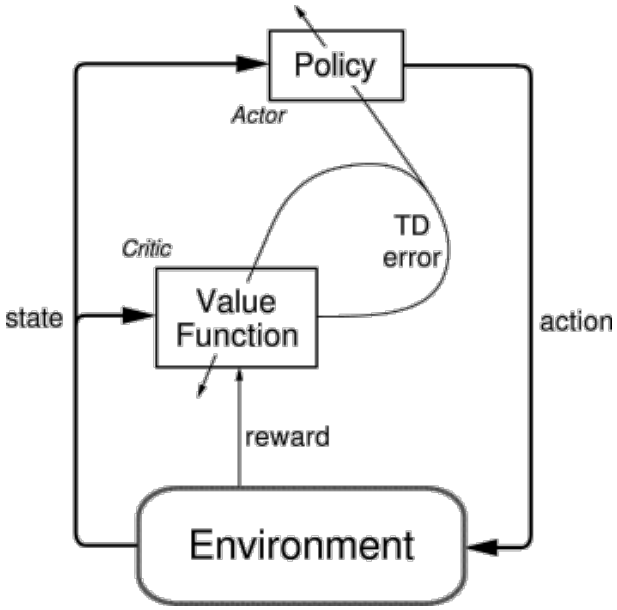


Figure 3.7: Basic principle of actor-critic methods. A value function is used to estimate the expected reward for the policy gradient. This allows updating the value function and the policy network with every transition sample. The Bellman equation is used to update the value function and the current approximation of the value function is used to compute the policy gradient to update the policy network.

agent's terminal state. In actor-critic methods, instead of sampling an entire trajectory, the Q-function is queried at the current state, to obtain an estimate of the expected reward. Consequently, the variance problem, which arises from sampling entire trajectories is reduced by updating the critic with tuples $\langle s, a, r, s' \rangle$ from single transitions with the Bellman loss as in value-based methods (Equation 3.11). The policy network can then be updated with the approximated Q-function. The Q-Actor-Critic (QAC) algorithm in Algorithm 4 explains the basic idea behind actor-critic methods.

More advanced algorithms in the class of actor-critic methods are Advantage Actor Critic (A2C) [76], Asynchronous Advantage Actor Critic (A3C) [55] and Deep Deterministic Policy Gradients (DDPG) [48]. The latter will be explained in the next section.

3.2.3.1 Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients (DDPG) [48] is one of the most widely used algorithms from the class of actor-critic methods. The algorithm will be explained in detail because the methods used in this thesis are based on the *DDPG* algorithm. As the name suggests *DDPG* allows to make deterministic predictions, but in contrast to Q-learning in a continuous action space. *DDPG* is an off-policy method that uses the Bellman equation to learn a Q-function, which then is used to learn the policy.

The algorithm is similar to Q-learning, in a sense that if you know the optimal Q-

Algorithm 4 QAC algorithm

Initialize state s
Initialize the parameter sets θ^π, θ^Q
Initialize learning rates α, β
Sample action $a \sim \pi_\theta$ from the current policy
for *each step* **do**
  Sample reward $r' \sim \mathcal{R}^a_{s,s'}$ and transitions $s' \sim \mathcal{P}^a_{s,s'}$ from the environment
  Sample action $a' \sim \pi_{\theta^\pi}(a'|s')$ from the current policy
  Update the policy parameters: $\theta^\pi \leftarrow \theta^\pi + \alpha \nabla_{\theta^\pi} \log \pi_{\theta^\pi}(a|s) Q_{\theta^Q}(s,a)$
  Compute the TD-error: $\delta = r + \gamma Q_{\theta^Q}(s',a') - Q_{\theta^Q}(s,a)$
  Update the parameters of the Q-function with gradient descent on the TD-error: $\theta^Q \leftarrow \theta^Q - \beta \nabla_{\theta^Q} \delta$
  $a \leftarrow a', s \leftarrow s'$
end for

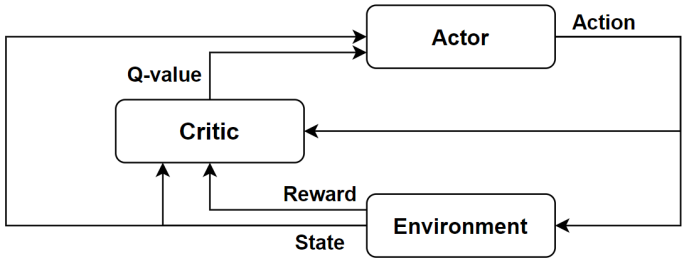


Figure 3.8: Basic principle of the DDPG algorithm. The actor predicts a deterministic action and the critic evaluates the action. The critic is updated with the Bellman loss (Equation 3.11) and the actor is updated by propagating the gradient through the critic network as shown in Figure 3.9.

function $Q^*(s,a)$, you can derive the optimal action a^* by the following property:

$$a^* = \arg\max_a Q^*(s,a) \tag{3.30}$$

To apply Q-learning on a continuous action space, the maximization in Equation 3.30 becomes an optimization problem instead of a simple lookup, which increases the complexity of the problem. To tackle this problem, *DDPG* uses an actor-critic architecture, where a second function approximator, the actor makes a *deterministic* prediction based on the current state and the critic, representing the Q-function, evaluates the action taken by the actor. The main difference to other actor-critic methods is, that the actor directly maps from a state to an action instead of predicting the probability distribution over actions.

Similar to *DQN*, the agent collects tuples of experience by following the current policy which are stored in an experience replay buffer. For exploration, noise is added to the

action predictions. In the original paper [48] an Ornstein-Uhlenbeck noise process [80] was used, which is a correlated noise process and is necessary for some problems to converge. For some problems however Gaussian noise is enough. The Q-network Q is parameterized with θ^Q and the policy network μ with θ^μ. To indicate the deterministic policy network, the letter μ is used instead of the previously used π, which referred to a stochastic policy. *DDPG* also uses the concept of a target network, with the difference, that instead of cloning the networks every fixed number of steps, the algorithm uses *soft updates* after every step. The equation for a soft update is shown in Equation 3.31. The target network's are indicated with an apostrophe.

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \qquad \tau \in [0,1] \tag{3.31}$$

Intuitively, the target networks weights are approaching the original weights. τ is a parameter that controls the speed of how fast the weights are approached. The smaller τ, the slower the target weights approach the original weights.

The Q-network is updated utilizing the Bellman equation. The *TD*-target y_i for sample i is computed as

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}). \tag{3.32}$$

As one can see in Equation 3.32 the next action is computed with the policy network μ', which is different to *DQN* and solves the maximization problem explained in Section 3.2.2. With the *TD*-target of Equation 3.32, we minimize the mean-squared error loss on a mini-batch of size N.

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{3.33}$$

For the policy function, the objective is to maximize the expected return, which is given by the Q-network.

$$J(\theta) = \mathbb{E}[Q(s,a)|s = s_t, a_t = \mu(s_t)] \tag{3.34}$$

Since we want to maximize the objective function of Equation 3.34, we calculate the gradient with respect to the actor parameters and use gradient ascend. The objective function is differentiable with respect to the actor parameters, since the output of the policy network is fed into the Q-network to predict the Q-value. Therefore we can use the chain rule to compute the gradient.

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta^\mu} \mu(s|\theta^\mu) \tag{3.35}$$

Since we are updating off-policy, we sample mini-batches and take the mean of the gradient.

$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N} \sum_i \left[\nabla_a (Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s = s_i \right] \tag{3.36}$$

The complete algorithm is shown in Algorithm 5.

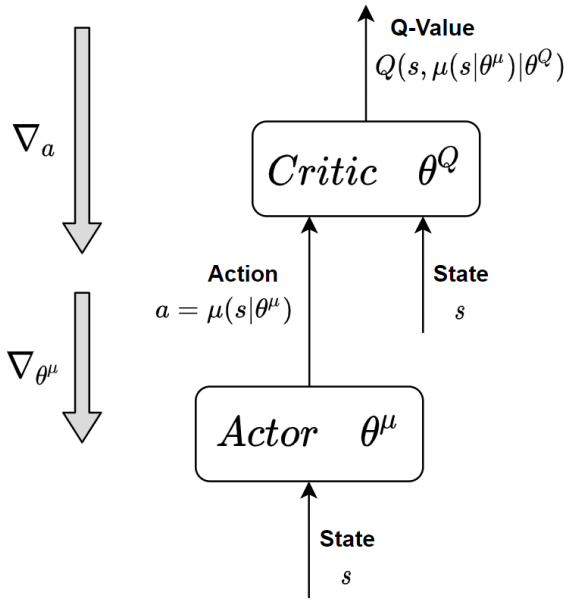


Figure 3.9: To compute the policy gradient, the gradient is propagated through the critic network to maximize the output of the critic: the Q-value. The chain rule is used to compute the gradient.

3.3 Multi-Agent Reinforcement Learning (MARL)

The objective of this thesis was, to implement efficient multi-landmark detection algorithms. This objective can be formulated as a multi-agent problem, where each agent is responsible for one landmark and communication between the agents ensures optimal behavior of the individual agents. The topic of Multi-Agent Reinforcement Learning (MARL) is a broad research field and includes elements of game theory and of social and behavioral theories. This section explains some basic concepts of *MARL* and is mostly based on [9], which provides a much more detailed introduction to the topic.

3.3.1 Multi-Task vs. Multi-Objective vs. Multi-Agent

A distinction has to be made between multi-task, multi-agent and multi-objective *RL*. Multi-task *RL* describes the scenario where an agent is taught multiple objectives. The agent has to be told which objective to follow. We follow a multi-task approach, which will be explained in more detail in Section 4.3.2. The objectives in our approach are the

Algorithm 5 DDPG

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for *episode=1,M* **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for *t=1, T* **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random mini-batch of N transitions $(s_i, a_i, r_i, s_{i+1}$ from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
    $\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N}\sum_i \left[\nabla_a (Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s = s_i\right]$
    Update the target networks:
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

individual landmarks, the agent has to be told which landmark to follow. Note that in multi-task *RL* a single agent is used, therefore multi-task *RL* is not part of multi-agent *RL*. Multi-Objective Reinforcement Learning (MORL) is similar to multi-task *RL* and also does not belong to the field of multi-agent *RL*. In *MORL* a single agent is trained for multiple objectives, however, the objectives are conflicting alternatives. The optimal policy for an agent in *MORL* is therefore the policy that achieves the Pareto optimum. Pareto optimality is achieved if no objective can further be improved without a loss in another objective. In *MORL* the scalar reward is replaced by multiple feedback signals, one for each objective. A tangible example is the production possibilities frontier, which is the problem of allocating various types of resources to produce various types of goods, such that the resources are allocated most efficiently. This problem, along with many other problems in economics and finance are part of multi-objective optimization.

MARL however deals with multiple agents, which are cohabitating in an environment. These agents either collaborate towards a common goal or compete, such that each agent wants to maximize its own reward at the expense of the other agents. The third possibility is that the agents neither collaborate, nor compete which is called a mixed cooperative-competitive environment. Due to the background of game theory, the environment in *MARL* is also called *game*.

3.3.2 Stochastic Game

The stochastic game is the generalization of the Markov decision process to the multi-agent case. The stochastic game is a tuple $\langle \mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{T}, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, where n is the number of agents, $\mathcal{S}$ is a finite set of states. $\mathcal{A}_i, i = 1, \ldots, n$ is the finite set of actions available to agent i, yielding the joint action set $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$. $\mathcal{T} : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ are the transition probabilities and $\mathcal{R}_i : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to \mathbb{R}, i = 1, \ldots, n$ are the reward functions of the individual agents. The stochastic game is visualized in Figure 3.10 as a *MARL* problem.

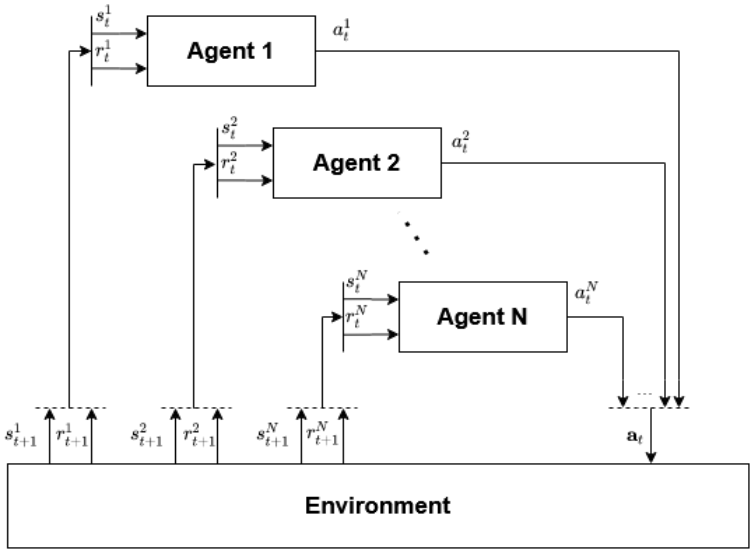


Figure 3.10: Interaction of multiple agents with the environment.

3.3.3 Fully Cooperative Games

The game is called fully cooperative if the reward functions for all agents are the same: $\mathcal{R}_i = \mathcal{R}_j \; \forall i, j \in 1, ..., n$. The learning goal is to maximize the common discounted reward. If a centralized controller were to be used, the stochastic game would reduce to an *MDP*, where the action space $\mathcal{A}$ is the joint action space $\boldsymbol{\mathcal{A}}$ and the reward function $\mathcal{R}$ is any linear combination of the individual reward functions of the stochastic game. In *MARL*, however, the agents are individual decision-makers, but still, in some cases, it might be possible to learn a common optimal Q-function and use greedy policies $\pi_i(s)$, such that

$$\pi_i(s) = \arg\max_{a_i} \max_{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots a_n} Q^*(s, \mathbf{a}), \tag{3.37}$$

with $\mathbf{a}$ being the joint action. However, for most problems, this naïve approach is not applicable. The biggest hurdle in fully cooperative tasks is coordination. If multiple joint actions result in the same return, the agents may assume wrong actions made by the other

agents, which makes proper coordination of multiple agents impossible. Consequently, many *coordination-based* methods were designed to solve the coordination problem. Algorithms that don't consider coordination are called *coordination-free methods*.

3.3.4 Fully Competitive Games

A game is fully competitive if the number of agents n equals 2 and the reward functions are opposing, such that $\mathcal{R}_1 = -\mathcal{R}_2$. In these games, the minimax principle can be applied: Maximizing one agent's action under the worst-case assumption, that the opponent will always try to minimize it. One important algorithm that solves the *MARL* problem for a fully competitive environment is the minimax-Q algorithm [50], which applies the minimax principle to Q-learning.

3.3.5 Mixed Stochastic Games

Mixed stochastic games distinguish from fully cooperative games and fully competitive games in that no constraints are imposed on the reward functions. Therefore mixed games can come in many forms and no universal algorithm exists, that can solve any mixed problem.

4 Methods

Contents

This chapter explains the methods of the contributions made in this thesis in detail and is divided into three sections. In the first section, the reference framework from the Deep Q-Network (DQN) based approach of Ghesu et al. [33] for medical landmark localization is explained. The reference framework has been rebuilt from scratch, to perfectly match the workflow of the other experiments for comparable results. The subsequent sections explain the contributions, starting with a Deep Deterministic Policy Gradients (DDPG) based framework for landmark localization, which allows continuous prediction and therefore a variable step size. In the third section, the baseline *DQN* algorithm and our *DDPG* based algorithm will be extended for multiple landmarks. The goal was to use a single neural network to make the predictions. Previous Reinforcement Learning (RL) based approaches for multi-landmark localization used a specific neural network for every landmark which is computationally very demanding, especially with increasing number of landmarks. To do so, we evaluated two different approaches. The first one is a multi-task approach (see Section 3.3.1), where a single input to the neural network predicts the actions for multiple target landmarks. We implement a multi-task *DQN* algorithm and a multi-task *DDPG* algorithm. These multi-task algorithms are trained with novel sample-efficient methods, described in Section 4.3 of this chapter. The second approach is derived from Multi-Agent Reinforcement Learning (MARL) and is given the name Simultaneous Acting and Localizing (SAL). This approach divides the agent into N sub-agents, where N is the number of landmarks. A single controller is used to update the sub-agents simultaneously.

4.1 Landmark Localization with DQN

The *DQN* algorithm is explained in Section 3.2.1.1. This section explains how the *DQN* algorithm is applied to the problem of landmark localization and is based on the work of Ghesu et al. [33]. The algorithm from [33] has been rebuilt, to match our work environment and to obtain comparable results in terms of training time and other measures. The *DQN* approach is included in the methods section because it helps to understand the following sections and because we use our own implementation of this algorithm as a baseline experiment. To formulate the problem of landmark localization as an *RL* problem, we first have to define the terms *state* and *environment* in this context. As an environment, medical images are used. The detailed specifications of the dataset are listed in Section 5.1. An episode takes place in a single image. After an episode was played, the image is updated. The agent starts an episode by being placed randomly in the image. From the location we define the observation to be an image patch centered around the agent's position. For the understanding of the algorithm, we describe the image patch to be a simple crop with a fixed size around the agent's location. In the experiments, we use more advanced methods to represent the state. These will be explained in Section 4.1.3. The agent further needs to be equipped with a set of actions, such that the problem can be formulated as a Markov Decision Process (MDP), which is a necessary condition for value-based *RL* methods. As explained in the theory chapter, a *MDP* is a tuple of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition functions and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function. The set of actions in *DQN* has to be discrete and is defined as

$$\mathcal{A} = \{up,\ down,\ left,\ right\}. \tag{4.1}$$

Since the environment in which the agent is interacting is deterministic, the transition probabilities are 1 for every possible transition, implying that an action leads to the expected behavior. Taking action *up* for example moves the agent exactly one pixel upward. The missing element for the *MDP* is the reward function $\mathcal{R}$, which can be simply defined as the change in euclidean distance from the agent to the landmark.

$$\mathcal{R} = \alpha(d_s - d'_s) \tag{4.2}$$

$$d_s = ||\mathbf{s} - \mathbf{g}||, \qquad d_{s'} = ||\mathbf{s'} - \mathbf{g}||, \tag{4.3}$$

with d_s being the distance from state $\mathbf{s}$ to landmark $\mathbf{g}$, d'_s the distance from state $\mathbf{s}'$ i.e. the new state after taking an action to landmark $\mathbf{g}$, and α is a scaling factor. In the original *DQN* paper [56] it has been shown, that clipping the reward to the interval $[-1, 1]$ increases stability during training. Consequently, we set the scaling factor to 1, which ensures the reward to be in the interval $[-1, 1]$. We introduce the scaling factor, because

it will be essential for later algorithms. For *RL* problems to converge it is important to include a discount factor into the *MDP* such that $MDP = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ with $\gamma \in [0, 1]$. The values for parameters like the discount factor will be listed in the experimental setup section for the experiments individually.

