HAW HAMBURG

MASTER THESIS
Katja Schöttler

Anwendung von Reinforcement Learning und Potentialfeldern für eine Multi-Agenten-Steuerung für Drohnen zur Lokalisierung von Funksignalen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Katja Schöttler

Anwendung von Reinforcement Learning und Potentialfeldern für eine Multi-Agenten-Steuerung für Drohnen zur Lokalisierung von Funksignalen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Dr. Ing. Peter Siebert

Eingereicht am: 22. Dezember 2023

Katja Schöttler

Thema der Arbeit

Anwendung von Reinforcement Learning und Potentialfeldern für eine Multi-Agenten-Steuerung für Drohnen zur Lokalisierung von Funksignalen

Stichworte

Reinforcement Learning, Potentialfelder, Drohnen, Schwarmkontrolle, Multi-Agenten, Digitaler Zwilling

Kurzzusammenfassung

In dieser Arbeit wird die Generierung von Potentialkarten mit einem CNN Autoencoders und Reinforcement Learning behandelt. Hierfür wird eine mit dem MARS-Framework entwickelte Simulationsumgebung genutzt, um die generierten Karten zu erforschen und zu bewerten. Neben dem technischen Aufbau werden die Ergebnisse der verschiedenen getesteten Netzkonfigurationen erläutert.

Katja Schöttler

Title of Thesis

Reinforcement learning and potential fields for a multi-agent control for drones to localise radio signals.

Keywords

Reinforcement Learning, Potetialfields, Drones, Schwarm control, Multi-Agent, digital Twin

Abstract

This thesis deals with the generation of potential maps with a CNN autoencoder and reinforcement learning. For this purpose, a simulation environment developed with the MARS framework is used to explore and evaluate the generated maps. In addition to the technical setup, the results of the various network configurations tested are explained.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

In den letzten Jahren hat die Entwicklung autonomer Drohnenschwärme eine bemerkenswerte Aufmerksamkeit auf sich gezogen, da sie das Potential haben eine Vielzahl von Anwendungen zu revolutionieren. Von Rettungsmissionen [6], über Umweltüberwachung [8, 33], bis hin zu landwirtschaftlichen Aufgaben [3, 49] finden Drohnenschwärme Anwendung. Besonders auch bei der Ortung von Funksignalen kommen die Vorteile zum Tragen, denn Drohnen können durch ihre hohe Mobilität und Flexibilität in Bereichen eingesetzt werden, in denen herkömmliche Ansätze an ihre Grenzen stoßen. [44] So können sie auch für den Menschen unwegsames Gelände schnell erkunden und durch ihre Vogelperspektive große Bereiche auf einmal überwachen. [2] Aufgrund der Vogelperspektive und der dadurch resultierenden geringeren Anzahl an Hindernissen um den Empfänger, kommt es zu weniger Reflexionen und Verzerrungen, wodurch eine präzisere Ortung möglich ist. Ein weiterer Vorteil ist die Kostenersparnis, da bereits einfache Drohnen ausreichen, die lediglich mit GPS und einem Funkempfänger ausgestattet werden müssen. [44] Mit den richtigen Technologien ist auch Echtzeiterfassung und -verarbeitung möglich, wodurch schnellere Reaktionen auf Veränderungen oder unerwartete Ereignisse möglich sind. Zudem führt die Mobilität, Flexibilität und Wirtschaftlichkeit dazu, dass es eine Vielzahl an zukünftigen Anwendungsfeldern für solche Drohnenschwärme geben wird. [6, 8, 33, 3, 49]

Eine entscheidende Komponente für den Erfolg von Drohnenschwärmen ist eine autonome Steuerung. Diese verleiht ihnen die Möglichkeit auch kooperative Aufgaben, für die es eine Vielzahl an Drohnen benötigt, effizient zu erfüllen. Hierfür benötigt es genaue und adaptive Steuerungssysteme, die in der Lage sind, verschiedene Herausforderungen der realen Welt zu bewältigen. Hierfür bietet sich besonders der Einsatz von digitalen Zwillingen an. [63]

Eine der Schlüsselkomponenten für die präzise Steuerung von Drohnen ist die Fähigkeit, genaue und zuverlässige Informationen über die Umgebung zu erhalten. Hierfür werden meist Sensoren eingesetzt, die es der Drohne ermöglichen ihre Umgebung wahrzunehmen und darauf basierend zu handeln. [3, 33] Dies bedeutet, dass zusätzliche und vermeintlich kostspielige Hardware benötigt wird, die zu einer höheren Nutzlast für die Drohnen führt. Das hat wiederum nicht nur einen höheren Energieverbrauch und somit auch kürzere Flugzeiten zur Folge, sondern kann auch dazu führen, dass Drohnen mit einer noch höheren Nutzlast benötigt werden. Wenn auf zusätzliche Sensoren verzichtet werden soll, benötigen die Drohnen alternative Wege, um sich in ihrer Umgebung zurechtzufinden.