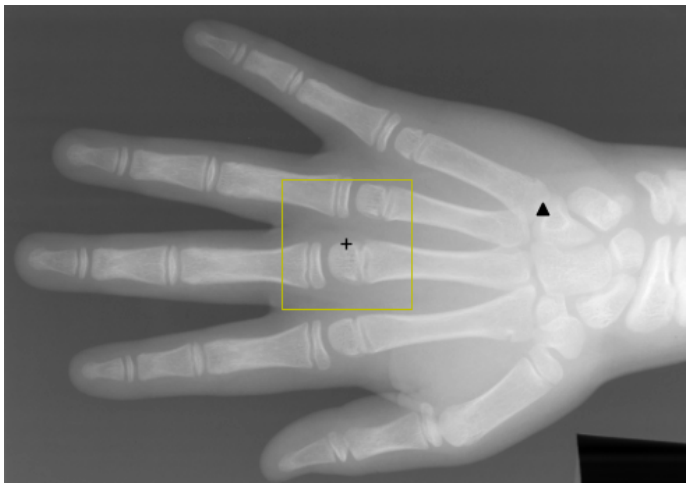


Figure 4.1: An example of the environment with the agent indicated with a black cross-hairs and the landmark indicated with a black triangle. The yellow square box is the observation, which is fed into the neural network.

In the *DQN* algorithm, the agent is parameterized by some function approximator to approach the optimal action-value function $Q(s, a; \theta) \approx Q^*(s, a)$ with θ being the parameters. As a function approximator a Convolutional Neural Network (CNN) is used, which includes both feature extractor and regressor and can be trained end-to-end. The input to the *CNN* is the cropped image as explained above. The architecture is explained in detail in section 4.1.2. As explained in Algorithm 2 of the theory section, the *CNN* is trained on the Bellman loss shown in Equation 3.17. As a target network Q' with weights θ^- a copy of the *CNN* is kept, which is updated every C steps. D indicates the replay buffer which stores the experience tuples and is restricted to a maximum size of 100000. Experience tuples are collected with an ϵ-greedy strategy, where the agent starts by taking random actions and the further the training proceeds, the more actions are chosen according to the current policy. A decay factor gradually reduces ϵ until a lower threshold of $\epsilon = 0.1$ is reached, which means, that the agent samples random actions with a probability of 10% and follows the current policy with a probability of 90%.

4.1.1 Multi-Scale Framework

Following [3] we further implement a multi-scale framework, to reduce the number of steps needed to localize the target landmark. In this multi-scale framework, the agent starts with large action steps and reduces the step size as he localizes the target landmark. The final landmark prediction is the result on the lowest level of the multi-scale search, where

the agent's step size is one pixel. We choose a multiplier of 3 for our experiments, which means that the agent starts with a step size of 9 and as soon as it signalizes termination reduces the step size to 3. At the next termination signal, the agent reduces the step size to one pixel and finally localizes the target with the highest possible accuracy. The Field of View (FoV) on each level is coupled to the step size, which means, that the agent starts with a *FoV* of 9 times the size at the lowest level. The *FoV* is increased by sampling with a fixed spacing around the current location of the agent. The same *DQN* is shared between all levels in the hierarchy.

4.1.2 Network Architecture

The network architecture is a standard *CNN* architecture and consists of a convolution block and a Fully Connected (FC) block. The convolution block uses the convolution operation to extract features, while the *FC* block is used as a regressor to map the features to Q-values. An overview of the architecture is shown in Figure 4.2. Figure 4.3 further describes the convolution block in more detail. The network uses convolutional layers, which are always followed by a Rectified Linear Unit (ReLU) activation function. The convolutional layers apply the discrete 2D convolution operation with a weight kernel $K \in \mathbb{R}^{m \times n}$, where we use $m = n = 3$ to define the kernel size for all convolutional layers. The convolution operation is defined by Equation 4.4, where i, j are the image coordinates of image I, S is the resulting (convolved) image and $*$ is the convolution operator. The kernel weights are learned as part of the training process with the Stochastic Gradient Descent (SGD) based Adam optimizer [39]. Also, we use zero-padding to ensure the convolved image to be of the same size as the input image.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n K(m,n) I(i-m, j-n) \tag{4.4}$$

The *ReLU* function is defined as

$$f(x) = \begin{cases} 0 & if \quad x < 0 \\ x & otherwise \end{cases} \tag{4.5}$$

and is used to allow the approximator to fit non-linear functions. Finally, average pooling is used to downsample the latent representations between convolution layers. Average pooling is a process where pixels of a defined kernel size are combined into a single pixel by averaging over the pixels. Average pooling has the advantage over the more frequently used Max-pooling, that the information from every pixel in the kernel contributes to the final pixel. We use a kernel size of 2×2, such that both image dimensions are halved after every average pooling layer. At the end of the convolution block, the features are flattened and fed to the *FC* block. The *FC* block contains three consecutive *FC* layers with 256 units each. Each *FC* layer is activated with the *ReLU* function. The output of the *FC* block is passed to a final dense output layer with 4 neurons without activation,

where each neuron predicts the Q-value for one action.

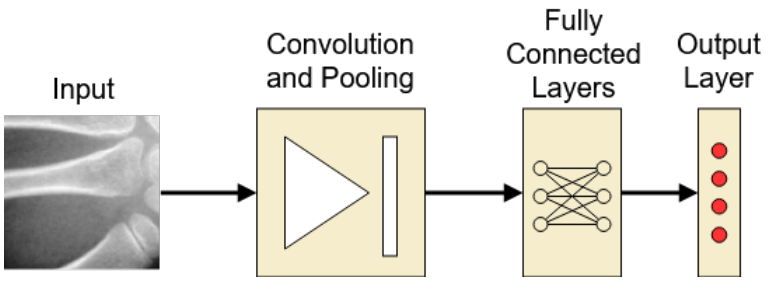


Figure 4.2: A diagram representing the network architectures. This is the general architecture, used for all experiments. The architecture is adapted to other experiments by exchanging the input- and output layer. The convolution block comprises several cycles of the operations convolution, *ReLU* activation and average pooling.

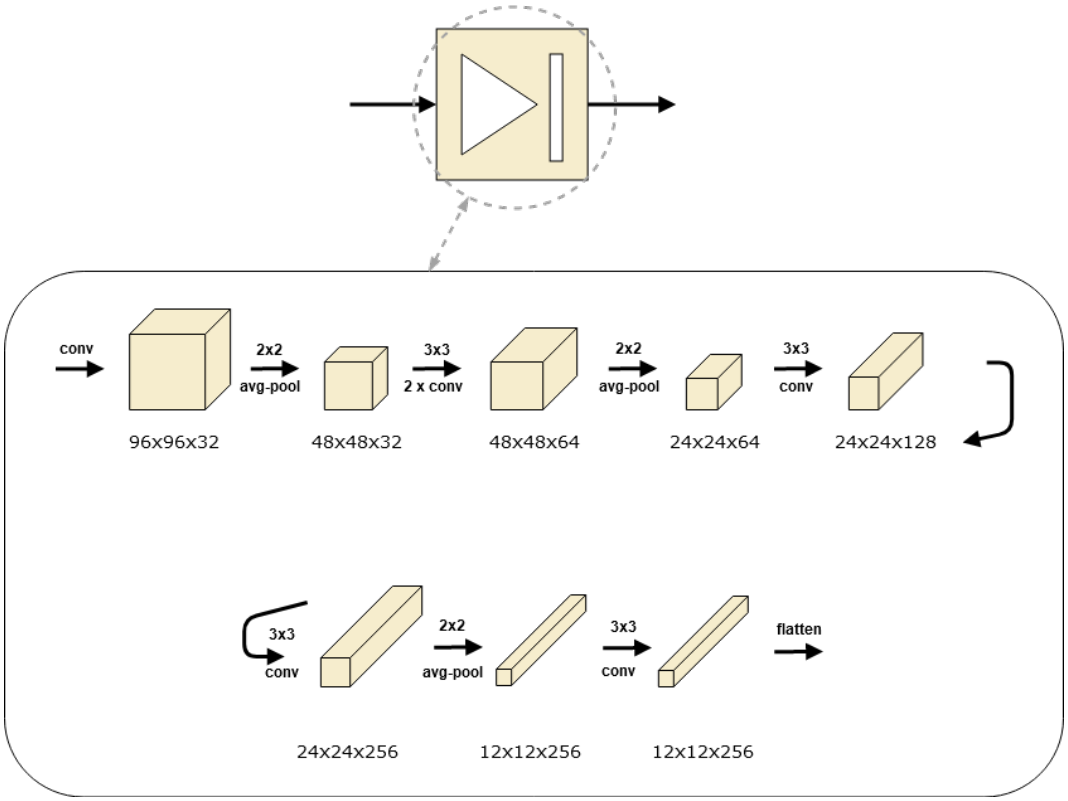


Figure 4.3: This schematic shows the convolution block in more detail. It consists of multiple convolution operations, each of which is activated with the *ReLU* function and average pooling layers.

4.1.3 State Representation

We evaluate various state representations, which allows us to tackle specific problems. The simplest and least accurate method is to crop an image patch centered around the agent's position. The *FoV* is a hyper-parameter that has to be determined. If chosen too large, computational demand increases drastically and if chosen too small, the *FoV* is not large enough to provide enough information to the agent. Therefore we use more advanced state representations to combine computational efficiency with high accuracy. We conduct various experiments with different state representations. The type of state representation used is listed in the experimental setup chapter at the corresponding experiment.

Image Pyramid

Image pyramids are used frequently in computer vision tasks. Image pyramids are multi-scale representations of images and are distinguished between *low-pass pyramids* and *band-pass pyramids*. A low-pass pyramid is constructed by smoothing the image with a filter kernel S_σ

$$B_l = B_{l-1} * S_\sigma \qquad \text{for} \quad l = 1 \dots L, \tag{4.6}$$

where B_l denotes the image at level l and $*$ indicates the convolution operator. After smoothing, the image is sub-sampled by a factor of 2 on each image dimension. The process is recursive and can be repeated until the desired level is reached or until the image size is too small to further be divided. A band-pass pyramid on the other hand uses second-order differential operators such as the Laplace operator to extract edges.

To address the problem of having a too-small *FoV* we use a low-pass image pyramid and create image patches of the same size on different pyramid levels. Due to image sub-sampling on higher levels, the patches cover a larger *FoV*. Using an image pyramid has the advantage of covering a larger *FoV* while saving computation power. To represent the same *FoV* as in the *single image patch* method, we only need to feed an image of half the size to the neural network. Higher levels of the image pyramid provide global information to the agent, which is needed for rough orientation in the image and do not need high resolution. The lowest level provides structural information at the point of interest and is therefore needed in full resolution. An example of an image pyramid is shown in Figure 4.4.

Embedded Image Pyramid

The use of image pyramids can be further optimized since we are using some information twice. The central regions from higher pyramid levels have already been represented by lower levels in higher resolution. These regions are redundant and can be removed. Therefore we merged individual pyramid levels into a single representation, where the center region of the resulting image contains the center region of the lowest level i.e. the

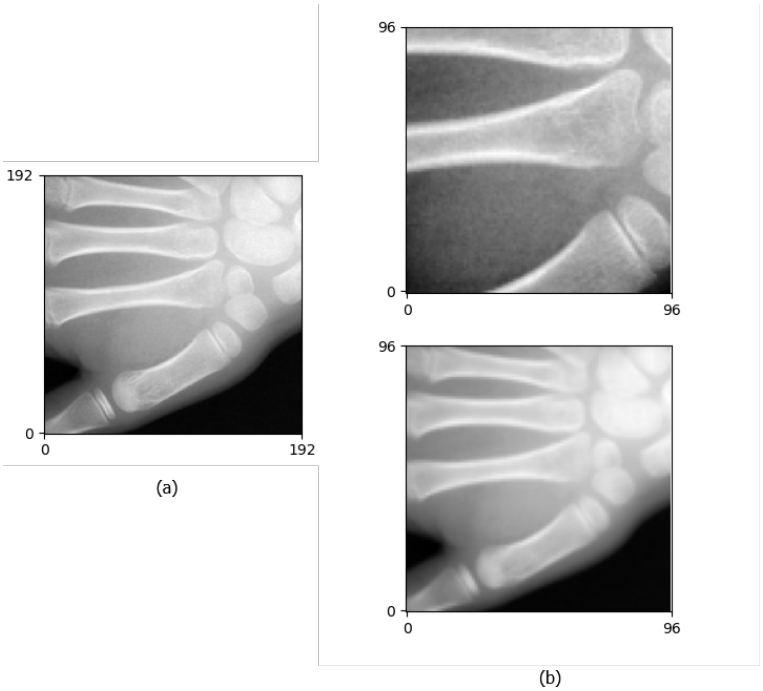


Figure 4.4: A single image patch (a) and an image pyramid with two levels (b). To cover the same FOV, the image pyramid only needs half the amount of pixels, compared to the full resolution image, while maintaining the same resolution at the important part of the image (b, top).

level with highest accuracy and outer rings represent higher levels with lower accuracy. Even though parts of the higher pyramid level (also parts that are not represented by lower levels) are now occluded, our experiments showed, that this representation yields best results. We assume the lower levels to be more important, therefore add a higher percentage of lower levels to the final image. An example of this approach is shown in Figure 4.5. The contributing percentage of each level is computed by the following formula:

$$contribution = \frac{\frac{1}{2^l}}{\sum_{i=1}^{L} \frac{1}{2^i}}, \tag{4.7}$$

with l being the evaluated level and L the total number of levels.

As an example consider an image pyramid with 3 levels. The contribution of the lowest level is

$$c_1 = \frac{\frac{1}{2^1}}{\frac{1}{2} + \frac{1}{4} + \frac{1}{8}} = 57.14\%. \tag{4.8}$$

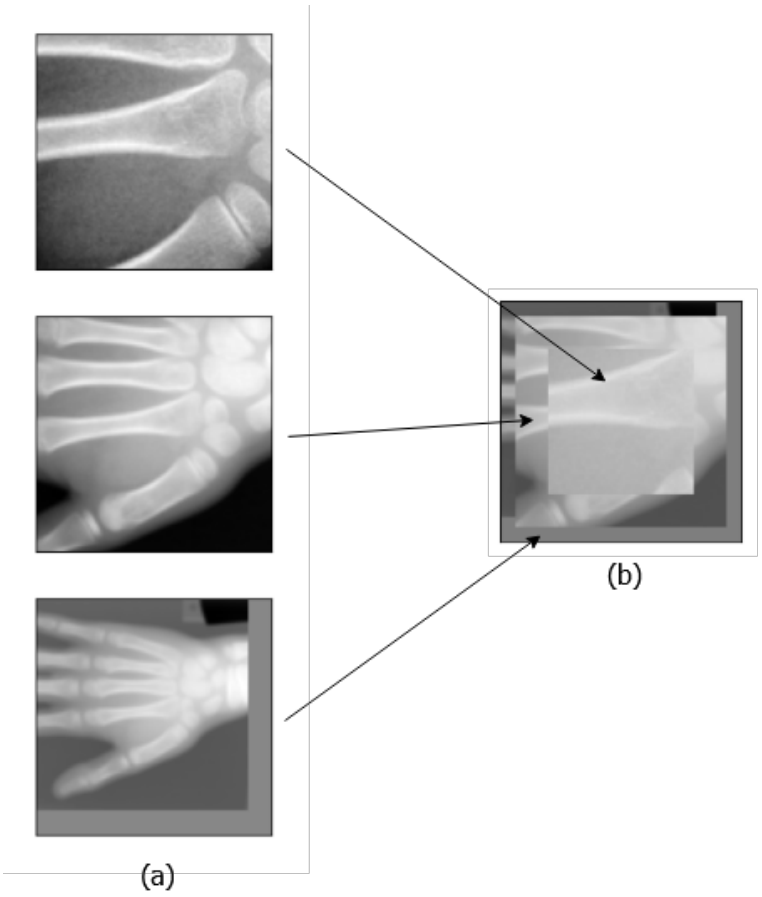


Figure 4.5: An image pyramid with 3 levels (a) combined into a single image (b).The resulting image has L times fewer pixels compared to the normal image pyramid with L being the number of pyramid levels. As one can see, the third pyramid level (a, bottom) already covers nearly the entire image. The grey rim at the bottom and the right side shows added zero-padding because the image patch exits the image. Therefore in our experiments, we use a maximum of three pyramid levels.

4.1.4 Terminal State

A terminal state is reached, if the agent has localized the landmark or if the agent reached the maximum number of steps, which is a predefined hyper-parameter. During training, the agent is terminated if it has localized the target landmark within a radius of R pixels centered around the target landmark. We evaluated different values of R and noticed that this parameter has a strong impact on the final result.

During inference, we terminate the agent if it oscillates between two pixels. Another option to terminate the episode during inference is, to use a trigger action which has been shown in [52] and [10]. This method however increases complexity by increasing the action space. Therefore we adopt the effective oscillating approach from [3].

4.2 Continuous Landmark Localization with DDPG

Using the *DQN* algorithm for landmark localization as explained in Section 4.1 has a major limitation. Since the *DQN* algorithm allows only discrete actions, the agent has to search with a fixed step-size. Alansary et al. [3] tackle this problem with a multi-scale framework, where initially they sample the image patch with a fixed spacing greater than one to cover a larger *FoV* and accordingly take large action steps. When the agent starts oscillating, they reduce spacing and step size until the original resolution and a step size of one is reached.

We tried a different approach, where we increase the action space by adding multiple actions for every direction with different step sizes and linear combinations of those, such that it allows the agent to move to specific locations on a grid around its current position. However, this results in a very large action space. Training a *DQN* algorithm with a large action space is a difficult task because the exploration space of the agent grows exponentially with the action space size. Consequently, we were not able to achieve convergence with this training setup. Therefore to be able to predict varying step sizes, we extended the algorithm to a continuous action space using the *DDPG* algorithm. The *DDPG* algorithm, explained in Section 3.2.3.1 is an algorithm to predict *continuous* and *deterministic* actions given a state, which makes it suitable for the landmark localization problem. We use the same state representations as in Section 4.1.

4.2.1 Action Representation

The *DDPG* algorithm allows us to predict continuous actions as explained in Section 3.2.3.1. Therefore, we represent the action $\mathbf{a}$ as a vector with two coordinates, which are restricted to a lower bound $B_l = -50$ and a higher bound $B_h = 50$.

$$\mathbf{a}_t = \begin{bmatrix} s_x \\ s_y \end{bmatrix} \qquad s_x, s_y \in [-50, 50] \tag{4.9}$$

The predicted action vector is simply added to the agent's current position $\mathbf{p}$ to take a step.

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{a}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} s_x \\ s_y \end{bmatrix} \tag{4.10}$$

Since neural network predictions are in $\mathbb{R}$, we round these values to integer values, to move on discrete image coordinates. In the Discussion chapter, we explain how the continuous property of this approach can be used for landmark localization with sub-pixel resolution.

4.2.2 Terminal State

The Q-network predicts a set of Q-values, where the action is implicit and derived by maximizing the Q-value. This does not allow to derive a terminal state directly from the prediction of the Q-network, except in the unusual case, where an additional trigger action for the terminal state is added. The policy network in the *DDPG* algorithm however directly predicts the action. Consequently, we can put a condition on the action and terminate the agent if this condition is met. We formulate the termination criterion such that the agent is terminated if the following condition is met.

$$|x| < 0.5px \quad \wedge \quad |y| < 0.5px \tag{4.11}$$

This condition is optimal for the landmark localization problem with the reason that the agent stops moving if the condition is met because of the `round` operator.

4.2.3 Architectures

As already explained in Section 3.2.3.1, the DDPG algorithm uses four neural networks. The actor network, the critic network and delayed copies of both networks as target networks. Our experiments showed, that best results are obtained if both actor and critic network use the same architecture. The networks only differ at the input- and the output layer. The actor network's output layer is reduced to two neurons for the two continuous actions. The critic network receives an additional input at the first dense layer, namely the output of the actor and the critic's output layer is reduced to a single neuron to predict the Q-value. A conceptual representation of both networks is shown in Figure 4.6 and Figure 4.7. The basic network structure remains the same as in Section 4.1.2.

4.3 Multi-Landmark Localization

We present three different methods for multiple landmark localization, as briefly introduced in the opening section of this chapter. In the first two algorithms, we formulate the problem as a multi-task *RL* problem. The first method extends the *DQN* approach to a multi-task agent. The second method extends the *DDPG* approach to a multi-task agent and presents a sample-efficient way to train this algorithm. Finally, the third method provides a way to simultaneously detect multiple landmarks by using a method derived from *MARL*.

4.3.1 Sample-Efficient Training with Multi-Task DQN

The simplest method to apply *RL* to multiple landmarks is, to train an individual agent for each landmark, which is equivalent to the problem of single landmark localization. The training of *RL* algorithms is very time consuming, because additionally to the optimization of the neural networks, *RL* algorithms have a lot of overhead due to agent-environment

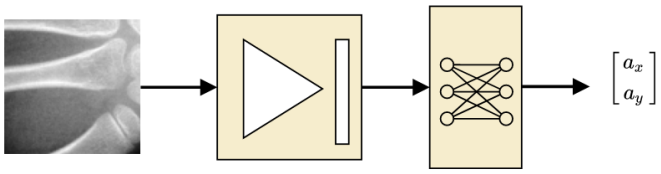


Figure 4.6: Conceptual representation of the actor network. The actor directly predicts the action as a displacement vector from the agent's current position.

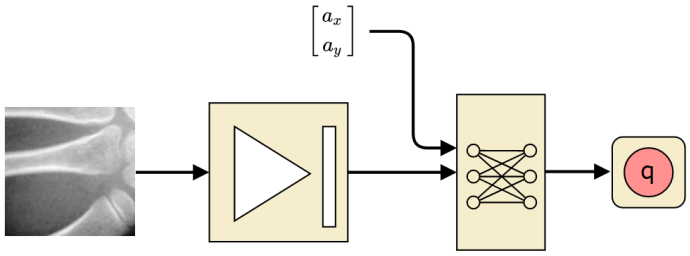


Figure 4.7: The critic belongs to the class of Q-networks and evaluates the actor's action. The action is directly fed to the first dense layer, along with the features extracted in the conv-block.

interactions. Therefore, training an individual agent for every landmark is very time consuming. In the problem of landmark localization, experience tuples of the form $\langle s, a, r, s' \rangle$ are stored in the replay buffer. As state s, we can only store the coordinates of the agent's position and additional relevant meta-data but we are not able to store the image patches themselves due to Random-Access Memory (RAM) limitations. Therefore we have to crop the image patches on the fly which poses a bottleneck to the training algorithm, especially, if these patches have to be copied repeatedly to the Graphics Processing Unit (GPU), which is used to accelerate neural network optimization. One possibility would be to process the image patches directly on the *GPU*, but it has been shown to be a very difficult task due to the limited instructions set of the *GPU* and the different preprocessing methods we use.

To reduce training complexity, Vlontzos et al. [83] followed the approach of sharing the weights in the convolutional layers between multiple agents, to exploit the fact, that every agent has to process the same structural information. However, since most of the parameters are located in the transition between convolutional layers and dense layers the improvement is marginal. Dense layers on the other hand cannot share weights, because they have to be trained on their specific landmark.

We develop a multi-task *DQN* agent for landmark localization. Multi-task *RL* is explained in Section 3.3.1 and essentially means training a single agent for multiple tasks, where we define the tasks to be the localization of individual landmarks. This allows training a single agent for multiple landmarks. During inference, the agent can then

localize the individual landmarks, one at a time. To implement a multi-task agent, we are learning a Q-function Q_n for every task. We extend the output layer of the Q-network, such that the prediction does not only consist of four Q-values (one for each action) but of N times four Q-values, a set of four Q-values for each task. We write the individual Q-functions in vector form as

$$\mathbf{Q}(s,a) = \begin{bmatrix} Q_1(s,a) \\ \vdots \\ Q_n(s,a) \\ \vdots \\ Q_N(s,a) \end{bmatrix}, \tag{4.12}$$

with Q_n being the Q-function of the n^{th} task.

For multiple tasks, it is important to define a reward function for each task. Equivalent to the single-landmark case, we define the reward function as the change in euclidean distance to the target landmark. For multiple landmarks, we write this reward function as a vector of the form:

$$\mathbf{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_n \\ \vdots \\ r_N \end{bmatrix} = \begin{bmatrix} ||\mathbf{p} - \mathbf{g}_1|| - ||\mathbf{p}' - \mathbf{g}_1|| \\ \vdots \\ ||\mathbf{p} - \mathbf{g}_n|| - ||\mathbf{p}' - \mathbf{g}_n|| \\ \vdots \\ ||\mathbf{p} - \mathbf{g}_N|| - ||\mathbf{p}' - \mathbf{g}_N|| \end{bmatrix}, \tag{4.13}$$

where $\mathbf{p}$ is the agent's position before the transition, $\mathbf{p}'$ is the position after the transition and g_n is the position of target landmark $n \in 1, ..., N$. This equation shows, that the reward for each task is only dependent on the agent's position before and after the transition. Since the reward is the element that provides information to the Q-function, the Bellman equation can be computed after every transition and for every task, regardless of the policy the agent follows. We use an ϵ-greedy policy π_m during training (Section 3.1.6) of a randomly selected target landmark $m \sim N$ to follow during an episode. After every episode, the target landmark is updated. We rewrite the Bellman equation as:

$$\mathbf{Q}(s,a) = \begin{bmatrix} r_1 + \max_a \gamma Q_1(a; s'_{\pi_m}) \\ \vdots \\ r_n + \max_a \gamma Q_n(a; s'_{\pi_m}) \\ \vdots \\ r_N + \max_a \gamma Q_N(a; s'_{\pi_m}) \end{bmatrix}. \tag{4.14}$$

From the Bellman equation, we derive the Bellman loss, which is minimized to fit the Q-network to the individual Q-functions. The Bellman loss for multi-task *DQN* becomes

the Mean Squared Error (MSE) between the current prediction $Q_n(s, a)$ of the state-action pair for task n and the Bellman equation of task n.

$$Loss = \mathbb{E}_{s,a,\mathbf{r},s' \sim D}[\frac{1}{N}\sum_{n=1}^{N}(r_n + \gamma \max_{\hat{a}} Q_n(\hat{a}; s'_{\pi_m}) - Q_n(s_{\pi_m}, a_{\pi_m}))^2], \tag{4.15}$$

where N is the number of landmarks and $\langle s_{\pi_m}, a_{\pi_m}, \mathbf{r}, s'_{\pi_m} \rangle$ indicates an experience tuple sampled from replay buffer D. The subscript π_m indicates that the experience tuple is collected by following the policy of a randomly selected landmark m.

4.3.2 Landmark Localization with Multi-Task DDPG

As a second contribution, we extend the *DDPG* algorithm to a multi-task algorithm, similar to the previous chapter. We developed a method to train a single agent for multiple landmarks, by extending the output layer of the actor network μ to $N \times 2$ neurons, where N is the number of landmarks and 2 the number of actions (x- and y prediction). The agent consequently follows a multi-task strategy, such that during inference, the agent is able to locate all landmarks, one at a time. This agent could be used as a foundation to a multi-agent problem, where N identical copies of the agent and a communication protocol are used to solve the Linear Bottleneck Assignment Problem (LBAP) [8]. The action i.e. the actor's output is represented as

$$\mu_t(s) = \begin{bmatrix} a_{1,x} & a_{2,x} & \dots & a_{N,x} \\ a_{1,y} & a_{2,y} & \dots & a_{N,y} \end{bmatrix} \qquad a_{n,x}, a_{n,y} \in [-50, 50]. \tag{4.16}$$

To move the agent in the direction of landmark n, the column vector n is chosen and added to the agent's current position $\mathbf{p}_t$.

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mu_t^n(s) = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} a_{n,x} \\ a_{n,y} \end{bmatrix} \tag{4.17}$$

4.3.2.1 Training

Training this method differs from training the standard *DDPG* algorithm. In the *DDPG* algorithm, the agent has a single action, or multiple actions, which are executed simultaneously. Consider the previous problem of single landmark localization. The agent in this case is equipped with two continuous actions, which are executed simultaneously. The movement in the x-direction and the movement in the y-direction. However, these actions are treated as a single action and are fed directly to the critic network as shown in Figure 4.7. If the agent follows a multi-task strategy, however, only one action is executed at a time. Therefore, each action has to be treated as an independent actor and consequently has to be evaluated with a specific critic network, which means, we have to keep track of N critic networks for N landmarks. Besides using a lot of memory resources, this is also

very sample inefficient, because, while the actor is updated once every time a transition is made, only one of the N critic networks is updated. Therefore to achieve the same result, we would need N times as many training steps, compared to single landmark localization. These problems can be eliminated by merging all N critic networks to a single one, which is possible, because of the neural network's universal approximation ability. The output of the critic network consequently does not consist of a single Q-value, but of N Q-values, one for each landmark, as shown in Equation 4.18.

$$\mathbf{Q}(s,a) = \begin{bmatrix} q_1 \\ \vdots \\ q_n \\ \vdots \\ q_N \end{bmatrix} \tag{4.18}$$

These Q-values evaluate, how good the agent's action was for each individual landmark. Note, that the critic is independent of the actor's policy. This means, if the agent follows a certain landmark, the critic always predicts the Q-values for every landmark because it evaluates the action given the agent's position. Therefore, we are also able to update the critic network with each step and for every landmark, which leads to much faster convergence.

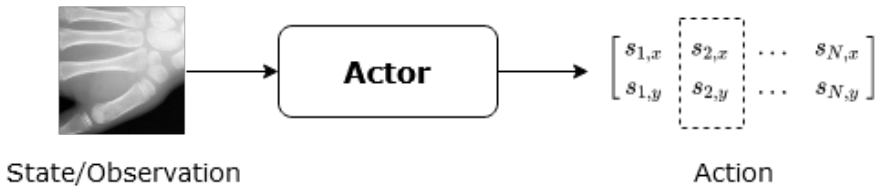


Figure 4.8: The actor receives an image patch and predicts the actions for all landmarks. The actor chooses a certain landmark and the corresponding action (indicated with the dashed rectangle).

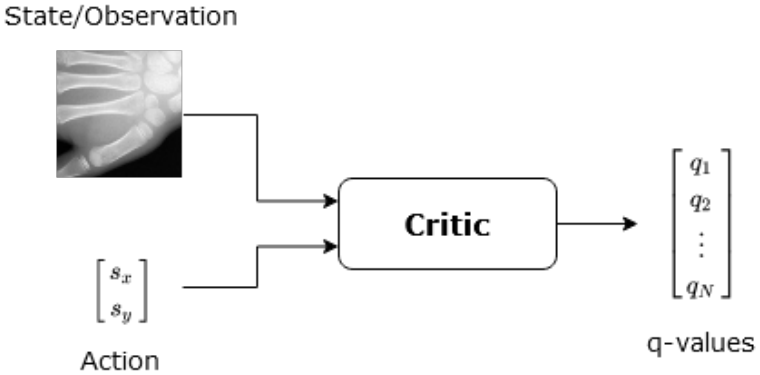


Figure 4.9: The critic evaluates a single action; the predicted Q-values are a measure of how good the agent's action was in a certain position for each of the landmarks.

The following section describes a complete iteration of the proposed algorithm. The agent starts at a random position, $\mathbf{p}_0 = [x_0, y_0]^T$ and chooses a random landmark m to follow during an episode. We retrieve the actor's prediction and perform the action corresponding to the chosen landmark $\mathbf{a}_{m,t_0} = [s_{m,x}, s_{m,y}] + \mathcal{N}_{t_0}$ with some exploration noise $\mathcal{N}$. We transition the agent to the state $\mathbf{p}_1 = [x_1, y_1]^T$ and obtain a reward for each possible landmark. We refer to this policy as π_m. The reward is again calculated as shown in Equation 4.13, where $\mathbf{g}_n$ denotes the position of the n^{th} landmark.

The experience tuples are stored in a replay buffer in the form $\langle s_{\pi_m}, \mathbf{r}, a_{\pi_m}, s'_{\pi_m} \rangle$. From these tuples the Bellman loss can be calculated, to update the critic network.

To calculate the Bellman loss, the Q-values of the next state have to be predicted. Recall, that to predict the Q-value of the next state in *DDPG*, the actor network has to be queried with the next state, to obtain the next action ($a = \mu(s')$). The Bellman equation for the standard *DDPG* algorithm is shown in Equation 4.19.

$$Q(s, a) = r_{t+1} + \gamma Q(s', \mu(s')) \tag{4.19}$$

The policy, i.e. the actor's prediction is denoted as $\mu(s)$. For multiple landmarks, the Bellman equation becomes a vector, since we do not only have a single Q-value, but a Q-value for every single landmark.

$$\mathbf{Q}(s, a) = \begin{bmatrix} r_1 + \gamma Q_1(s'_{\pi_m}, \mu^1(s'_{\pi_m})) \\ \vdots \\ r_n + \gamma Q_n(s'_{\pi_m}, \mu^n(s'_{\pi_m})) \\ \vdots \\ r_N + \gamma Q_N(s'_{\pi_m}, \mu^N(s'_{\pi_m})) \end{bmatrix} \tag{4.20}$$

Equation 4.20 is the label for the multi-landmark *DDPG* critic network, where $\mu^n(s)$ is the actor's prediction for the n^{th} landmark, as defined in Equation 4.16. To compute this label we need to predict the action from the next state, which is a single query. The Q-values however cannot be retrieved with a single query, since according to Equation 4.20, each Q-value has to be predicted with a different input for the action. This means, that for a critic network update, we need to make N predictions, with N being the number of landmarks, while in the normal *DDPG* algorithm only one prediction per critic update is needed. This increases the computational effort and RAM requirements, in practice, however, we observed, that the training time per sample is increased only marginally, compared to single landmark *DDPG*, because the bottleneck during training is the calculation of gradients. Since we are using *CNNs* with almost the same number of parameters, also the number of gradient calculations is similar. The critic network is updated with a gradient descent step on the Bellman loss for multi-task *DDPG*, which is the *MSE* between the Bellman equation for multi-task *DDPG* (Equation 4.20) and the prediction of the critic network (Equation 4.18). The loss is calculated as

$$Loss = \mathbb{E}_{s,a,r,s' \sim D} \left[\frac{1}{N} \sum_{n=1}^{N} (r_n + \gamma Q_n(s'_{\pi_m}, \mu^n(s'_{\pi_m})) - q_n)^2 \right], \tag{4.21}$$

where N indicates the number of landmarks and the expectation $\mathbb{E}$ again is used to indicate the use of mini-batches sampled from the replay buffer D. The actor is updated with gradient ascent by propagating the gradient through the critic network, as in the standard *DDPG* algorithm. However, since the critic can only evaluate one action at a time, each action has to be evaluated separately.

4.3.3 Simultaneous Acting and Localizing (SAL)

While our first method for multiple landmark localization has some significant advantages over existing approaches, it also has a major limitation. The method was optimized for sample efficient training, such that we are able to train the agent for all landmarks in one training run. During inference, however, the agent is only able to perform a single action and therefore can only detect one landmark at a time. This is not a problem, since we can use multiple identical copies of the agent and let them work in parallel, where each agent searches for a particular landmark. In some scenarios, however, we want the agents to communicate. In our second approach for multiple landmark localization, we follow an approach similar to a collaborative multi-agent system. However, we only use a single agent, to perform simultaneous prediction and execution of multiple actions by keeping track of multiple positions. The agent's state s in this scenario is described by N image coordinates $s_n = [x_n, y_n], n \in N$ with N being the number of landmarks. We call the individual sets of image coordinates the sub-agents. The input to the neural network consists of $N \cdot C$ image patches, with C being a multiplier for the image pyramid, which in most scenarios equals 1, but in the case of normal image pyramid as shown in Figure 4.4 equals the number of pyramid levels. The network predicts the action in matrix form as in Equation 4.16, where each column represents the movement in x- and y-direction for one landmark and is applied to the corresponding sub-agent. The setup is shown in Figure 4.10.

4.3.3.1 Learning spatial relations between landmarks

This method has one major advantage over the previous approach. Because the method uses a single agent as a centralized controller, we can incorporate inter-agent communication between the individual sub-agents by adding relative positions between the sub-agents to the neural network input. This makes it possible to learn spatial correlations of the landmarks, which during inference helps to detect occluded or missing landmarks because the landmark's position is often determined by the position of other landmarks. The relative positions are represented in form of two pairwise difference matrices $\mathbf{D}_x$ and $\mathbf{D}_y$. The pairwise difference matrix is created by computing the pairwise difference vector between

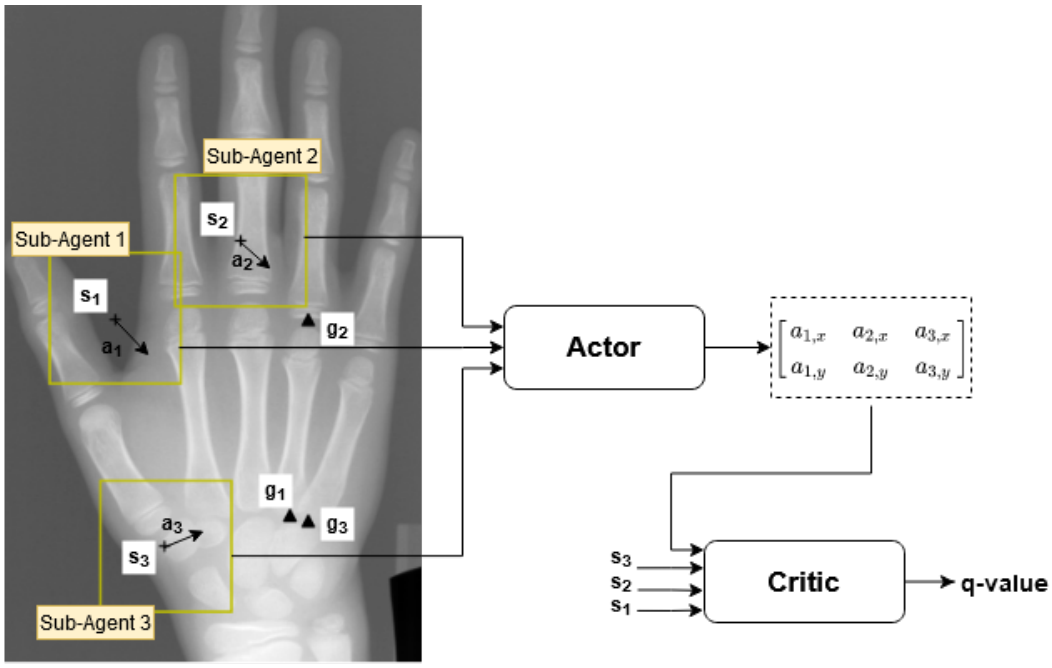


Figure 4.10: Setup of landmark localization with SAL. The observations of N sub-agents are fed to the actor, which predicts a translation vector a_n for each sub-agent. The target landmark is denoted as g_n. The predicted matrix is fed to the critic along with the observations from the sub-agents to predict a single Q-value.

all sub-agents positions, resulting in a 2D vector for every possible combination of two sub-agents. The x and y component of these vectors are entered in the two matrices $\mathbf{D}_x$ and $\mathbf{D}_y$ respectively, such that $\mathbf{D} = \sqrt{\mathbf{D}_x + \mathbf{D}_y}$, where $\mathbf{D}$ is the pairwise distance matrix containing the euclidean distances.

$$\mathbf{D}_x = \begin{bmatrix} 0 & d_{x_1,x_2} & \dots & d_{x_1,x_N} \\ d_{x_2,x_1} & 0 & & \\ \vdots & & \ddots & d_{x_{N-1},x_N} \\ d_{x_N,x_1} & & d_{x_N,x_{N-1}} & 0 \end{bmatrix} \qquad d_{x_i,x_j} = p_{x,i} - p_{x,j} \tag{4.22}$$

Equation 4.22 shows matrix $\mathbf{D}_x$ with $p_{x,i}$ being the x-coordinate of sub-agent i. Matrix $\mathbf{D_y}$ is created equivalently. The entries from the two matrices are flattened and concatenated to a single vector and fed to the first dense layer of the neural network as shown in Figure 4.11.