Eine Möglichkeit hierfür sind sogenannte Potentialfelder, welche häufig im Bereich der Robotik für die Fortbewegung Verwendung finden. [30] Diese können, basierend auf Informationen über topografische Merkmale, Hindernisse und andere relevante Aspekte, generiert werden und dadurch die Umwelt abbilden. Traditionell werden solche Karten mithilfe von Geoinformationssystemen (GIS) erstellt, indem verschiedene räumliche Ebenen miteinander kombiniert werden. Der Nachteil eines solchen Vorgehens ist jedoch, dass diese in der Komplexität sehr eingeschränkt sind. Für solche Verfahren bedarf es eines vorher definierten Regelsatzes, der vorgibt wie die Input Daten verrechnet werden. Dies führt auch dazu, dass besonders mit steigender Komplexität spätere Änderungen immer zeitaufwendiger werden.

Hier bietet die Integration von Reinforcement Learning (RL) in den Prozess der Potentialfeldererstellung eine Reihe von überzeugenden Vorteilen. Anders als traditionelle GIS-Ansätze ermöglicht RL eine adaptive und lernfähige Generierung von Potentialfeldern. Durch die Verwendung von mehreren Ebenen, darunter beispielsweise Höhendaten, Steigungen, Ausrichtungen der Steigungen und Informationen über Gebäude, kann das RL-Modell komplexe Korrelationen zwischen diesen Merkmalen erfassen und ein tieferes Verständnis der Umgebung entwickeln.

Ein weiterer entscheidender Vorteil liegt in der Fähigkeit des RL-Modells, aus Erfahrungen zu lernen und sich an unterschiedliche Umgebungen anzupassen. Dies ermöglicht es dem Steuerungssystem sich an Änderungen anzupassen und unvorhergesehene Situationen erfolgreich zu bewältigen. Dieser adaptive Ansatz hebt sich von traditionellen, statischen GIS-Methoden ab. So kann auf veränderte Anforderungen schnell reagiert werden. Mithilfe von kleineren Änderungen, beispielsweise an der Belohnungsfunktion und mit ein paar weiteren Trainingsiterationen kann das Modell an die geänderten Anforderungen angepasst werden. Beispielsweise kann der Input erweitert werden. Als Beispiel

hierfür könnten neben der in dieser Arbeit genutzten Karten zusätzlich Wetterkarten hinzugefügt werden. Dadurch würden die Bewertungen deutlich dynamischer werden, da nicht nur statische Hindernisse existieren, sondern Aspekte wie Windschatten oder Bereiche mit starken Böen relevant werden. Besonders auch das Trainieren kann sehr gut automatisiert werden, sodass neue Modelle mit wenig Aufwand erstellt werden können. Ein weiterer Vorteil gegenüber dem klassischen Verschneiden, bei dem Regeln konkret definiert werden müssen ist, dass Modelle in der Lage sind, komplexe Zusammenhänge und zugrunde liegende Muster zu erlernen und diese Strukturen dann auch auf neue Umgebungen anzuwenden.

Dennoch birgt ein solches Vorgehen auch einige Herausforderungen, die gemeistert werden müssen. So bedarf es genügend Ressourcen, die für das rechenintensive Training genutzt werden können. Ist dies vorhanden, so ist oftmals das Finden einer guten Netzwerkarchitektur und die Auswahl der richtigen Hyperparameter nicht minder komplex und benötigt oftmals experimentelles Tuning. Dies stellt einen entscheidenden Faktor für den Erfolg des Modells dar. Neben der Planung der Machine Learning Komponente muss auch ein geeigneter Weg für das Lernen gefunden werden. Es muss ein passender Prozess zum Erforschen und Bewerten der generierten Karten gefunden werden. Dafür wird zum einen ein Weg benötigt die Umwelt wahrzunehmen und zum anderen eine gute Planung und ein gutes Design einer geeigneten Belohnungsfunktion.

1.2 Forschungsfrage und Ziele

In dieser Arbeit wird die Anwendung von Reinforcement Learning zur Generierung von Potentialfeldern als Grundlage für die Steuerung autonomer Drohnenschwärme untersucht. Hierbei wird ein digitaler Zwilling eines Drohenschwarms genutzt, der sich, basierend auf der generierten Karte, in seiner Umwelt bewegt. Hierfür wird eine selbst entwickelte Simulationsumgebung genutzt, um so eine Umgebung zu schaffen, in der verschiedenen Ansätze erprobt werden können und evaluiert werden kann, welcher sich für die Generierung der Karten am besten eignet. In diesem Rahmen soll ein geeigneter Prozess entstehen, der es ermöglicht ein Modell über mehrere Iterationen zu trainieren. Auf diese Weise können die Ergebnisse ausgewertet werden, um anschließend zu evaluieren wie diese sich verbessert haben. Dazu gehört neben dem simulierten Erforschen und Bewerten der Karten und dem Trainieren des Netzes auch das Zusammenstellen und Aufbereiten eines geeigneten Input Datensatzes für Tests.

Basierend auf dieser Struktur können verschiedene Netzwerkkonfigurationen getestet, verglichen und ausgewertet werden. Die gewonnen Ergebnisse können als Grundlage für weitere Arbeiten in diesem Bereich dienen. Zudem entsteht ein prototypisches Modell, das die Tauglichkeit dieses Ansatzes überprüft.

1.3 Gliederung der Arbeit

Zu Beginn der Arbeit werden in Kapitel 2 die relevanten Grundlagen erläutert. Dabei wird zunächst auf die Funkortung im Allgemeinen und den Einsatz von Drohnen für diesen Zweck eingegangen. Anschließend werden die technischen Grundlagen wie digitale Zwillinge, Multiagentensysteme, Neuronale Netze und Reinforcement Learning ausführlich dargestellt.

In Kapitel 3 wird der Aufbau des Systems und der Ablauf des Reinforcement Learnings erklärt. Hierbei wird der erstellte Datensatz im Detail erläutert und der Trainingsablauf beschrieben. Zudem werden der allgemeine Aufbau sowie die einzelnen Komponenten der Simulationsumgebung vorgestellt. Des Weiteren wird die erstellte Simulation zur Ortung von Funksignalen mit Drohnen erläutert.

Kapitel 4 beinhaltet die verschiedenen durchgeführten Experimente, die Ergebnisse und deren Auswertungen. Anschließend wird in Kapitel 5 nochmal Bezug auf die Ergebnisse genommen und diese diskutiert.

Abschließend bietet das Kapitel 6 ein Fazit und einen Ausblick auf mögliche Weiterentwicklungen und Anwendungen.

2 Grundlagen

Dieses Kapitel enthält eine Einführung in die Grundlagen, die für diese Arbeit relevant sind. Zu Beginn wird in Abschnitt 2.1 eine Einführung in die Funkortung mithilfe des Time Difference of Arrival Verfahren gegeben. Die Funkortung ist als domänenspezifische Grundlage für das Verständnis der Arbeit relevant. Im Anschluss erfolgt eine kurze Einführung in die Thematik der Multiagentensysteme und der digitalen Zwillinge. Anschließend wird das MARS Framework welches für die Entwicklung genutzt wird vorgestellt. Gefolgt von einer Vorstellung der allgemeinen Funktionsweise von Neuronalen Netze und des Reinforcement Learnings sowie konkreter Methoden für den hier genannten Einsatzfall. Bevor abschließend eine Auswahl verwandter Forschungsarbeiten vorgestellt werden, werden noch Potentialfelder erläutert.

2.1 Funkortung mithilfe des Time Difference of Arrival Verfahren

Im Folgenden werden die Grundlagen des Time Difference of Arrival Verfahren erläutert. Zu Beginn wird die Funktionsweise des Verfahrens erklärt. Anschließend wird auf die Einschränkungen des Verfahrens eingegangen und den Einsatz von Drohnen hierfür erläutert.

2.1.1 Funktionsweise

Die Funkortung findet in verschiedenen Bereichen Einsatz. Von GPS bis hin zu Einsatzbereichen wie Lawinenverschüttetensuchgeräte. [27, 55] Sie ermöglicht die präzise Bestimmung der Position von Signalquellen. Dies wird nicht nur zur Navigation eingesetzt, sondern auch zur Überwachung oder Datenerfassung. Die Nutzung von Funksignalen zur Ortung ist ein fundamentaler Bestandteil moderner Technologien und Systeme. Hierbei

lässt sie sich in zwei verschiedene Bereiche einteilen. Die Eigenpeilung ist der erste Bereich. Hierbei orientert sich der Nutzer selbst im Funkfeld, um den eigenen Standort zu ermitteln. Dem gegenüber steht die Fremdpeilung. [46] Für die Ausführung einer Fremdpeilung gibt es verschiedene Verfahren, um die Position der Signalquelle zu bestimmen. Eine häufig verwendete Methode ist das Time Difference of Arrival (TDoA) Verfahren. [46, 27] Hierfür wird lediglich eine Empfangseinheit, bestehend aus einem Empfänger und einer Antenne, und eine gemeinsame, synchronisierte Zeitreferenz, wie beispielsweise GPS, benötigt. [55, 35]

Bei diesem Verfahren werden die Laufzeitdifferenzen der elektromagnetischen Wellen der selben Frequenz, von mindestens zwei Empfangseinheiten an verschiedenen Positionen gemessen. Basierend auf den Zeitdifferenzen lässt sich die Differenz der Entfernung zwischen der Signalquelle und den beiden Empfängern bestimmen.