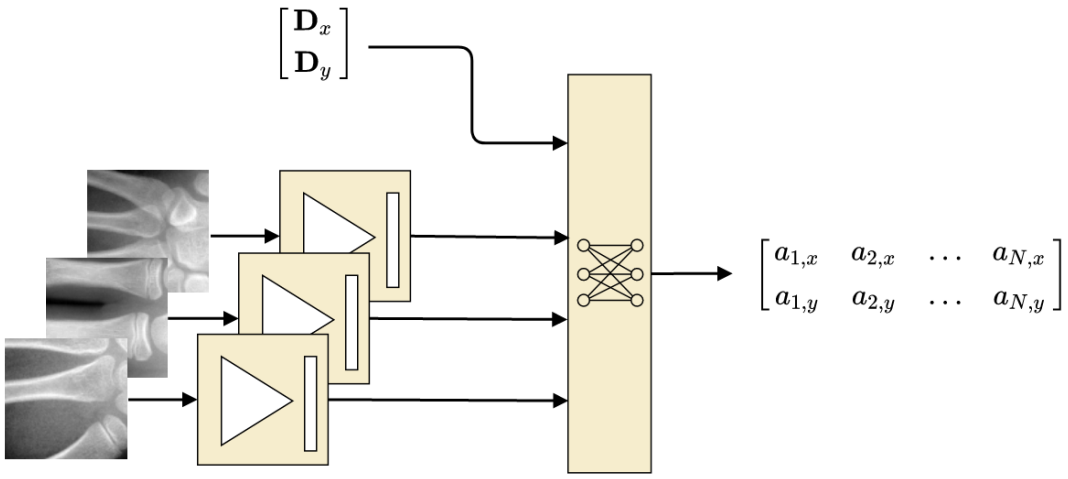


Figure 4.11: Architecture of SAL with pairwise difference matrices as input to the first dense layer to integrate spatial relations between the sub-agents. the individual observations from the sub-agents are passed to a conv-block and then flattened and concatenated at the first dense layer. The same architecture is used in the standard SAL algorithm but without the pairwise difference matrices.

5 Experimental Setup and Results

Contents

5.1 Dataset

We evaluate the algorithms on a publicly available dataset of X-ray images from hands [59]. The dataset consists of 895 images, which have been acquired with multiple different scanners and with different settings, which results in image resolution and intensity variations. The images were annotated by an expert with 37 characteristic landmarks that include mainly bone joints. Since the images have different resolutions at an average image size of 1563×2169, the resolution has to be converted into the physical resolution to obtain expressive results. The physical resolution is defined as the actual physical length of a feature in mm divided by the pixels spanning this feature ($\frac{mm}{pixel}$). However, the images do not contain information about the physical resolution, therefore, following [60], the physical resolution is calculated by assuming a wrist width of $50mm$, which is defined by two annotated landmarks and dividing by the number of pixels that span this distance. The two landmarks used for this calculation are shown in Figure 5.1. To ease computation, the images were preprocessed by downsampling them to a common long-axis size of 512 pixels. The data set is split in 80:20 fashion such that 716 images are used for training and 179 images for testing.

Figure 5.1: The two landmarks indicated as red dots are used to calculate the physical resolution of the image, by assuming a distance of 50mm between these two landmarks.

5.2 Resources

The experiments were conducted on three different workstations with Nvidia® GPUs. The code was written in Python™ 3.7. For neural network optimization, the framework TensorFlow 2.1 [1] was used. See Table 5.1 for specifications of the individual workstations.

	Workstation		
	1	2	3
CPU	Intel Core i5-4570 3.2 GHz	Intel Core i7-3770 3.4 GHz	Intel Core i7-8700 3.2 GHz
GPU	Nvidia Geforce GTX Titan X	Nvidia Titan V	Nvidia GeForce RTX 2080 Ti
RAM	24 GB	32 GB	64 GB
VRAM	12 GB	12 GB	11 GB

Table 5.1: Workstation specifications.

5.3 Training Parameters

The parameter **training episodes** describes how many passes of the agent-environment loop are played during training. Since the training dataset contains 716 images, one epoch would correspond to 716 episodes. Termination of the episode occurs if the agent localizes the landmark within a certain radius of tolerance or if the **maximum number of steps** is reached, which is another training parameter. The radius is set to one pixel for all experiments, which means, the agent is allowed to have a tolerance of one pixel in the four principal directions. The parameter **total steps** is a consequence of training success since one can not know in advance how many steps are needed for one episode. It is

bounded however by *training episodes* × *maximum number of steps*. A detailed list of all algorithm specific hyperparameters can be found in the appendix. Please note that all experiments have been trained until the training curve completely flattened out to ensure a fair comparison between experiments, except in cases where it is explicitly mentioned.

5.4 Evaluation Metrics

To ensure deterministic results for all experiments, we place the agent in the center of the image, instead of placing it randomly like during training. We use the euclidean distance between the agent and the target landmark as a measure of distance and convert it to the physical distance in *mm* as explained in Section 5.1. This metric is referred to as the **distance error**. In the results tables, we use the abbreviation (*Err.* ± *SD*) for the average distance error, with *SD* being the standard deviation. We also determine the **average number of steps** (*Avg. steps*) the agent needs to localize the target landmark. The agent is limited to the same number of steps as during training. Consequently, we calculate the **termination rate** as the percentage of images, where the agent terminates by the termination criterion (e.g. oscillation in DQN) and not by reaching the maximum number of steps. The **success rate** is defined as the percentage of images, where the agent finds the landmark within a radius of 5mm. We also monitored training time with the **average time per training step** being the most significant metric. All time metrics were measured on Workstation 2 from Table 5.1, even if training was executed on another workstation, to obtain comparable measurements. **Complete training time** is calculated as the average time per training step times the number of **total steps**. We visualize the results as the cumulative distance error and as a spread image, where we scatter the error vector of the individual test images on a symbolic image. The error vector is defined as the vector from the target landmark to the agent's localization.

5.5 Experiments for Single-Landmark Localization

5.5.1 Baseline Experiment: DQN

This section explains the experimental setup of the Deep Q-Network (DQN) algorithm for landmark localization [33] as explained in Section 4.1. This experiment is used as a baseline. We use the carpometacarpal joint of the pinky finger as a landmark, which is visualized in Figure 5.2. The most important training parameters of the experiment are listed in Table 5.2. The *DQN*-agent is terminated if oscillations occur. Oscillations are detected if the agent oscillates between two pixels or circulates between four pixels. Furthermore, we implement a multi-scale framework [3] as explained in Section 4.1.1 to reduce the number of steps needed to reach the target landmark.

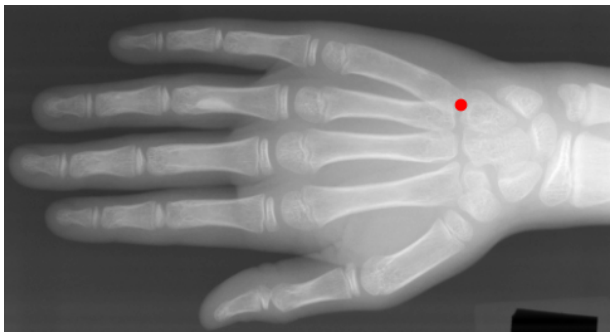


Figure 5.2: The carpometacarpal joint of the pinky finger.

Parameter	DQN	DQN multi-scale
Episodes	20000	20000
Maximum number of steps	250	250
Total Steps	3,670,000	1,660,100
State representation	embedded image pyramid	embedded image pyramid
Pyramid levels	2	2
Feature size	96x96	96x96
Network parameters	10,584,900	10,584,900
Average time per training step	75ms	75ms
Complete training time	76h	35h

Table 5.2: Training parameters of the DQN based algorithm for landmark localization.

5.5.2 Single Landmark Localization with DDPG

This section explains the experimental setup of the Deep Deterministic Policy Gradients (DDPG) algorithm for landmark localization as explained in Section 4.2. Again, we use the carpometacarpal joint of the pinky finger as a landmark. Training parameters are shown in Table 5.3. The termination criterion is fulfilled if the agent's predicted action is lower than 0.5 pixels for both x and y value. In some cases, however, oscillations around the target landmark occur. In this case, the agent is also terminated and as the agents final location the interpolated value between the two or three points of oscillation is determined. We compare the results with the baseline *DQN* approach and a state-of-the-art U-Net based approach [60] using heatmap regression in Table 5.4.

Parameter	Value
Episodes	40000
Maximum number of steps	100
Total Steps	2,292,000
State representation	embedded image pyramid
Pyramid levels	2
Number of network parameters	10,584,386
Average time per training step	195ms
Complete training time	124h

Table 5.3: Training parameters of the DDPG based algorithm for landmark localization.

5.5.3 Results: DQN and DDPG for Single-Landmark Localization

Figure 5.3 shows the cumulative distance error of the *DQN* and the *DDPG* algorithm for landmark localization. We compare the results to the state-of-the-art supervised U-Net based approach from [60]. Table 5.4 shows a detailed comparison of the results of the different algorithms and Figure 5.4, 5.5 and 5.6 show the spread of the individual experiments.

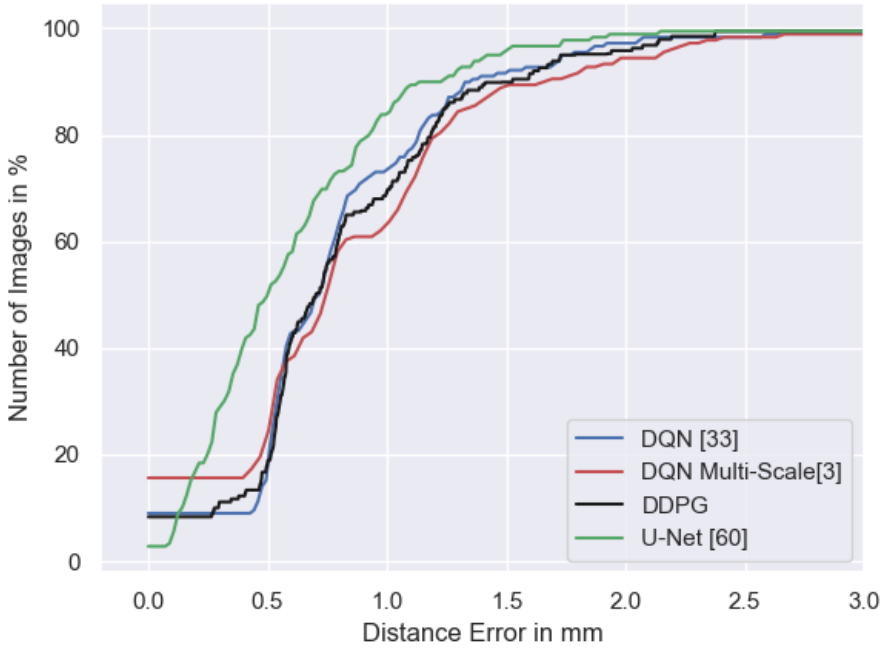


Figure 5.3: Comparing the cumulative distance error of Reinforcement Learning (RL) methods with the state-of-the-art U-Net based approach using heatmap regression [60]. The graph shows, that *RL* methods are able to yield comparable results for single landmarks.

	Reinforcement learning			Supervised learning
Parameter	**DQN** [33]	**DQN multi-scale** [3]	**DDPG**	**U-Net** [60]
Err. ± SD (mm)	0.827 ± 0.54	0.896 ± 0.75	0.819 ± 0.53	**0.62 ± 0.46**
Avg. steps	177.5	30.9	**5.85**	-
Termination rate	100%	100%	100%	-
Success rate	100%	99.4%	100%	-

Table 5.4: Results of the DDPG algorithm for landmark localization for a single landmark. We compare the results to the baseline DQN approach with and without multi-scale framework and a state-of-the-art supervised method based on U-Net [60].

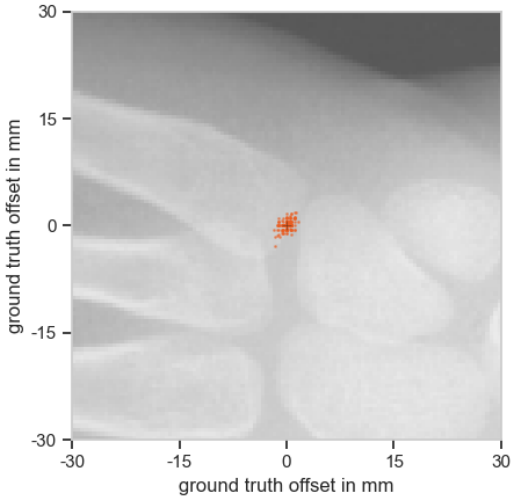


Figure 5.4: Spread of the *DQN* algorithm trained on a single landmark.

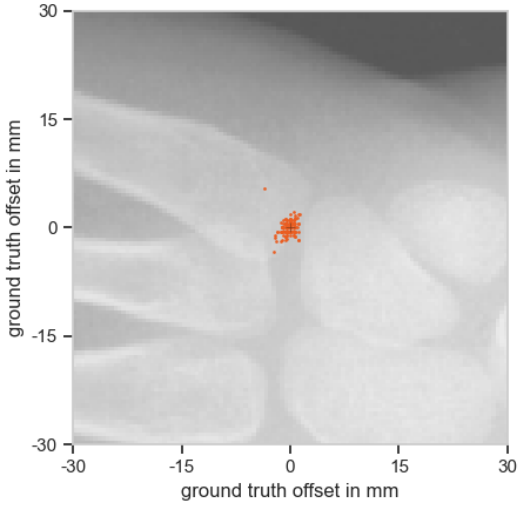


Figure 5.5: Spread of the *DQN* algorithm with multi-scale framework.

5.5.4 Ablation Study 1: Comparison of State Representations

With this experiment, we evaluate different state representations which have been described in Section 4.1.3. We use the DDPG algorithm and train on a single landmark. We chose the distal interphalangeal joint of the ring finger as a landmark, which is the foremost joint in the finger. The landmark is rather indistinguishable from the same joint in the other fingers, therefore global information is required. Paired with a feature size of 64×64 it perfectly reproduces the problems explained in Section 4.1.3. Figure 5.11

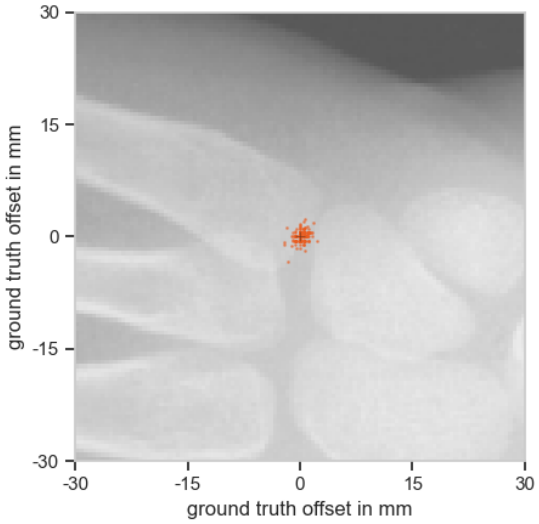


Figure 5.6: Spread of the *DDPG* algorithm for single landmark localization

shows the cumulative distribution for all experiments. All experiments were conducted with 15000 episodes and with a maximum number of steps of 100.

Image Patch

The image patch is the simplest of all state representations and is the cropped region of size 64×64 pixels centered around the agent's position. In many cases, this kind of state representation might result in ambiguous states because the Field of View (FoV) is not large enough to cover global information. This means, that the agent might have difficulties to distinguish between certain landmarks.

Image Pyramid as Conv-Paths

Using an image pyramid tackles the aforementioned problem by including a second image patch of the same size, but with twice the *FoV*. The input to the neural network consequently doubles. An example of an observation is shown in Figure 5.10. We feed the individual levels of the image pyramid as different convolution paths to the neural network. At the first dense layer, the features are then combined as shown in Figure 5.7. This method increases the number of parameters by a significant amount.

Image Pyramid as Color Channels

In this experiment, we are using the same pyramid as in the previous paragraph, but instead of feeding the individual pyramid levels to different convolution paths, we stack the images on top of each other, similar to the channels of an RGB image. This method allows reducing the number of parameters.

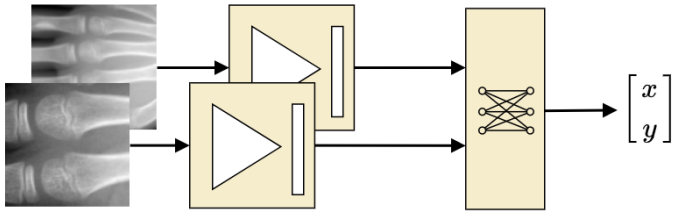


Figure 5.7: Feeding the image pyramid to individual convolution paths of the Convolutional Neural Network (CNN).

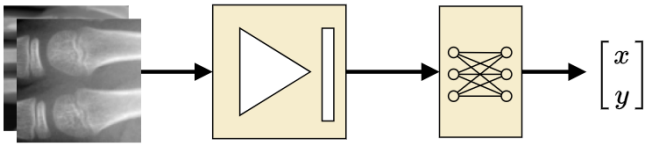


Figure 5.8: Representing the image pyramid as color channels of a single image results in a single convolution path and reduces the number of parameters of the *CNN*.