2.1.2 Probleme des Verfahrens

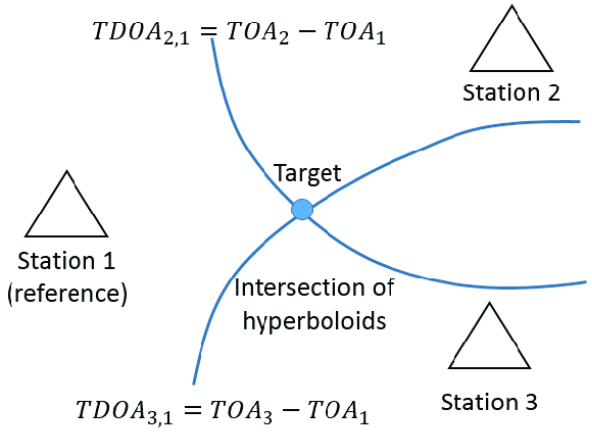

Abbildung 2.1: Ortung mit drei Empfängern und Signal innerhalb der Vermaschung [18]

Damit eine Lokalisierung des Empfängers möglich ist, werden mindestens drei Empfänger benötigt, sodass mehrere Hyperbeln berechnet werden können. Sind die Empfänger gut zum Sender positioniert, kann hierbei ein Schnittpunkt entstehen. Dies ist in Abbildung 2.1 dargestellt. Dieser Schnittpunkt stellt den Standort des Senders dar. Aufgrund von Messungenauigkeiten entsteht in der Praxis kein genauer Schnittpunkt, sondern ein Be-

reich, der den ungefähren Standort des Senders beschreibt. Dementsprechend wird die Berechnung des Standorts mit höherer Anzahl an Empfängern genauer. [55, 46]

Es gibt auch Einschränkungen dieses Verfahrens. Die Quelle eines Signals sollte sich zwischen den Empfänger befinden. Ist dies nicht der Fall, schneiden sich die hyperbolischen Bögen in einem sehr kleinen Winkel, wie in Abbildung 2.2 zu sehen ist. Dadurch wird der mögliche Bereich, in dem sich das Signal befindet, größer und die Ortung ungenauer, wodurch das Ergebnis eher eine Einschätzung der Richtung ist als eine genaue Ortung. [55]

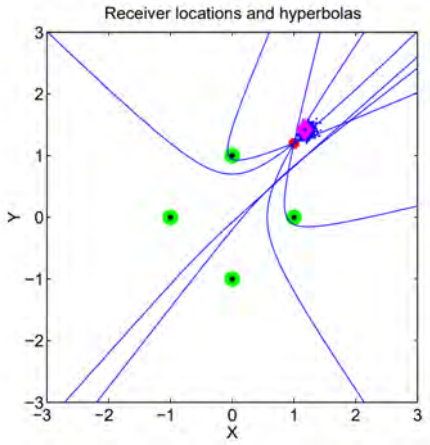

Abbildung 2.2: Ortung mit drei Empfängern und Signal außerhalb der Vermaschung [26]

Ein weiteres Problem stellen hierbei Hindernisse dar. Das Verfahren geht davon aus, dass das Signal auf dem kürzesten Weg zum Ziel gelangt. Befinden sich allerdings Hindernisse, wie Gebäude, Tunnel oder ähnliches im Weg kann dies dazu führen, dass das Signal nicht über den kürzesten Weg empfangen wird, wodurch die Lokalisierung ungenau wird. [46]

2.1.3 Einsatz von TDoA mit Drohnen

Dadurch, dass lediglich eine einfache Empfangseinheit und ein GPS Empfänger benötigt werden, entsteht wenig Nutzlast, was es ermöglicht diese Hardware auf einer Drohne zu montieren. Vorteilhaft ist hierbei die Perspektive von oben. Aufgrund der Vogelperspektive ist die Signalausbreitung zwischen dem Sender und den Empfängern nahezu ideal, da hier in der Regel weniger Hindernisse existieren.

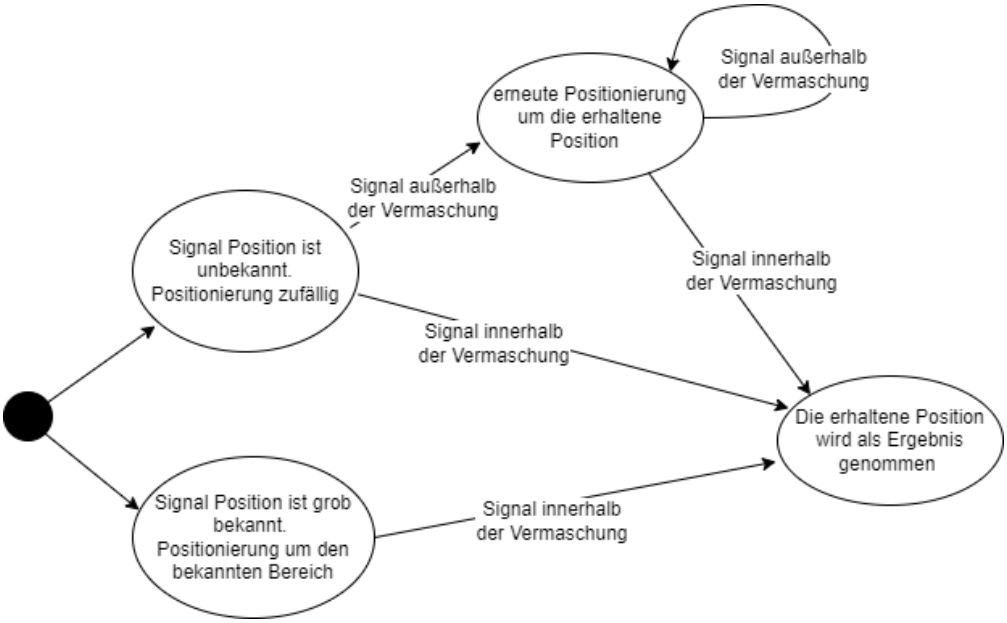

Abbildung 2.3: Mögliche Abläufe bei der Ortung von Signalen mit Drohnen und TDoA

Ein weiterer Vorteil ist die nahezu uneingeschränkte Bewegungsfreiheit der Drohnen im Raum aufgrund fehlender Hindernisse. Auf diese Weise können sie sich optimal zum Signal positionieren und dafür sorgen, dass das Signal sich zwischen den Drohnen befindet. Hierbei gibt es zwei mögliche Szenarien, wie in Abbildung 2.3 zu sehen ist. Im ersten Szenario kennen die Drohnen bereits eine grobe Schätzung des Standortes des Senders und können sich direkt in einer idealen Formation um dessen Standort positionieren, wodurch bereits die erste Ortung sehr präzise ist. Dieser Ablauf ist im unteren Zweig des Baumes in Abbildung 2.3 dargestellt. Das zweite Szenario ist im oberen Zweig in Abbildung 2.3 dargestellt. Hier ist der Standort vorher nicht bekannt, lediglich die Frequenz. In diesem Fall ist ihre anfängliche Position zunächst unbedeutend und nur die Verteilung im Raum ist relevant. Dann ist die Wahrscheinlichkeit sehr hoch, dass sich die Signalquelle außerhalb der Vermaschung befindet und die Ortung ungenau ist. In diesem Fall können die Drohnen diesen Standort für eine erneute Positionierung um das Signal nutzen. Wenn sich der neu erhaltene Standort innerhalb der Vermaschung befindet, haben die Drohnen die Quelle gefunden. Andernfalls kann dieser Vorgang beliebig oft wiederholt werden.

2.2 Multiagentensysteme

Ein Agent ist eine Softwareeinheit, die in einer Umgebung handlungsfähig ist. Er kann die Umgebung beispielsweise über Sensoren wahrnehmen und basierend auf dem Input reagieren sowie Handlungen ausführen. Dabei kann beispielsweise ein einfacher Agent lediglich auf den Input über eine Schwellenfunktion reagieren. Ein komplexerer Agent hingegen kann eigenständige Entscheidungen treffen und intelligent handeln. [52]

Dennoch reicht ein einzelner Agent oft nicht aus, wodurch Multiagentensysteme (MAS) entstehen. Hierbei handelt es sich um Systeme, die aus einer Vielzahl von Agenten bestehen. Die Agenten arbeiten auf ein übergeordnetes Ziel hin. Dabei lässt sich die zentrale Verhaltensweise in kooperatives und nicht-kooperatives Verhalten unterteilen. [52]

Bei kooperativem Verhalten wird mithilfe von Zusammenarbeit versucht ein Ziel gemeinsam zu erreichen. Die größte Herausforderung hierbei ist die Entscheidungsfindung, besonders in großen Systemen. Hierfür können die Agenten in der Regel miteinander kommunizieren. Die Alternative ist ein nicht-kooperatives Verhalten. Dabei kann es sich, abhängig von der übergeordneten Aufgabe, um ein kompetitives Verhalten handeln, bei dem die Agenten aktiv gegeneinander arbeiten. Es kann aber auch eine Arbeitsweise existieren, bei der es keine zentrale Vereinbarung gibt, und die Agenten unabhängig voneinander arbeiten. [52]

Diese beiden Arten müssen allerdings nicht immer gesondert voneinander auftreten, es kann auch, abhängig von der Aufgabenstellung, beide Verhaltensweisen in einem System geben. [52] Dies ist auch in dieser Arbeit der Fall ist.