Embedded Image Pyramid

The embedded image pyramid combines the two observations of the standard image pyramid into a single image to reduce the input size of the neural network. It has to be considered, that the number of parameters is dependent on the input size, therefore the network with the image pyramid as conv-paths has approximately twice the number of parameters, which makes it slower to converge.

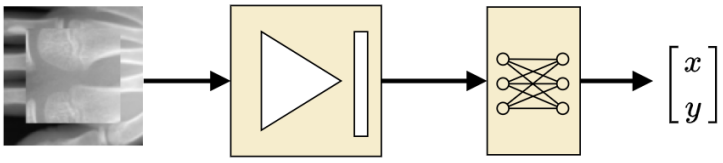


Figure 5.9: Merging the pyramid levels into a single image.

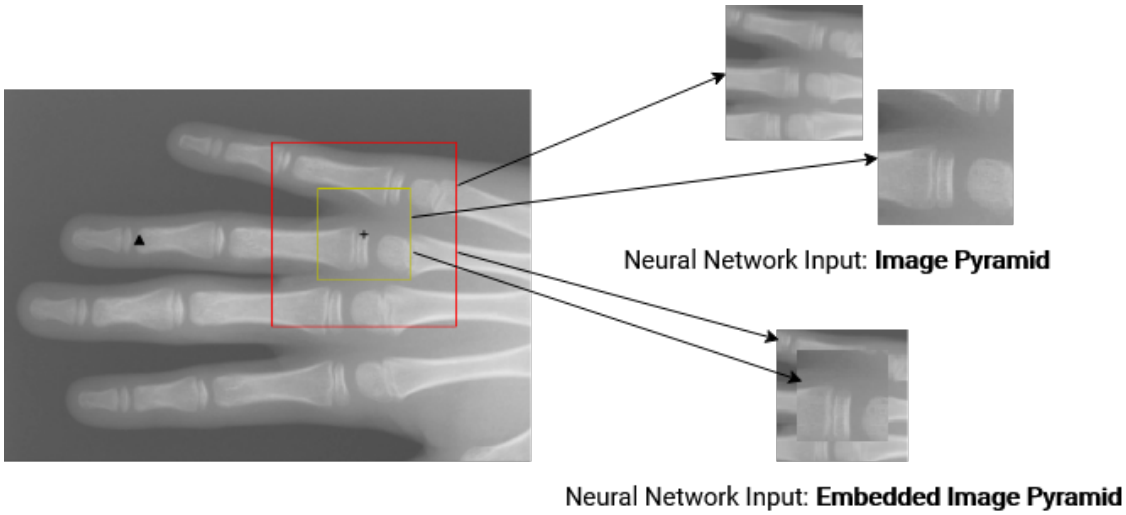


Figure 5.10: An example of an image pyramid with two levels. The two image patches of the image pyramid are fed into the neural network as a single observation. By adding higher pyramid levels we can increase the FoV. The yellow square box shows the first pyramid level, which covers a FoV of 64×64 pixels and the red square box the second level with a FoV of 128×128 pixels. the black triangle indicates the target landmark.

Results

This section shows the results of the *DDPG* algorithm for landmark localization for different state representations as explained in the previous section.

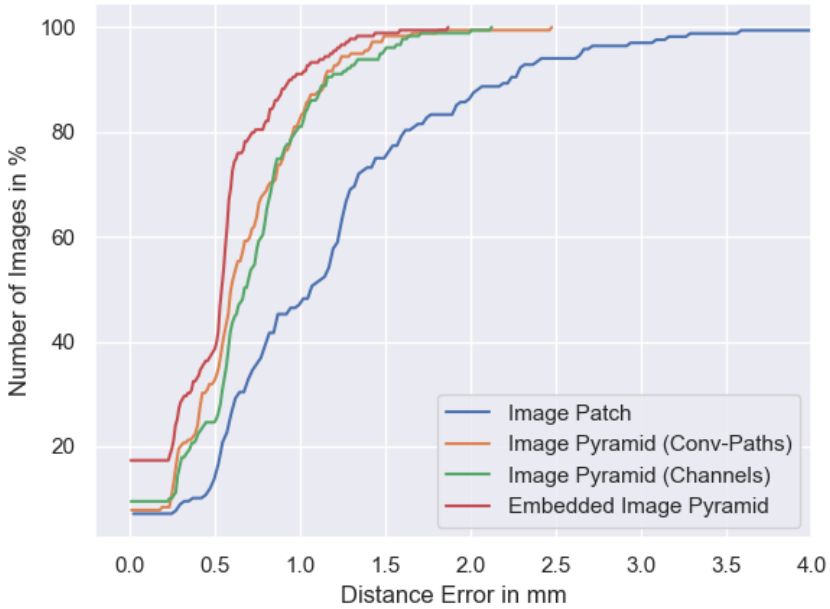


Figure 5.11: Cumulative distribution of the distance error for different state representations.

Parameter	Image Patch	Image Pyramid as Conv-Paths	Image Pyramid as Channels	Embedded Image Pyramid
Average distance error (mm)	2.64	0.66	0.69	**0.52**
Standard deviation	7.19	0.39	0.4	**0.35**
Average number of steps	12.4	11.6	10.8	**8.26**
Termination rate	97.8%	98.3%	98.9%	100%
Success rate	94.4%	100%	100%	100%
Network parameters	5,341,506	10,550,658	5,341,794	5,341,506
Average time per training step	**118ms**	206ms	150ms	144ms

Table 5.5: Comparing the influence of different state representations, trained on the same landmark, with the same training parameters. We use the *DDPG* algorithm for this experiment. One can clearly see, that without global information as in the simple image patch, the agent has difficulty to distinguish the individual fingers. This can be derived from the fact, that the results have a high standard deviation and that the success rate is lower than the termination rate, which indicates, that the agent is terminating at the wrong finger.

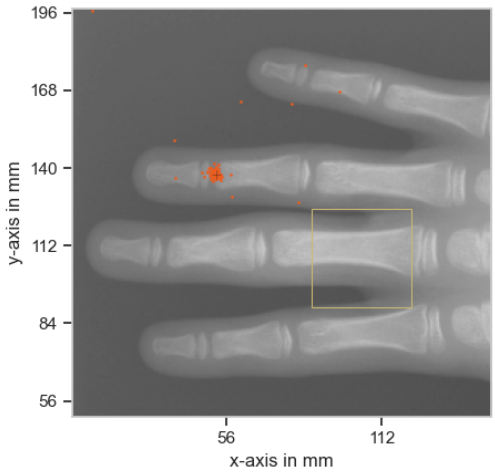


Figure 5.12: Spread around the target landmark of the DDPG algorithm applied to a single landmark. The agent's observation is an image patch of size 64×64. Since the observation only includes local information, the agent in some cases cannot distinguish between the individual fingers. The yellow square box shows an example of an agent's observation.

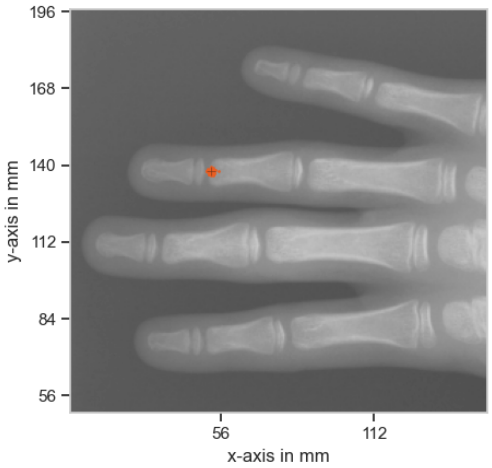


Figure 5.13: Spread around the target landmark with an image pyramid of size 64×64 as input, where the individual pyramid levels are fed to different convolution paths of the neural network.

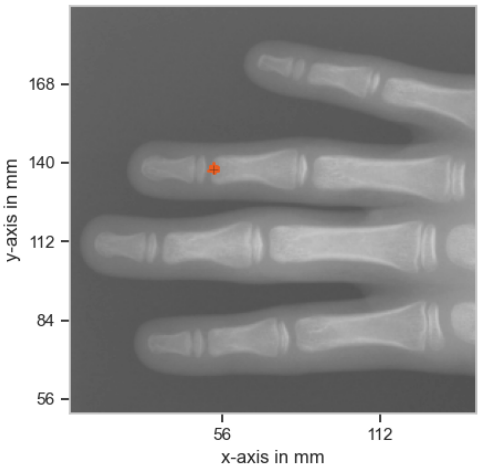


Figure 5.14: Spread around the target landmark with an image pyramid of size 64×64 as state representation. The image pyramid is fed to the neural network as individual channels of a single image.

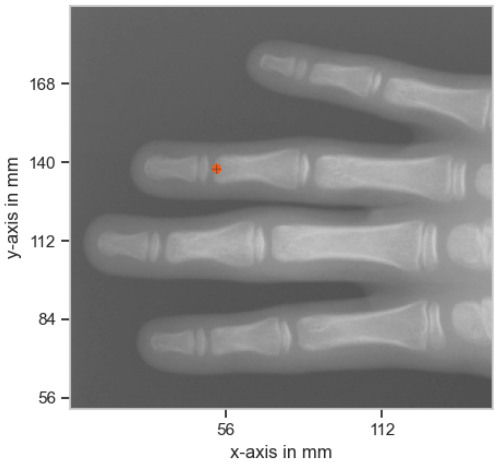


Figure 5.15: Spread around the target landmark. The agent's observation is an embedded image pyramid with a size of 64×64 pixels.

5.5.5 Ablation Study 2: Random Initialization

We further evaluate the stability of the algorithm regarding the agent's starting position. We evaluate the *DDPG* algorithm for landmark localization on the entire test set 10 times with random starting position of the agent, such that each image in the dataset has been evaluated 10 times with different starting positions.

Results

Table 5.6 shows the results of Ablation Study 2.

Parameter	**DDPG** center initialization	**DDPG** random initialization
Err. ± SD	0.819 ± 0.53	0.824 ± 0.53
Avg. steps	5.85	7.22
Termination rate	100%	100%
Success rate	100%	100%

Table 5.6: Evaluating stability of the DDPG algorithm for landmark localization with regard to the agents starting position. We evaluate the entire test dataset 10 times with random starting position. As a comparison, the results from the previous section are shown, where the agent has been initialized in the center of the image.

5.6 Experiments for Multi-Landmark Localization

5.6.1 Multi-Task DQN

In this experiment, we evaluate the multi-task approach for the *DQN* algorithm as explained in Section 4.3.1. Table 5.7 shows the training parameters of this experiment. We use the multi-scale framework from section 4.1.1 to obtain reasonable performance in terms of inference and training speed. Different landmarks introduce different problems: the landmarks in the fingers are ambiguous and the landmarks in the hand-bones have higher variance. To cover the challenges of different landmarks, we carefully select five landmarks which are shown in Figure 5.16.

Parameter	Value
Episodes	25000
Maximum number of steps	150
Total Steps	3.503.325
State representation	embedded image pyramid
Pyramid levels	2
Network parameters	10,589,012
Average time per training step	75ms
Complete training time	73h

Table 5.7: Experiment parameters of the *DQN* based algorithm for landmark localization.

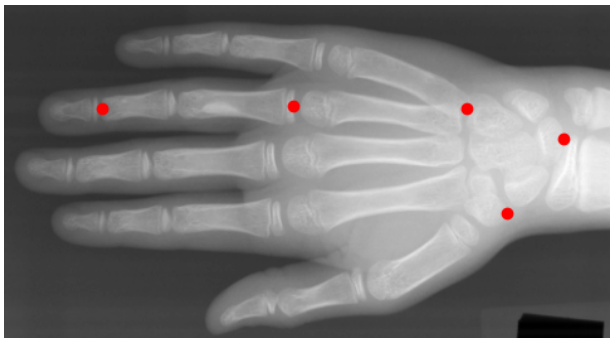


Figure 5.16: Selected landmarks for multi-landmark localization. The landmarks in the fingers are ambiguous and the landmarks in the hand-bones have higher variance. We use these landmarks for all multi-landmark experiments.

Results

Figure 5.17 shows the cumulative distance error of the *DQN* algorithm for landmark localization on the individual landmarks. Numeric results are listed in Table 5.8 and the

spread at the individual landmarks is shown in Figure 5.18.

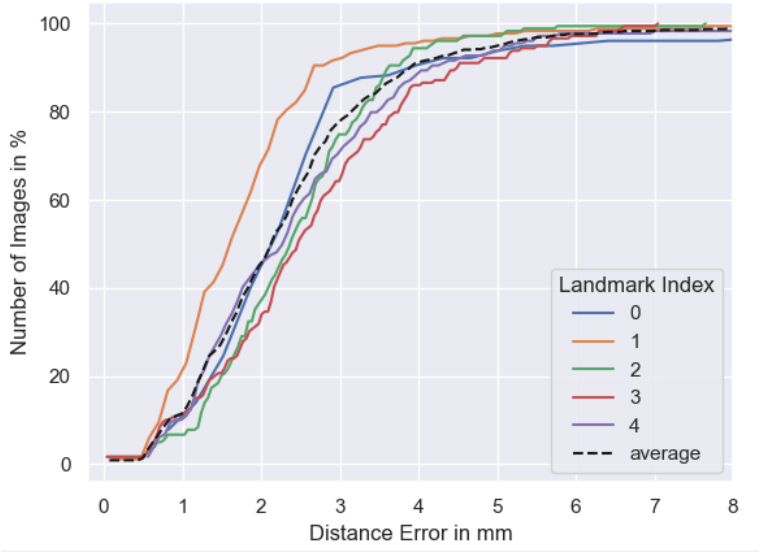


Figure 5.17: Results of the multi-task *DQN* algorithms trained on 5 landmarks. The plot shows the cumulative distribution of the distance error in mm. The landmark index is related to Figure 5.18

Parameter	**LM 0**	**LM 1**	**LM 2**	**LM 3**	**LM** 4	**Total**
Err ± SD (mm)	3.1 ± 6.45	1.84 ± 1.76	2.41 ± 1.42	2.64 ± 1.63	2.5 ± 1.63	2.5 ± 3.2
Avg steps	41	26	32.5	41.2	48.5	37.9
Termination rate	99.4%	99.4%	100%	100%	99.4%	99.7%
Success rate	93.9%	97.8%	97.2%	92.2%	93.9%	95%

Table 5.8: Experiment results of the multi-task DQN algorithm for landmark localization.

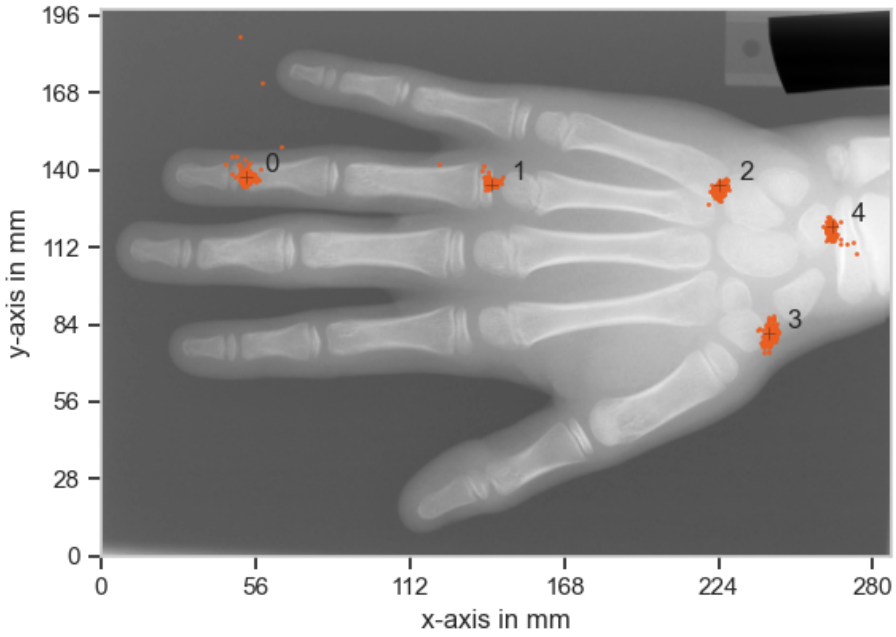


Figure 5.18: Spread of the multi-task *DQN* algorithm.

5.6.2 Multi-Task DDPG

This section explains the experimental setup of the multi-task framework presented in Section 4.3.2. The algorithm was trained on the five landmarks of Figure 5.16 with the parameters of Table 5.9. It is noticeable, that the training time per batch is almost identical to the single landmark case, shown in Table 5.3.

Parameter	Value
Episodes	40000
Maximum number of steps	100
Total Steps	2414000
State representation	embedded image pyramid
Pyramid levels	2
Network parameters	10,586,442
Average time per training step	210ms
Complete training time	141h

Table 5.9: Training parameters of the DDPG based algorithm for landmark localization.

Results

Figure 5.19 shows the cumulative distance error of the *DDPG* algorithm for landmark localization on the individual landmarks. Numeric results are listed in Table 5.10 and the spread at the individual landmarks is shown in Figure 5.20.

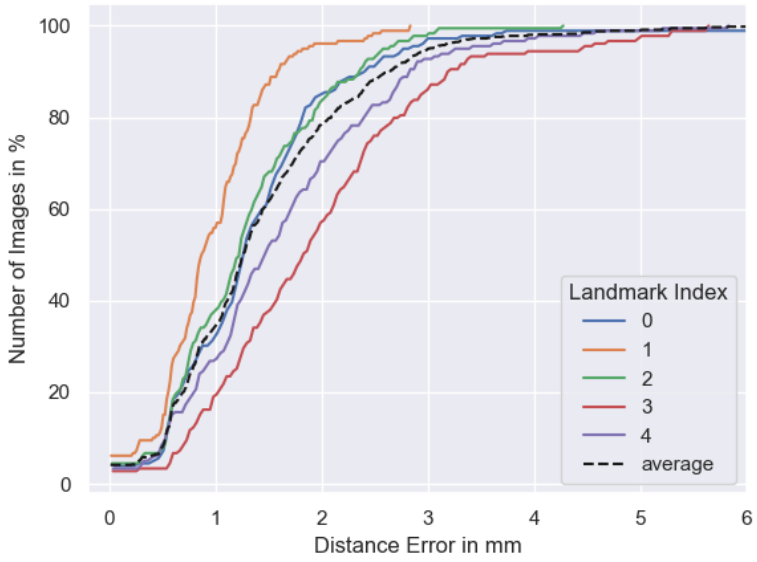


Figure 5.19: Results of the multi-task DDPG algorithms trained on 5 landmarks. The plot shows the cumulative distribution of the distance error in mm. The landmark index is related to Figure 5.20

Parameter	LM 0	LM 1	LM 2	LM 3	LM 4	Total
Err. ± SD (mm)	1.39 ± 1.01	0.96 ± 0.53	1.28 ± 0.73	1.95 ± 1.1	1.62 ± 0.99	1.44 ± 0.96
Avg. steps	7.46	5.77	7.07	9.17	9.01	7.69
Termination rate	100%	100%	100%	99.4%	100%	99.9%
Success rate	98.9%	100%	100%	97.2%	98.9%	99%

Table 5.10: Experiment results of the multi-task DDPG algorithm for landmark localization. Note that the average number of steps depends on the position of the landmark (LM), since the agent always starts at the center of the image, landmarks further away from the center also require more steps.