2.3 Digitaler Zwilling

Eine exakte virtuelle Repräsentation eines physischen Modells wird als digitaler Zwilling bezeichnet. Diese virtuelle Repräsentation bildet die exakten Inhalte, Strukturen und Zustände der realen Welt parallel und in Echtzeit ab. Der Begriff des digitalen Zwillings wurde von Michael Grieves in den 2000er Jahren geprägt. [11, 14] Seitdem finden diese in vielen verschiedenen Bereichen Anwendung. [14]

Die Begrifflichkeit des digitalen Zwillings hat sich in den letzten Jahren erweitert und gelockert, sodass er heutzutage zur Charakterisierung einer Vielzahl an digitalen Simulationsmodellen verwendet wird. [11] In diesem Zusammenhang bedarf es einer Abgrenzung

zwischen der Begrifflichkeit des digitalen Zwillings im ursprünglichen Sinn und der des digitalen Modells. Hierfür wird die Erläuterung von Batty (2018) in [11] als Basis genommen. Demnach ist ein Modell per Definition erstmals nur eine abstrahierte Abbildung des Systems und muss nicht das vollständige physische System abbilden. Damit ein Modell als digitaler Zwilling angesehen werden kann, muss der digitale Zwilling eng mit dem System verbunden sein und darf in gewissen Teilen nicht vom originalen System unterschieden werden können. Folglich ist ein digitaler Zwilling immer auch ein Modell des Systems, allerdings ist nicht jedes Modell ein digitaler Zwilling. [11, 17]

2.4 MARS Framework

Das Multi-Agent Research and Simulation (MARS) Framework [31] ist ein verteiltes Simulationsframework, das speziell für die Entwicklung von Multiagenten-Simulationen konzipiert wurde. Es wurde an der Hochschule für angewandte Wissenschaften (HAW) Hamburg entwickelt und steht öffentlich zur Verfügung. Das Framework ist in C# geschrieben und kann als NuGet Package eingebunden werden. [43] Aufgrund seiner Eignung für das vorliegende Anwendungsgebiet wurde entschieden, dieses Framework in dieser Arbeit zu verwenden. Zahlreiche Arbeiten haben das MARS Framework bereits erfolgreich für unterschiedliche Aufgaben genutzt. Ein Beispiel ist die Implementierung eines digitalen Zwillings von Hamburg. [17] Weitere Anwendungen erstrecken sich über die Modellierung adaptiven menschlichen Verhaltens [39, 45] bis hin zu Verkehrssimulationen. [60]

Das Konzept eines typischen agentenbasierten Modells, das mit MARS entwickelt wird, sieht wie folgt aus. Der Kern bildet die *Simulation*, welche die Ausführung eines Modells in einem vordefinierten Szenario beschreibt. Das *Szenario* wiederum umfasst Aspekte wie die Zeitspanne, die Tick-Größe, die Anzahl der Agenten und die genutzten Eingangsdaten für das Modell. Jedes Szenario besteht aus einer *Modellbeschreibung*, die angibt, welche Elemente des Modells Teil des Szenarios sind. [43, 17]

Ein Modell in MARS setzt sich aus Agenten, Entitäten und Ebenen zusammen. Ein *Agent* ist eine eigenständige Komponente, welche die Umgebung wahrnimmt, entsprechend reagiert und Aktionen ausführt. Hierfür hat der Agent eine Tick()-Funktion, die jeden Zeitschritt aufgerufen wird. Agenten können nicht nur mit der Umgebung oder untereinander interagieren, sondern auch *Entitäten* nutzen. Diese sind einfache Komponenten mit einem Lebenszyklus, die selbst keine Aktionen ausführen können. Die letzte

Komponente, die zu einem Modell gehört, sind die *Ebenen*. Diese sind die Grundlage auf der die Agenten interagieren. Sie sind Teil der Umwelt in der die Agenten leben und enthalten verschiedene Informationen. Die Agenten haben eine Hauptebene auf der sie leben. [43, 17]

Basierend auf den Ebenen gibt es drei grundlegende Modellarten, die beschreiben, wie sich die Agenten fortbewegen können. Die einfachste Art sind *Grid-Modelle*, bei denen sich die Agenten auf einem Raster bewegen. Die Felder des Rasters können verschiedene Eigenschaften haben und dienen als Grundlage für die Fortbewegung. Für Informationen aus der realen Welt gibt es zwei Arten von *Real-World-Modellen*. Bei grundlegenden Modellen können sich die Agenten frei bewegen, während infrastrukturbasierte Modelle Straßenmodelle umfassen. Diese enthalten nicht nur Straßen, sondern auch relevante Punkte, wie Restaurants oder Sehenswürdigkeiten. Zudem stellt das MARS Framework bereits grundlegende Wegplanungsalgorithmen bereit, die für die Fortbewegung in den verschiedenen Modellen genutzt werden können. [43]

2.5 Neuronale Netze

Es existieren viele verschiedene Typen von Neuronalen Netzen. Das grundlegendste Neuronale Netz besteht aus drei Schichten. Die erste ist die Eingabeschicht. Darauf folgt eine verborgene Schicht und die letzte Ebene ist die Ausgabeschicht. Bei einem klassischen vorwärtsgerichteten Netz sind die Neuronen einer Schicht mit denen in der folgenden verbunden. [23] Dies ist in Abbildung 2.4 dargestellt. In diesem Beispiel enthält die Eingabeschicht fünf Neuronen. Diese Neuronen sind mit den drei Neuronen der verborgenen Schicht verbunden und diese entsprechend mit der Ausgabeschicht.