5.6.3 Multi-Landmark Localization with SAL

This section explains the experimental setup of the Simultaneous Acting and Localizing (SAL) algorithm from Section 4.3.3. The algorithm was evaluated on the same five landmarks as the multi-task experiments from the previous sections. As one can see in Table

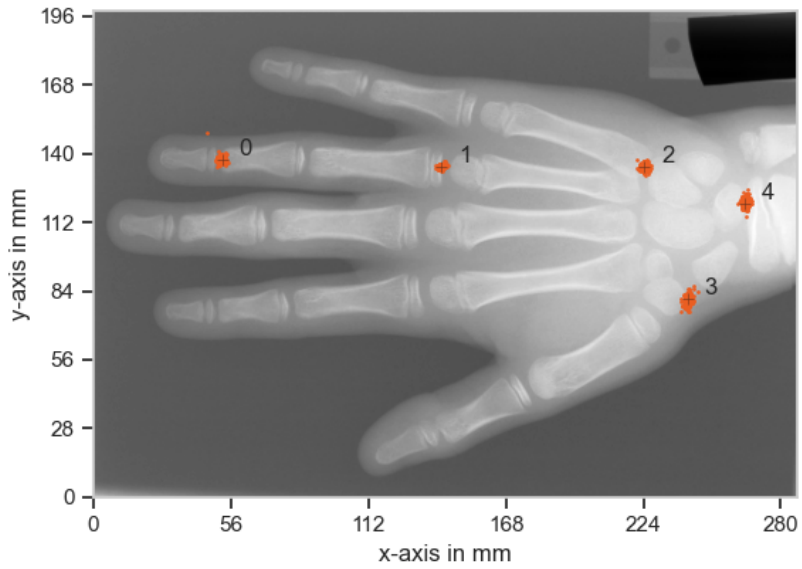


Figure 5.20: Spread of the multi-task *DDPG* algorithm.

5.11, the training time is considerably longer than in previous experiments, therefore the algorithm was only trained on 20000 episodes, even though the learning curve still showed a potential increase in accuracy. We also evaluate the *SAL* approach including spatial configurations from Section 4.3.3.1, with the same training parameters. We refer to this method as Simultaneous Acting and Localizing with Spatial Configuration (SAL-SC).

Parameter	Value
Episodes	20.000
Maximum number of steps	100
Total Steps	2.000.000
State representation	embedded image pyramid
Pyramid levels	2
Network parameters	17.005.130
Average time per training step	395ms
Complete training time	219.4h

Table 5.11: Experiment parameters of the *SAL*/*SAL-SC* algorithm for landmark localization.

Results

This section shows the results of the *SAL* algorithm and the *SAL-SC* algorithm. The experimental setup is explained in Section 5.6.5.

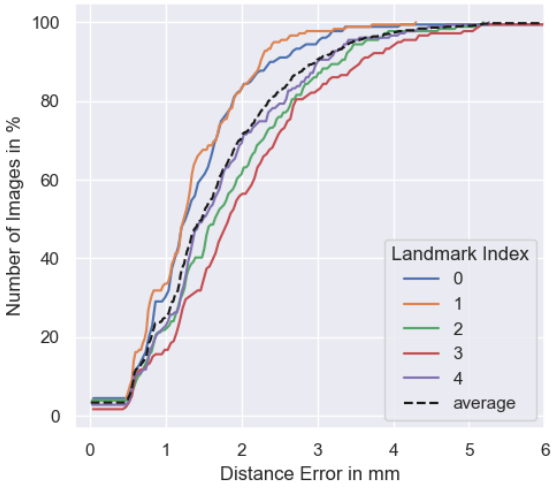


Figure 5.21: Results of the *SAL* algorithm trained on 5 landmarks. The plot shows the cumulative distribution of the distance error in mm. The landmark index is related to Figure 5.23

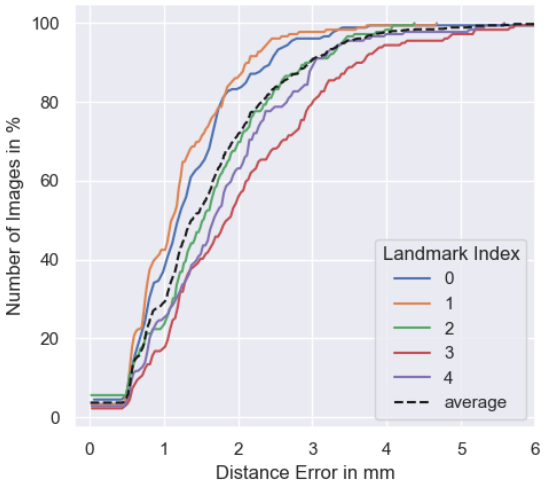


Figure 5.22: Cumulative error distribution of the *SAL-SC* algorithm trained on 5 landmarks. The landmark index is related to Figure 5.24

Parameter	LM 0	LM 1	LM 2	LM 3	LM 4	Total
Err ± SD (mm)	1.41 ± 0.88	1.32 ± 0.71	1.82 ± 1.03	2.02 ± 1.14	1.71 ± 0.97	1.66 ± 0.99
Avg. steps	-	-	-	-	-	31.8
Termination rate	-	-	-	-	-	83.8 %
Success rate	99.4%	100%	98.9%	97.7%	99.4%	99.1%

Table 5.12: Experiment results of the *SAL* algorithm for landmark (LM) localization.

Parameter	LM 0	LM 1	LM 2	LM 3	LM 4	Total
Err ± SD (mm)	1.35 ± 1.03	1.22 ± 0.72	1.63 ± 0.94	2.04 ± 1.2	1.78 ± 1.03	1.6 ± 1.04
Avg. steps	-	-	-	-	-	27.7
Termination rate	-	-	-	-	-	86.6%
Success rate	99.4%	100%	100%	97.2%	97.8%	98.9%

Table 5.13: Experiment results of the *SAL-SC* algorithm for landmark (LM) localizationn.

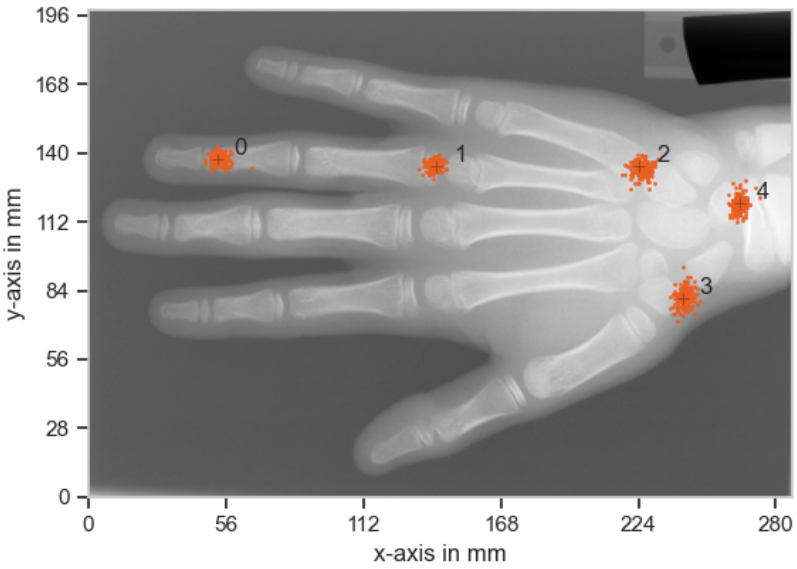


Figure 5.23: Spread of the *SAL* algorithm trained on five landmarks.

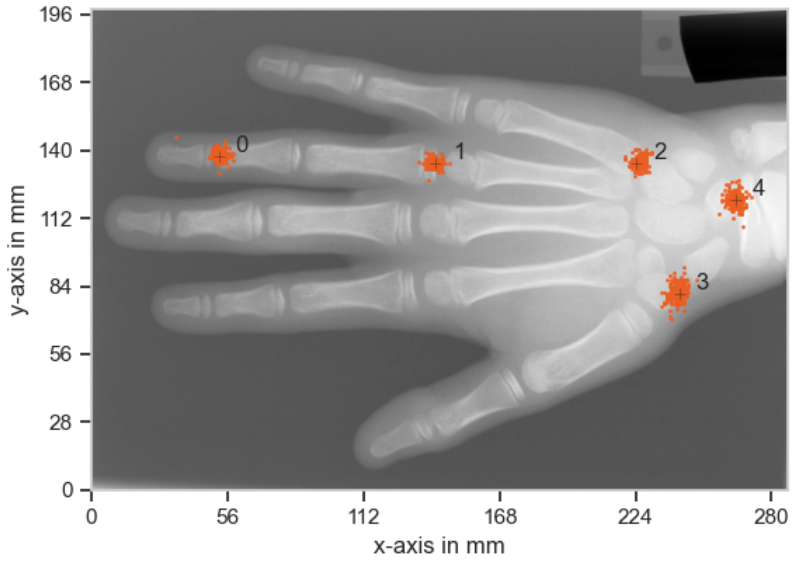


Figure 5.24: Spread of the *SAL-SC* algorithm trained on five landmarks.

5.6.4 Combined Results: Multi-Landmark Localization

This section compares our proposed algorithms for multi-landmark localization with the state-of-the-art approach SpatialConfiguration-Net (SCN) [60] and a U-Net based approach for landmark localization [60]. Figure 5.25 and 5.26 show the cumulative distance error of the multi-landmark algorithms. Numeric results are shown in Table 5.14.

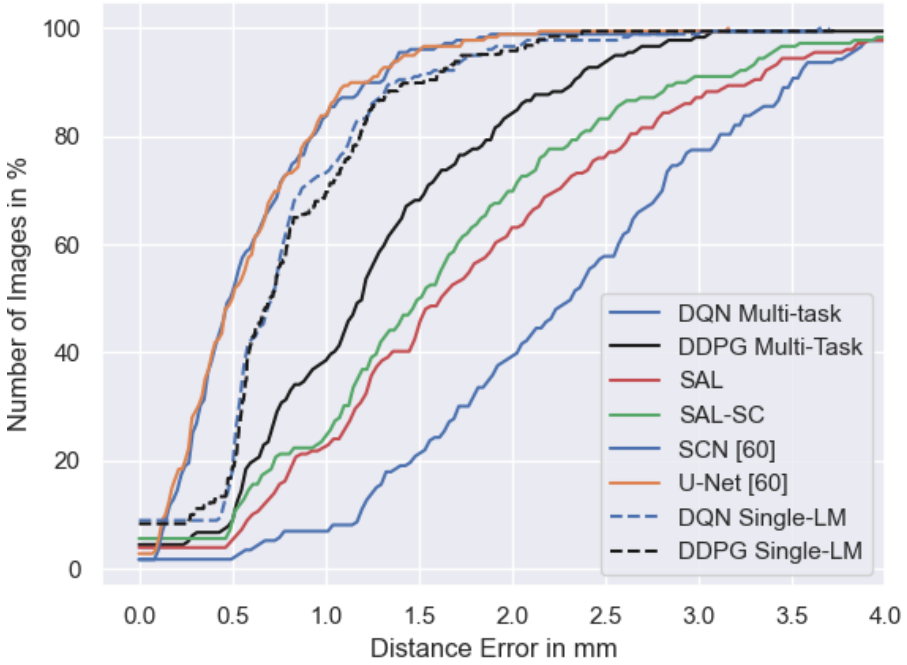
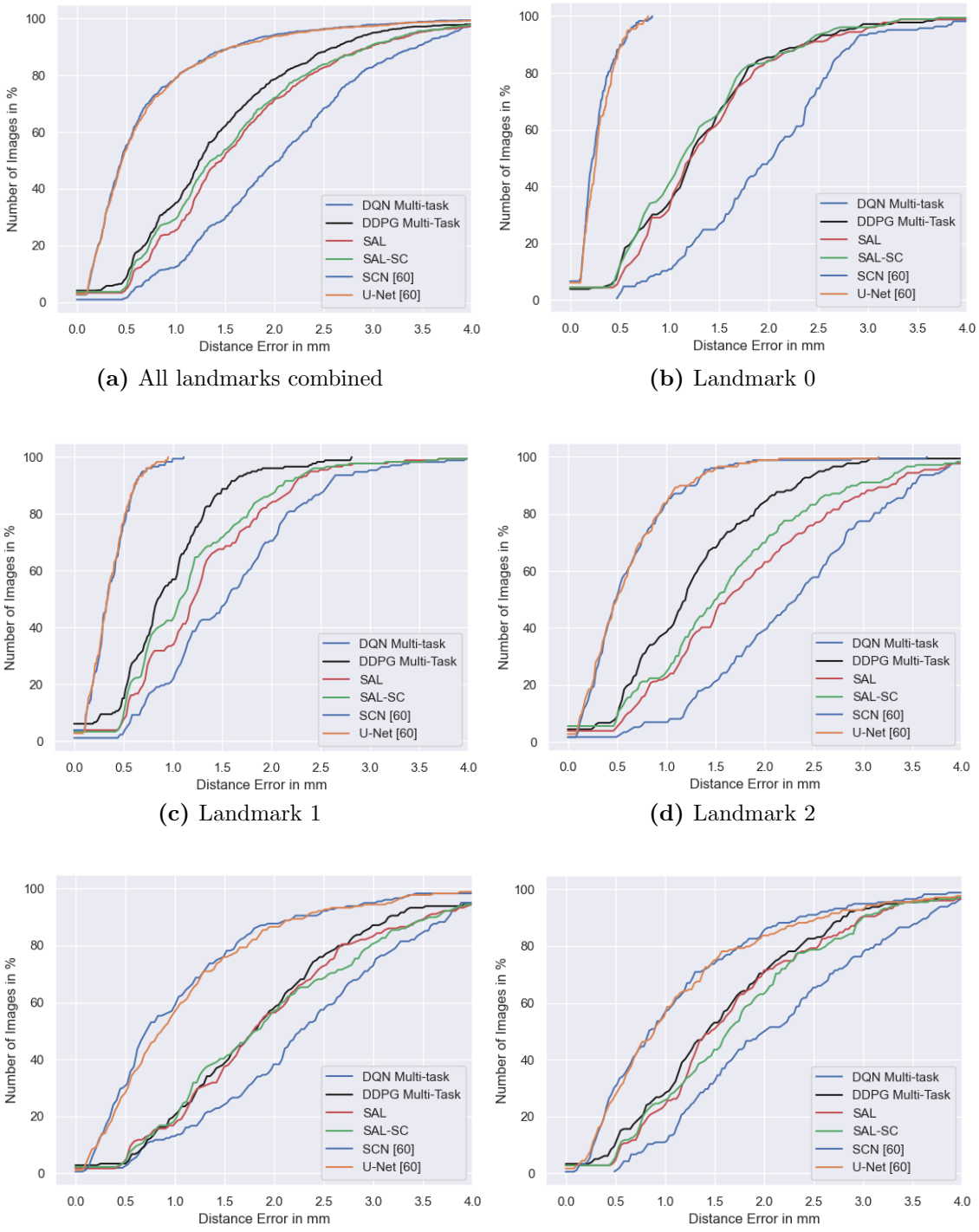


Figure 5.25: Comparing results of methods for single landmark localization to the results of the same landmark in multi-landmark methods, which corresponds to Landmark 2 of Figure 5.26.



(a) All landmarks combined

(b) Landmark 0

(c) Landmark 1

(d) Landmark 2

(e) Landmark 3

(f) Landmark 4

Figure 5.26: Landmark-wise results for different algorithms

	Reinforcement learning			
Parameter	**DQN multi-task**	**DDPG multi-task**	***SAL***	***SAL-SC***
Err ± SD (mm)	2.5 ± 3.2	1.44 ± 0.95	1.65 ± 0.99	1.6 ± 1.04
Avg. steps	37.9	**7.7**	31.1	27.7
	Supervised learning			
Parameter	***SCN*** [60]	**U-Net** [60]		
Err ± SD (mm)	**0.7 ± 0.73**	0.72 ± 0.75		

Table 5.14: This table compares the results of our algorithms for multi-landmark localization with state-of-the-art *CNN*-based results. The individual results are averaged over all five landmarks. Per landmark results can be found in the corresponding sections.

5.6.5 Learning Spatial Relations with SAL

The goal of this experiment was to show, how we can use spatial relations between agents to learn a configuration of the landmarks, such that occluded landmarks can be detected as explained in Section 4.3.3.1. We evaluate the algorithm by occluding the left half of the test images. Occluding the left half has the consequence, that one landmark is completely occluded and another one is partially occluded, as shown in Figure 5.27. The occluded areas are replaced by uniform noise, which resembles the distribution of the dataset.

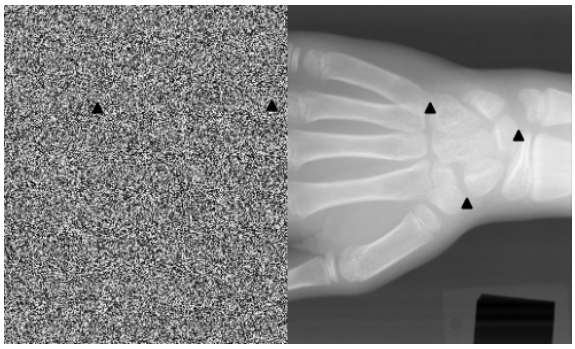


Figure 5.27: A test image with the left half occluded i.e. replaced by uniform noise. The landmark in the middle of the image is considered partially occluded, because the agents observation includes occluded and visible areas of the X-ray image.

Results

Table 5.15 shows the results of the *SAL-SC* algorithm on corrupted images. We compare the algorithm to our other methods for multi-landmark localization, evaluated on the same occluded test images. Figure 5.28 shows qualitative results.

Algorithm	LM 0 (occluded)	LM 1 (partially occluded)	LM 2	LM 3	LM 4	Total
Multi-Task *DQN*	*Failed*	*Failed*	2.34 ± 1.09	2.62 ± 1.5	2.53 ± 1.63	-
Multi-Task *DDPG*	*Failed*	17.9 ± 12.6	**1.57 ± 2.45**	**1.94 ± 1.68**	**1.62 ± 0.99**	-
SAL	53 ± 10.4	101.9 ± 16.1	20.08 ± 6.24	19.2 ± 6.1	18.25 ± 8.2	42.48 ± 34
SAL-SC	**11.44 ± 6.1**	**6.81 ± 3.94**	1.94 ± 1.07	2.69 ± 1.4	2.99 ± 1.23	**5.17 ± 4.92**

Table 5.15: Results of different multi-landmark algorithms, evaluated on corrupted images. We replace the left half of the image with noise, such that one landmark is completely occluded and one is partially occluded as shown in Figure 5.27. This table shows, how different algorithms perform if the image contains occluded landmarks.

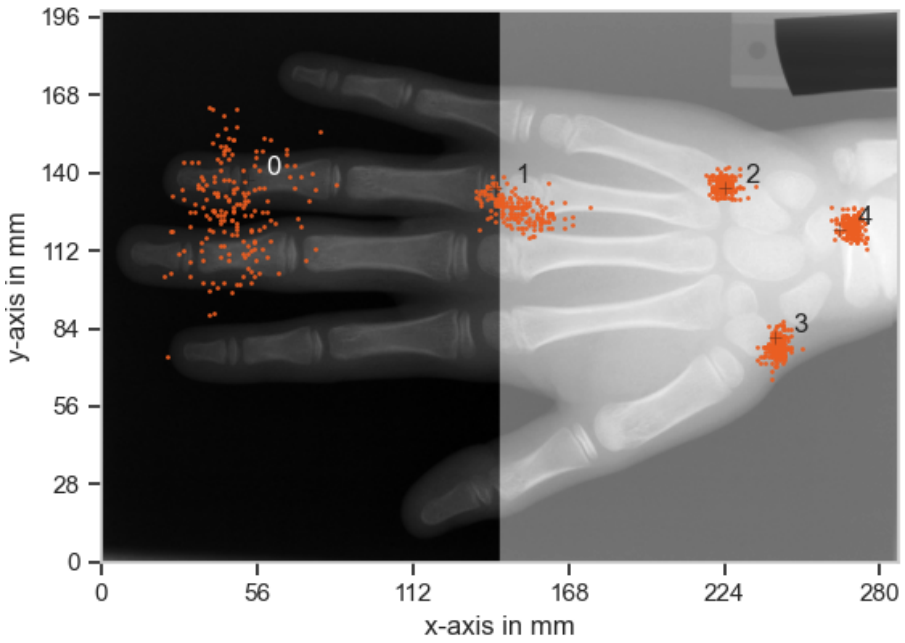


Figure 5.28: Spread of the *SAL-SC* algorithm trained with spatial relations between the agents. The darker left half symbolizes the area of the image that has been replaced with uniform noise during testing.

6 Discussion and Future Work

Contents

6.1 Discussion

In this work, we evaluate the application of Reinforcement Learning (RL) to the problem of landmark localization. *RL* being a patch-based method has some promising properties compared to current state-of-the-art methods, which are mainly based on Convolutional Neural Networks (CNNs). *CNN* based methods are able to achieve high accuracy, however, they suffer from two major limitations. Image-based methods process the image as a whole, which puts high demands on the hardware, especially when working with 3D images. Depending on the resolution and the size of the images it might become impossible to process the data with current hardware, e.g. using whole-body scans with high resolution. In such cases, the data has to be sub-sampled or divided into smaller partitions. Patch-based methods are computationally more efficient, but in many cases require learning an additional model for global guidance. With *RL* we are able to learn an agent, which is able to internally keep a representation of the anatomy, while benefiting in terms of computational requirements from processing image patches. This internal representation is modeled by a so-called Q-function, which tells the agent in every state, which action leads to a more favorable state. This has the consequence, that the agent is able to localize the target landmark from any arbitrary starting position inside the image, which was proven in the experiment of Section 5.5.5. Furthermore, using patch-based methods we can artificially increase the dataset, which addresses a great problem in Medical Image Processing (MIP). Based on the work of Ghesu et al. [33], where for the first time *RL*

was introduced to the field of landmark localization, we develop novel methods, to address specific problems for both single- and multi-landmark localization. The results of our contributions, which are presented in Chapter 5 are discussed in this section.

6.1.1 Single-Landmark Localization

Baseline Experiment

To compare our algorithms, we implemented the Deep Q-Network (DQN) algorithm for landmark localization, first introduced by Ghesu et al. [33]. The experimental setup of the baseline experiments is explained in Section 5.5.1 and the results are shown in Section 5.5.3. This algorithm uses a *DQN* agent with a discrete set of actions, allowing the agent to move in it's four principal directions - up, down, left and right. This limits the actions to a fixed step size, which makes inference slow because the minimal number of steps needed to reach the target landmarks is equivalent to the Manhattan distance from the agent's starting position to the target landmark in pixels. This does not only slow down inference but also training because the agent needs more steps to cover the important areas of the state-space. We know, that it is more important to sample transitions closer to the target to obtain a high accuracy, which is why we use an ϵ-greedy strategy to explore actions closer to the target. With a step size of one, the agent, however, needs more time to reach these important areas during training which slows down training. Therefore, as a second baseline, we also experimented with a multi-scale framework [34], using the approach of Alansary et al. [3], where a single *DQN* agent is shared on all levels of the multi-scale framework. With our experiments, we are able to show, that this approach vastly reduces training time from 76 hours to 35 hours (Table 5.2) and inference time by reducing the average number of steps from 177.5 to 30.9. However, with this approach also a small loss in accuracy has been observed (Table 5.4), which probably results from sharing multiple levels of resolution in a single *CNN*. With our baseline experiments we were not able to reach state-of-the-art performance, however, these *RL* methods benefit from processing only image patches, which allows reducing inference complexity.

Landmark Localization with DDPG

In this experiment we tackle two specific problems that come with the *DQN* algorithm. Since the *DQN* algorithm belongs to the class of value-based methods, it is not possible to directly derive a terminal state of the agent from the agent's prediction, except in the case where an additional trigger action signalizes a terminal state. This additional trigger action increases training complexity, therefore most recent methods have used an oscillating approach, where the agent is terminated as he starts oscillating between two pixels. Using the oscillating approach introduces a source of error because as a terminal state one of the two pixels has to be chosen, which can only be guessed. An alternative is to interpolate between the two final pixels, which automatically introduces error because the

landmark is always defined on a specific pixel. A further and more troublesome limitation is, that *DQN* is very difficult to adapt to large action spaces, which makes the algorithm difficult to generalize for more complex problems. In the case of landmark localization, this means, that the agent is limited to a fixed step size. To adapt a varying step size, we would need to add additional actions, which increases the action space, and makes training more complex or use the multi-scale framework, which has been shown to reduce accuracy.

We use the Deep Deterministic Policy Gradients (DDPG) algorithm, which was designed for continuous action spaces and deterministic behavior of the agent to address the problems of the *DQN* algorithm. Because the *DDPG* algorithm is based on an actor-critic architecture, the actor network directly predicts the action instead of the value function, which can be used to derive a terminal state. By the nature of the algorithm, the agent stops moving if the actor's prediction is below a threshold of 0.5 pixels for both directions, because the agent can only move on discrete image coordinates. This is an ideal termination criterion. Furthermore, by using a continuous action space, the agent is able to move to every coordinate in the cartesian plane with a single step. In our experiments we restricted the step size to 50 pixels for both x- and y-direction, resulting in a 100×100 pixel plane, where the agent can move freely. In cases where efficiency is needed such as real-time landmark localization in dynamic Magnetic Resonance Imaging (MRI) this poses a huge advantage because the number of steps needed to reach the target landmark can be dramatically reduced. Results of this algorithm are shown in Section 5.5.3. In our experiments, the agent was able to localize the target landmark with an average of 5.81 steps, compared to 177.5 with the *DQN* algorithm. The bound of 50 pixels was chosen arbitrarily, which signifies that the number of steps could be further reduced by increasing the allowed step size. Also in Table 5.4 we show, that we achieve the same accuracy as with the *DQN* algorithm. The same results are visualized in Figure 5.3-5.6.

While during inference, the *DDPG* algorithm clearly shows an advantage over the *DQN* algorithm, we encountered some limitations during training. The biggest limitation of the *DDPG* algorithm is the slow improvement during learning phase. While reasonable results are already observed after only a few hundred episodes it takes at least another 30000 episodes for the learning curve to completely flatten out. While *RL* algorithms, in general, tend to converge slowly, the *DDPG* algorithm is even more challenging, because of it's actor-critic architecture, where two neural networks are trained in parallel and the actor network is dependent on the reliable prediction of the critic network. Depending on the hardware, this can translate to several weeks of training. Therefore, we were not able to run the experiments for all the landmarks, therefore, we chose a landmark with intermediate difficulty. As was shown by [60], the landmarks in the wrist are much harder to localize than the ones in the fingers.

The algorithm clearly shows advantages during inference, therefore we investigated methods to speed up training. We expect strong improvements in training speed using mixed-precision training [53]. During our first implementations, however, we did not

observe any speed-up, which is why we stopped investigating mixed-precision training for *RL*. To be able to obtain quick results, inspired by [55] we suggest implementing a framework for parallelizable training of multiple agents.

Ablation Study 1: Comparing State Representations

Goal of the experiment from Section 5.5.4 was to show, how different state representations affect the agent. In the theory chapter, we explain, that the state representation needs to provide enough information to the agent to uniquely describe every state in the Markov Decision Process (MDP). The problem with patch-based methods is, that some landmarks might be ambiguous, like the joints in the fingers. Therefore it is necessary to also provide global information to the agent for disambiguation. We compare the ordinary case of an image patch around the agent's position with three different representations of an image pyramid. These experiments were conducted with a fixed number of episodes of 15000, which means that we did not train until full convergence. We observed that by using an image patch, the agent is not able to distinguish between the individual fingers (Figure 5.12). Table 5.5 shows, that an image pyramid with two levels is able to solve disambiguation, which is also visualized in Figure 5.13-5.15. We compare different ways to represent the image pyramid, either as input to individual conv-paths of the *CNN*, as color channels of a single image or as a combination of different pyramid levels into a single image. We observed that the last method yields best performance, even though some information of the pyramid is thrown away. The method of feeding the images as individual conv-channels is expected to improve accuracy, however, since this method almost doubles the number of parameters, it also substantially increases training time, which is why we did not choose this method. Table 5.5 also shows, that these experiments yield better accuracy, than the experiments of Table 5.4, which is due to the fact that we use a landmark in the fingers, instead of the metacarpal bones, which in general yield better accuracy.

Ablation Study 2: Random Initialization

The experiment of Section 5.5.5 evaluates how stable the *DDPG* algorithms is regarding the agent's starting position. We evaluate stability by running ten episodes per test image with random placement of the agent in the image. We observe (Table 5.6), that accuracy is nearly unchanged and termination/success rate remain at 100%, which indicates that the agent is able to learn a representation of the anatomy to localize the target landmark from any arbitrary starting position inside the image.

6.1.2 Multi-Landmark Localization

Multi-Task DQN

In this experiment from Section 5.6.1, we extended the *DQN* algorithm for landmark localization [33] to the multi-agent case, by adopting a multi-task strategy. Multi-task means, that we train a single agent for multiple landmarks. We chose a multi-task strategy, because of our limited computational resources. With the multi-task strategy, we are able to substantially reduce training time. This is possible because we can compute the reward for each landmark with every transition, which allows to update the Q-function for every landmark with every transition. Table 5.7 shows, that one complete iteration of the agent-environment loop, including a network update, takes the exact same amount of time, that would be needed for a single landmark. However, in Table 5.8 and Figure 5.17 one can see, that we are not able to achieve the same accuracy as in the single-landmark case. Also, increased variance can be observed in Figure 5.18. We also observed that the agent's behavior during training is very unstable, which means after a certain amount of training steps, the optimization process diverges. We conclude that the *DQN* algorithm is not well suited for this task, because it is difficult to adapt the *DQN* algorithm to large action spaces. The multi-task problem is different from just increasing the action space, but since we are using the same neural network, an update of the Q-value of a single state-action-landmark trio, does also affect the Q-values of all other trios. This has the consequence, that oscillations occur at training stage.

Multi-Task DDPG

Similar to the previous approach, we are also extending the *DDPG* algorithm for landmark localization to the multi-landmark case by adopting a multi-task strategy. This method uses an actor-critic architecture. The critic is used to approximate the individual Q-functions for the individual tasks i.e. the landmarks. In our experiments from Section 5.6.2, we are using five landmarks, therefore our critic network has to approximate five Q-functions, as in the multi-task *DQN* method. The difference is in the way these two methods approximate the Q-function. The *DQN* algorithm uses an output neuron for every action-landmark pair, whereas the *DDPG* algorithm only needs one output neuron per landmark. This is possible by using a second network, the actor-network, which predicts a continuous action. This action is then fed as an input to the neural network as shown in Figure 3.5 (right). We observed that this architecture allows much more stable training, however it also slower during training, because we have to train two neural networks simultaneously, where the actor network is dependent on the reliable prediction of the critic network.

With our experiments, we show, that we are able to reliably localize the five chosen landmarks (Table 5.10, Figure 5.19-5.20). The average distance error is larger than for the single-landmark case. It has to be noted, however, that the networks use almost the same

amount of parameters than the networks for single-landmark localization. Consequently also training time per agent-environment interaction is comparable, which increases from 195ms in single-landmark localization to 210ms in the multi-task case. This means, that we are able to train the multi-task agent for five landmarks in almost the same time needed for a single landmark. We assume that similar accuracy is achievable by increasing the complexity of the network. Obviously, this would slow down training, but from experience, we know, that training time does not increase linearly with the number of parameters, which is why we would still expect a decrease in training time with the same accuracy.

Simultaneous Acting and Localizing (SAL)

Inspired from Multi-Agent Reinforcement Learning (MARL), we developed the Simultaneous Acting and Localizing (SAL) algorithm. The experimental setup of this algorithm is explained in Section 5.6.3. The *SAL* algorithm uses multiple agents inside an environment to simultaneously localize multiple landmarks. We are using a single neural network as a controller, therefore, this algorithm does not belong to the class of multi-agent algorithms. Instead, the five agents we use to localize five distinct landmarks are represented as a single agent with joint action-space and joint state-space. As a consequence, our agent has an exponentially larger state- and action-space, which results in much slower convergence during training. During inference, however, this algorithm allows predicting all landmarks simultaneously in the same amount of time needed to localize a single landmark with the *DDPG* algorithm. Since training time is considerably longer with this setup, however, we were not able to train this algorithm to complete convergence, therefore, we were not able to achieve the same accuracy and also the average number of steps is considerably larger than in the single-landmark *DDPG* case. These results are shown in Table 5.12 and Figure 5.21. Figure 5.23 also shows increased variance compared to the multi-task *DDPG* algorithm. We assume, that these two problems could be resolved with longer training. Table 5.14 and Figure 5.25-5.26 compare the results of different multi-landmark approaches and shows that *CNN*-based methods clearly outperform multi-landmark *RL* methods in terms of accuracy. Of the *RL* methods the multi-task *DDPG* approach has been shown to be most accurate and efficient.

SAL with Spatial Configuration

Including spatial configuration of the agents in the *SAL* algorithm allows us learning the spatial configuration of the landmarks during training. During inference, this learned knowledge can be used to localize hidden or occluded landmarks. This is possible because landmarks are spatially dependent on other landmarks. The experimental setup of this algorithm is explained in Section 4.3.3.1. We evaluate the algorithm in two steps. First, we evaluate, if including spatial relations between agents to the neural network input has any benefit in the standard case, where all landmarks are visible. We compare the results with the *SAL* algorithm without spatial configurations in Section 5.6.5. Table 5.13 and

Figure 5.22 and 5.24 show, that this modification achieves comparable performance and therefore does not provide any benefit in the regular case. Furthermore, we evaluate this method on occluded images. We prepare the test set, by replacing the left half of the images with uniform noise, such that one landmark is completely occluded, one partially occluded and three visible. Table 5.15 and Figure 5.28 shows, that this method is able to localize the occluded and the partially occluded landmark up to a certain accuracy. However, the accuracy of the visible landmarks is reduced, which is a consequence of combining the state-space of the individual agents to a joint state-space. Using the *SAL* algorithm without spatial configuration, this problem is even more impactful as can be seen in Table 5.15. The missing information of the occluded landmarks drastically reduces accuracy of the visible landmarks, even though in these regions the agent's observation is given in full detail. The Simultaneous Acting and Localizing with Spatial Configuration (SAL-SC) algorithm, however, is able to compensate this missing information with the relative position of other agents. Table 5.15 also shows, that the multi-task approaches perform well at visible landmarks, but fail if the observation does not provide complete information of the state.

6.2 Future Work

Sub-pixel resolution

The discrete action-space of the *DQN* algorithm allows the agent to only move on discrete image coordinates. The *DDPG* algorithm on the other hand allows predicting continuous actions. As a proposal for future work, this property could be used to localize landmarks with sub-pixel resolution.

Multi-agent reinforcement learning

With the implementation of our multi-task approach, we developed an algorithm, where a single agent is able to localize multiple landmarks sequentially. During inference, multiple identical agents of this kind can be used to simultaneously localize multiple landmarks with reasonable accuracy. With the *SAL-SC* approach, we showed, that using elements of *MARL* it is possible to learn spatial configurations of the landmarks. Both of these methods however have it's disadvantages. With the multi-task approach, we are not able to learn spatial configurations. With the *SAL-SC* approach, we are not able to achieve the accuracy of the multi-task approach. Caused by the joint state-space, occluded landmarks also affect the prediction of visible landmarks, which would not be the case in multi-task landmark localization.

These two methods build a solid basis for a multi-agent approach, where multiple identical multi-task agents are used and a communication protocol is used to learn spatial configurations between the landmarks. Using multi-task agents, we are also able to ensure, that the individual agents localize a landmark, such that the maximum number of steps

among the agents is minimized. This problem of minimizing the maximum cost (i.e. the maximum number of steps) of individual agents is called the Linear Bottleneck Assignment Problem (LBAP).

Data augmentation

Data augmentation is a method, to increase the amount of training data by applying random transformation such as scaling, translating or rotating. This is especially helpful in medical applications where data is costly. While data augmentation is a common method in deep learning applications, it is fairly unexplored in reinforcement learning. During the writing of this thesis, the first extensive study of general data augmentation for *RL* has been published [43]. The problem with data augmentation in the landmark localization setup with *RL* arises from the replay buffer. While in deep learning a training sample can be augmented on the fly, in *RL* the augmented image would have to be stored in RAM until the last sample from this image has disappeared from the replay buffer. Another possibility is to only store the augmentation parameters, and repeatedly augment the image before updating the neural networks with experience samples, which would induce heavy speed losses. Assuming independence of the individual samples from a trajectory, it should be possible to augment the individual observations instead of the entire images, which could be a topic for future research.

3D landmark localization

Medical images are often acquired as 3D volumes, like in 3D *MRI* or Computed Tomography (CT). Consequently, landmark localization has to be done in 3D. While [33] already showed, the *DQN* algorithm is easily applicable to the 3D case, we did not experiment with 3D volumes because of already very long training times. In theory, the *DDPG* algorithm as we proposed in this thesis should be equally applicable to 3D volumes, which should be confirmed in a future project. The speed advantage during inference should be much more significant in 3D volumes.