Das Ziel eines solchen Netzes besteht darin, eine Funktion $f(x)$ zu erlernen. Die Daten, die der Eingabeschicht eines vorwärtsgerichteten Netzes übergeben werden, werden dann in den verschiedenen Schichten bis zur Ausgabeschicht verarbeitet. Dabei enthält jedes Neuron eine Aktivierungsfunktion, die abhängig vom Input etwas ausgibt (siehe Abschnitt 2.5.1). Zudem werden die einzelnen Verbindungen gewichtet, über die der Input weitergegeben wird. Auf diese Weise werden die Daten verarbeitet, sodass im Idealfall das gewünschte Ergebnis am Ende herauskommt. [23, 52]

Die Differenz zwischen dem gewünschten Ausgabewert und dem tatsächlichen Ausgabewert ist der Fehler (Loss), der mit einer Loss-Funktion bestimmt wird (Abschnitt 2.5.2).

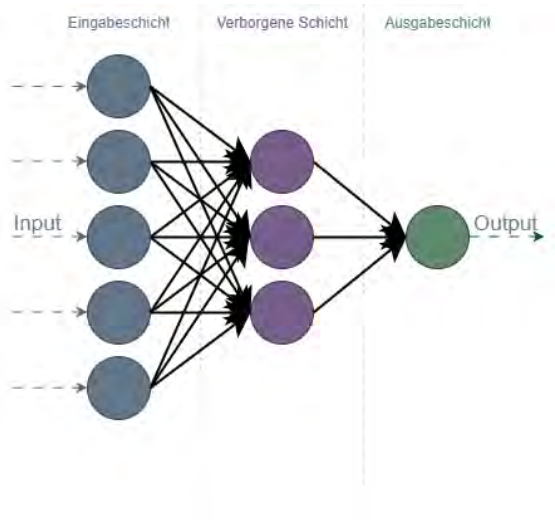

Abbildung 2.4: Visualisierung eines klassischen vorwärtsgerichteten Neuronalen Netzes

Basierend auf diesem Fehler wird dann mittels Backpropagation vom Ende zum Anfang durch das Netz gegangen, und die Gewichtungen werden angepasst. Hierfür wird eine Optimierungsfunktion benötigt, wie im Abschnitt 2.5.3 genauer erläutert. [23, 52]

2.5.1 Aktivierungsfunktionen

Jedes Neuron besteht lediglich aus einer parametrisierten Funktion. Durch die Kombination mehrerer solcher Funktionen in einer Schicht und den Einsatz mehrerer Schichten können komplexe Aufgaben gelöst werden. Jedes dieser Neuronen bekommt dabei Inputs von den vorherigen Neuronen und hat Ausgänge zu den Neuronen der nachfolgenden Schicht. Ein Neuron selbst berechnet lediglich die gewichtete Summe der Inputs und wendet darauf eine nicht-lineare Funktion an, um den Output zu generieren, der an die nächste Schicht weitergegeben wird. [23, 52]

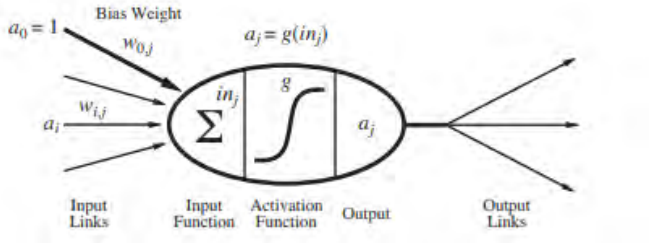

Abbildung 2.5: Visualisierung eines Neurons [51]

Ist a_j die Ausgabe von Neuron j und $w_{i,j}$ das Gewicht zur Verbindung von Neuron i zu j, dann entsteht folgende Funktion:

$$a_j = g_j(\sum_i w_{i,j} a_i) \equiv g_j(in_j) \qquad [52]$$

In dieser Funktion ist g_j die nicht-lineare Aktivierungsfunktion, die mit dem Neuron j assoziiert wird und in_j ist die Input Funktion, welche die gewichtete Summe der Inputs von j bestimmt. Dies ist nochmal in Abbildung 2.5 visualisiert. [23, 52]

Es gibt verschiedene Aktivierungsfunktionen, die eingesetzt werden können. Die in dieser Arbeit verwendeten, werden in Abbildung 2.6 dargestellt. [52]