7 Conclusion

In this thesis, we evaluate the application of Reinforcement Learning (RL) to the problem of landmark localization and propose new methods for single- and multi-landmark localization. *RL* agents are computationally efficient during inference, because they only need to process local image patches, which give them enough information to identify their position in the image and to decide, which action leads to the optimal path to the target landmark. This process of localizing anatomical landmarks based on the perception of local image information is similar to the way a physician localizes anatomical structures in a medical image. The physician, based on his learned knowledge, can estimate the position of an anatomical structure relative to another structure in the image. The closer the initial observation was, the more accurately he is able to estimate the sought landmark. While previous *RL* based algorithms for landmark localization used a Deep Q-Network (DQN) agent with a fixed step size [33], our proposed Deep Deterministic Policy Gradients (DDPG) agent is even closer to mimic the physician's way, because it allows a variable step size, which results in larger action steps if the initial guess was far away from the true landmark and smaller steps if the agent's location is close to the target landmark. Also, using a displacement vector as the agent's action, we immediately know the direction to the target landmark.

The *DDPG* algorithm allows continuous action spaces, which allows the agent to move freely on any coordinate in the cartesian plane, which makes the agent more versatile and faster compared to the *DQN* agent. We were able to show that we achieve slightly better performance in terms of accuracy with the *DDPG* algorithm compared to the *DQN* algorithm. Furthermore, we compare the algorithms to a state-of-the-art Convolutional Neural Network (CNN) approach, where we are able to show that single-landmark *RL* algorithms achieve comparable performance in terms of accuracy. Furthermore, with the *DDPG* approach, we were able to reduce the average number of steps needed to localize the target landmark from 177.5 with the *DQN* approach to 5.85 steps.

The second part of the thesis focused on multi-landmark localization. We extend the algorithms for single-landmark localization to the multi-landmark case by using multi-

task *RL*. We implement a *DQN* multi-task approach and a *DDPG* multi-task approach. The discrete *DQN* approach showed instable training behavior. Therefore the *DDPG* algorithm outperformed the *DQN* algorithm for multi-landmark localization in terms of accuracy. We train these multi-task algorithms on the same network architecture as in the single landmark case, with almost the same number of parameters, which results in similar training time. This means, with these methods we are able to reduce the training time for five landmarks five-fold, if we compare it to the case, where for each landmark an individual agent has to be trained. We did not achieve the performance of single landmark localization in terms of average distance error. However, we assume, that the average distance error could be reduced by designing more complicated network architectures.

We further provide an algorithm for the simultaneous localization of multiple landmarks by using a single agent, that controls the position of multiple "sub-agents" inside the environment. We use the term sub-agent to not confuse this problem with a Multi-Agent Reinforcement Learning (MARL) problem, even though this algorithm resulted from a literature review in *MARL*. A sub-agent is a local image patch inside the image of interest, where each sub-agents goal is to localize a certain landmark. The sub-agents are controlled by a single neural network, which predicts the displacement vector for the sub-agents to simultaneously update them. We were not able to match the performance of our multi-task approaches, probably due to the increased state-space and action-space, which composes of the joint state-space and the joint action-space of the individual sub-agents. However, this algorithm allowed us to integrate spatial relations between the sub-agents during training, to learn spatial correlations of the landmarks. Consequently, we were able to show, that with this approach it is possible to localize hidden or occluded landmarks.

In summary, we evaluate the potential of *RL* to localize landmarks in medical images by learning from patches, thus reducing the need for storing the whole image in *GPU* memory, which is a challenge for *CNN*-based methods, especially when working with large 3D volumes. Moreover, *RL* approaches show more similarity to the way human experts locate anatomical landmarks and as such is an interesting research direction. While we are not able to achieve the high accuracy of current state-of-the-art *CNN*-based methods which process the images as a whole, we show, that our algorithm for single-landmark localization is able to outperform current *RL* methods, while also reducing inference time to a great degree. We state, that *RL* algorithms are able to localize landmarks accurately and efficiently, and should be further evaluated for large 3D volumes, to fully exploit their benefits. Furthermore, our experiments for multi-landmark localization reveal further benefits of *RL* in landmark localization. The multi-task *DDPG* algorithm can be used as a method to efficiently learn an agent on multiple landmarks and the Simultaneous Acting and Localizing with Spatial Configuration (SAL-SC) algorithm showed, that it is possible to localize occluded landmarks, by incorporating elements of *MARL*.

A

Appendix

General Parameters of the DQN Algorithm

Parameter	Value	Description
Learning rate	0.00025	Learning rate used by Adam [39] optimizer
Batch size	32	Number of samples used for Stochastic Gradient Descent (SGD) per training step
Replay buffer size	100,000	Size of the experience replay buffer
Target network update frequency	10,000	The frequency (number of training steps) with which the target network is updated. Corresponds to number C of Algorithm 2
Discount factor γ	0.95	Discount factor in Bellman equation (Equation 3.11)
Initial ϵ	1.0	Initial probability of sampling random actions for exploration in ϵ-greedy exploration
Final ϵ	1.0	Final probability of sampling random actions for exploration in ϵ-greedy exploration
ϵ decay	0.999	Decay factor with which ϵ is reduced after every episode
Scaling factor α	1	Scaling factor α for the reward function (Equation 4.2)

General Parameters of the DDPG Algorithm

Parameter	Value	Description
Learning rate actor	10^{-5}	Learning rate used by Adam [39] optimizer for the actor network
Learning rate critic	10^{-3}	Learning rate used by Adam optimizer for the critic network
Batch size	10^{-3}	Number of experience tuples used for *SGD* to update the critic network
Replay buffer size	100,000	Size of the experience replay buffer
Noise process	gaussian	Noise process used for action exploration
σ (noise)	0.15	Variance of the noise process
τ (soft update)	0.125	Controls the speed, by which the target network's parameters approach the online network. Corresponds to τ of Algorithm 5
Discount factor γ	0.85	Discount factor in Bellman equation for the critic network
Scaling factor α	100	Scaling factor α for the reward function (Equation 4.2) //

B

List of Acronyms

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
AAM	Active Appearance Model
ASM	Active Shape Model
CLM	Constrained Local Model
CNN	Convolutional Neural Network
CT	Computed Tomography
DDPG	Deep Deterministic Policy Gradients
DDQN	Double DQN
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
FC	Fully Connected
FoV	Field of View
GPU	Graphics Processing Unit
LBAP	Linear Bottleneck Assignment Problem
MARL	Multi-Agent Reinforcement Learning
MC	Monte Carlo
MDP	Markov Decision Process
MIA	Medical Image Analysis
MIP	Medical Image Processing
MORL	Multi-Objective Reinforcement Learning
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
MRP	Markov Reward Process
MSE	Mean Squared Error
PCA	Principal Component Analysis

PER	Prioritized Experience Replay
POMDP	Partially Observable Markov Decision Process
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
RF	Random Forest
RL	Reinforcement Learning
SAL	Simultaneous Acting and Localizing
SAL-SC	Simultaneous Acting and Localizing with Spatial Configuration
SCN	SpatialConfiguration-Net
SGD	Stochastic Gradient Descent
SSM	Statistical Shape Model
TD	Temporal Difference
UAV	Unmanned Aerial Vehicle
US	Ultrasound
USI	Ultrasound Imaging

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, `http://tensorflow.org/`. Software available from tensorflow.org. (page 56)

[2] A. Alansary, L. Le Folgoc, G. Vaillant, O. Oktay, Y. Li, W. Bai, J. Passerat-Palmbach, R. Guerrero, K. Kamnitsas, B. Hou, et al. Automatic view planning with multi-scale deep reinforcement learning agents. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 277–285. Springer, 2018. (page 9)

[3] A. Alansary, O. Oktay, Y. Li, L. Le Folgoc, B. Hou, G. Vaillant, K. Kamnitsas, A. Vlontzos, B. Glocker, B. Kainz, et al. Evaluating reinforcement learning agents for anatomical landmark detection. *Medical image analysis*, 53:156–164, 2019. (page 9, 24, 39, 44, 45, 57, 60, 80)

[4] R. Beichel, H. Bischof, F. Leberl, and M. Sonka. Robust active appearance models and their application to medical image analysis. *IEEE transactions on medical imaging*, 24(9):1151–1169, 2005. (page 8)

[5] M. Betke, H. Hong, D. Thomas, C. Prince, and J. P. Ko. Landmark detection in the chest and registration of lung surfaces with an application to nodule registration. *Medical Image Analysis*, 7(3):265–281, 2003. (page 7)

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (page 6)

[7] A. Bulat and G. Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks). In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1021–1030, 2017. (page 7)

[8] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems: revised reprint*. SIAM, 2012. (page 49)

[9] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010. (page 32)

[10] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2488–2496, 2015. (page 44)

[11] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018. (page 7)

[12] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017. (page 7)

[13] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4733–4742, 2016. (page 7)

[14] E. Cheng, J. Chen, J. Yang, H. Deng, Y. Wu, V. Megalooikonomou, B. Gable, and H. Ling. Automatic dent-landmark detection in 3-d cbct dental volumes. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6204–6207. IEEE, 2011. (page 8)

[15] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001. (page 6)

[16] T. F. Cootes, G. J. Edwards, C. J. Taylor, et al. Comparing active shape models with active appearance models. In *Bmvc*, volume 99, pages 173–182. Citeseer, 1999. (page 6)

[17] T. F. Cootes, A. Hill, C. J. Taylor, and J. Haslam. Use of active shape models for locating structures in medical images. *Image and vision computing*, 12(6):355–365, 1994. (page 7)

[18] T. F. Cootes and C. J. Taylor. Active shape models—'smart snakes'. In *BMVC92*, pages 266–275. Springer, 1992. (page 5, 7)

[19] T. F. Cootes, C. J. Taylor, and A. Lanitis. Multi-resolution search with active shape models. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 610–612. IEEE, 1994. (page 6)

[20] A. Criminisi and J. Shotton. *Regression Forests*, pages 47–58. Springer London, London, 2013, `https://doi.org/10.1007/978-1-4471-4929-3_5`. (page 6)

[21] A. Criminisi, D. Robertson, E. Konukoglu, J. Shotton, S. Pathak, S. White, and K. Siddiqui. Regression forests for efficient anatomy detection and localization in computed tomography scans. *Medical image analysis*, 17(8):1293–1303, 2013. (page 8)

[22] D. Cristinacce and T. Cootes. Automatic feature localisation with constrained local models. *Pattern Recognition*, 41(10):3054–3067, 2008. (page 6, 8)

[23] B. C. Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001. (page 21)

[24] M. de Bruijne, B. van Ginneken, M. A. Viergever, and W. J. Niessen. Adapting active shape models for 3d segmentation of tubular structures in medical images. In *Biennial International Conference on Information Processing in Medical Imaging*, pages 136–147. Springer, 2003. (page 7)

[25] X. Dong, Y. Yan, W. Ouyang, and Y. Yang. Style aggregated network for facial landmark detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 379–388, 2018. (page 7)

[26] T. Ebner, D. Stern, R. Donner, H. Bischof, and M. Urschler. Towards automatic bone age estimation from mri: localization of 3d anatomical landmarks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 421–428. Springer, 2014. (page 8)

[27] G. J. Edwards, T. F. Cootes, and C. J. Taylor. Face recognition using active appearance models. In *European conference on computer vision*, pages 581–595. Springer, 1998. (page 6)

[28] G. J. Edwards, C. J. Taylor, and T. F. Cootes. Interpreting face images using active appearance models. In *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, pages 300–305. IEEE, 1998. (page 6)

[29] J. Ehrhardt, H. Handels, W. Plötz, and S. Pöppl. Atlas-based recognition of anatomical structures and landmarks and the automatic computation of orthopedic parameters. *Methods of information in medicine*, 43(04):391–397, 2004. (page 7)

[30] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool. Random forests for real time 3d face analysis. *International journal of computer vision*, 101(3):437–458, 2013. (page 6)

[31] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. Rmpe: Regional multi-person pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2334–2343, 2017. (page 7)

[32] S. Frantz, K. Rohr, and H. S. Stiehl. Localization of 3d anatomical point landmarks in 3d tomographic images using deformable models. In *International conference on medical image computing and computer-assisted intervention*, pages 492–501. Springer, 2000. (page 7)

[33] F. C. Ghesu, B. Georgescu, T. Mansi, D. Neumann, J. Hornegger, and D. Comaniciu. An artificial agent for anatomical landmark detection in medical images. In *International conference on medical image computing and computer-assisted intervention*, pages 229–237. Springer, 2016. (page 4, 9, 37, 38, 57, 60, 79, 80, 83, 86, 87)

[34] F.-C. Ghesu, B. Georgescu, Y. Zheng, S. Grbic, A. Maier, J. Hornegger, and D. Comaniciu. Multi-scale deep reinforcement learning for real-time 3d-landmark detection in ct scans. *IEEE transactions on pattern analysis and machine intelligence*, 41(1):176–189, 2017. (page 80)

[35] T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995. (page 6)

[36] T. J. Hutton, S. Cunningham, and P. Hammond. An evaluation of active shape models for the automatic identification of cephalometric landmarks. *The European Journal of Orthodontics*, 22(5):499–508, 2000. (page 7)

[37] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, pages 34–50. Springer, 2016. (page 7)

[38] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998. (page 15)

[39] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (page 40, 89, 90)

[40] M. Koestinger, P. Wohlhart, P. M. Roth, and H. Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 2144–2151. IEEE, 2011. (page 3)

[41] J. Krebs, T. Mansi, H. Delingette, L. Zhang, F. C. Ghesu, S. Miao, A. K. Maier, N. Ayache, R. Liao, and A. Kamen. Robust non-rigid registration through agent-based action learning. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 344–352. Springer, 2017. (page 9)

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. (page 6)

[43] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020. (page 86)

[44] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990. (page 6)

[45] G. Leroy, D. Rueckert, and A. Alansary. Communicative reinforcement learning agents for landmark detection in brain images. *arXiv preprint arXiv:2008.08055*, 2020. (page 9)

[46] Y. Li, B. Sun, T. Wu, and Y. Wang. Face detection with end-to-end integration of a convnet and a 3d model. In *European Conference on Computer Vision*, pages 420–436. Springer, 2016. (page 7)

[47] R. Liao, S. Miao, P. de Tournemire, S. Grbic, A. Kamen, T. Mansi, and D. Comaniciu. An artificial agent for robust image registration. *arXiv preprint arXiv:1611.10336*, 2016. (page 9)

[48] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. (page 29, 31)

[49] C. Lindner, P. A. Bromiley, M. C. Ionita, and T. F. Cootes. Robust and accurate shape model matching using random forest regression-voting. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1862–1874, 2014. (page 6)

[50] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994. (page 35)

[51] J. Lv, X. Shao, J. Xing, C. Cheng, and X. Zhou. A deep regression architecture with two-stage re-initialization for high performance facial landmark detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3317–3326, 2017. (page 7)

[52] G. Maicas, G. Carneiro, A. P. Bradley, J. C. Nascimento, and I. Reid. Deep reinforcement learning for active breast lesion detection from dce-mri. In *International conference on medical image computing and computer-assisted intervention*, pages 665–673. Springer, 2017. (page 9, 44)

[53] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. (page 81)

[54] S. C. Mitchell, J. G. Bosch, B. P. Lelieveldt, R. J. Van Der Geest, J. H. Reiber, and M. Sonka. 3-d active appearance models: segmentation of cardiac mr and ultrasound images. *IEEE transactions on medical imaging*, 21(9):1167–1178, 2002. (page 8)

[55] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. (page 29, 82)

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. (page 3, 8, 22, 38)

[57] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016. (page 6, 7)

[58] J. M. Noothout, B. D. de Vos, J. M. Wolterink, T. Leiner, and I. Išgum. Cnn-based landmark detection in cardiac cta scans. *arXiv preprint arXiv:1804.04963*, 2018. (page 8)

[59] U. V. S. of Engineering. Digital hand atlas database system, 2015, `https://ipilab.usc.edu/research/baaweb/`. (page 55)

[60] C. Payer, D. Štern, H. Bischof, and M. Urschler. Regressing heatmaps for multiple landmark localization using cnns. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 230–238. Springer, 2016. (page 8, 55, 58, 59, 60, 74, 76, 81)

[61] T. Pfister, J. Charles, and A. Zisserman. Flowing convnets for human pose estimation in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1913–1921, 2015. (page 6)

[62] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4929–4937, 2016. (page 7)

[63] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135, 2017. (page 7)

[64] K. Rohr. On 3d differential operators for detecting point landmarks. *Image and Vision Computing*, 15(3):219–233, 1997. (page 7)

[65] K. Rohr. *Landmark-based image analysis: using geometric and intensity models*, volume 21. Springer Science & Business Media, 2001. (page 5)

[66] K. Rohr, H. S. Stiehl, R. Sprengel, T. M. Buzug, J. Weese, and M. Kuhn. Landmark-based elastic registration using approximating thin-plate splines. *IEEE Transactions on medical imaging*, 20(6):526–534, 2001. (page 7)

[67] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. (page 6)

[68] H. R. Roth, L. Lu, A. Seff, K. M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, and R. M. Summers. A new 2.5 d representation for lymph node detection using random sets of deep convolutional neural network observations. In *International conference on medical image computing and computer-assisted intervention*, pages 520–527. Springer, 2014. (page 8)

[69] F. Sahba, H. R. Tizhoosh, and M. M. Salama. A reinforcement learning framework for medical image segmentation. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 511–517. IEEE, 2006. (page 9)

[70] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. (page 25)

[71] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. (page 28)

[72] D. Silver. Introduction to reinforcement learning with david silver, 2015, `https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver`. (page 11)

[73] D. Štern, B. Likar, F. Pernuš, and T. Vrtovec. Automated detection of spinal centrelines, vertebral bodies and intervertebral discs in ct and mr images of lumbar spine. *Physics in Medicine & Biology*, 55(1):247, 2009. (page 7)

[74] K. Sun, B. Xiao, D. Liu, and J. Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5693–5703, 2019. (page 6)

[75] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3476–3483, 2013. (page 7)

[76] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. (page 11, 20, 29)

[77] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 648–656, 2015. (page 6)

[78] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in neural information processing systems*, pages 1799–1807, 2014. (page 6)

[79] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014. (page 6)

[80] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930. (page 31)

[81] M. Urschler, T. Ebner, and D. Štern. Integrating geometric configuration and appearance information into a unified framework for anatomical landmark localization. *Medical image analysis*, 43:23–36, 2018. (page 8)

[82] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015. (page 9, 24)

[83] A. Vlontzos, A. Alansary, K. Kamnitsas, D. Rueckert, and B. Kainz. Multiple landmark detection using multi-agent reinforcement learning. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 262–270. Springer, 2019. (page 9, 47)

[84] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016. (page 9, 24, 25)

[85] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. (page 20)

[86] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016. (page 6)

[87] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. (page 27)

[88] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015. (page 8)

[89] D. Yang, S. Zhang, Z. Yan, C. Tan, K. Li, and D. Metaxas. Automated anatomical landmark detection ondistal femur surface using convolutional neural network. In *2015 IEEE 12th international symposium on biomedical imaging (ISBI)*, pages 17–21. IEEE, 2015. (page 8)

[90] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014. (page 7)

[91] Y. Zheng, D. Liu, B. Georgescu, H. Nguyen, and D. Comaniciu. 3d deep learning for efficient and robust landmark detection in volumetric data. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 565–572. Springer, 2015. (page 8)