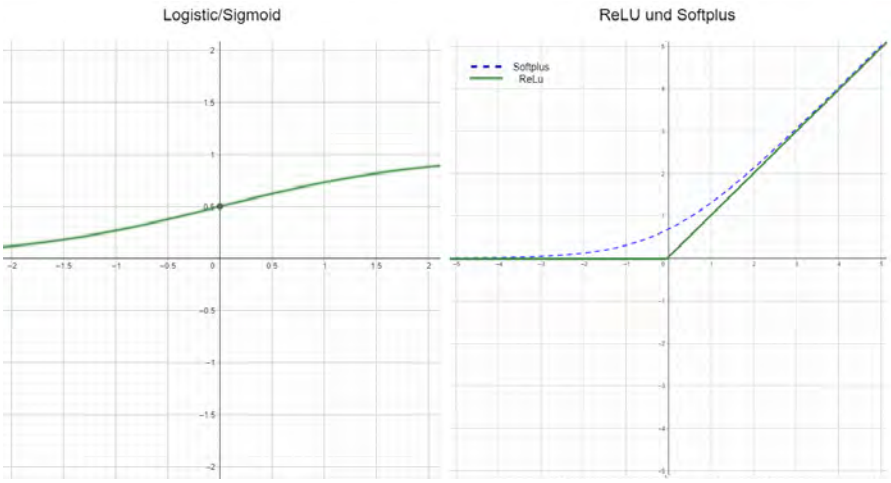

Abbildung 2.6: Grafische Darstellung der zwei verwendeten Aktivierungsfunktionen

Die in Abbildung 2.6 links dargestellte logistische oder auch Sigmoid Funktion, berechnet sich mit folgender Formel:

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \qquad [52]$$

Die Rectified Linear Unit (ReLU) Funktion ist auf der rechten Seite in der Abbildung 2.6 dargestellt und wird wie folgt berechnet:

$$ReLU(x) = max(0, x) \qquad [52]$$

2.5.2 Loss-Funktion

Um die Vorhersage des Netzes zu bewerten, wird klassischerweise nicht der erwartete Erfolg maximiert, sondern es wird versucht den Fehler (Loss) zu minimieren. Die Loss-Funktion ist hierbei die Funktion $L(y, \hat{y})$, welche die Differenz zwischen dem richtigen Ergebnis y und dem vom Model vorhergesagten Ergebnis $\hat{y}$ bestimmt. [52] Abhängig von der Aufgabe des Netzes können verschiedene Funktionen eingesetzt werden. Im Folgenden sind der Mean Squared Error (MSE) und der Mean Absolute Error (MAE) aufgelistet, da diese im Rahmen dieser Arbeit angewendet werden. Es gibt jedoch auch viele andere Metriken, wie beispielsweise die Cross Entropy, welche eher zur Klassifikation eingesetzt wird, oder Abwandlungen von den hier aufgezeigten, wie der Root Mean Squared Error. [24, 47] Eine genauere Erläuterung würde den Umfang dieser Arbeit überschreiten. Für weitere Informationen über verschiedene Loss-Funktionen empfiehlt sich das Werk [47] von J. Patterson and A. Gibson (2017).

MSE und MAE berechnen sich wie folgt:

1. Mean Squared Error (MSE):

$$L(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \qquad [47]$$

2. Mean Absolute Error (MAE):

$$L(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i) \qquad [24]$$

In beiden Formeln steht m für die Menge an Daten im Datensatz. $\hat{y}$ ist der vom Netz vorhergesagte Wert und y ist der tatsächliche Wert. [24, 20, 47]

2.5.3 Optimierungsfunktion

Basierend auf dem Fehler muss das Netz optimiert werden. Hierfür wird eine Optimierungsfunktion genutzt, welche versucht, die einzelnen Gewichte des Netzes optimal zu

bestimmen. [20] Dabei gilt bei einer Trainingsmenge $D = \{X; Y\}$, ist das Ziel den summierten quadratischen Fehler $F(W)$ des Outputs zu minimieren.

$$F(W) = \sum_{y_D \epsilon Y} (\hat{y}_D - y)^2 \qquad [20]$$

Hierbei muss beachtet werden, dass y nur von den Gewichten W abhängig ist. Anschließend kann versucht werden, dieses Optimierungsproblem mithilfe unterschiedlicher Funktionen zu lösen. Der einfachste Weg ist ein Gradientenabstiegsverfahren zu verwenden. Dies beruht darauf, dass der Gradient einer Funktion in Richtung des steilsten Anstiegs bzw. des steilsten Abstiegs zeigt. Die Schwierigkeit besteht darin, nicht in einem lokalen Minimum zu landen, sondern das globale Minimum zu finden, wie in Abbildung 2.7 dargestellt. Dieses Problem betrifft alle Optimierungsfunktionen. [20]

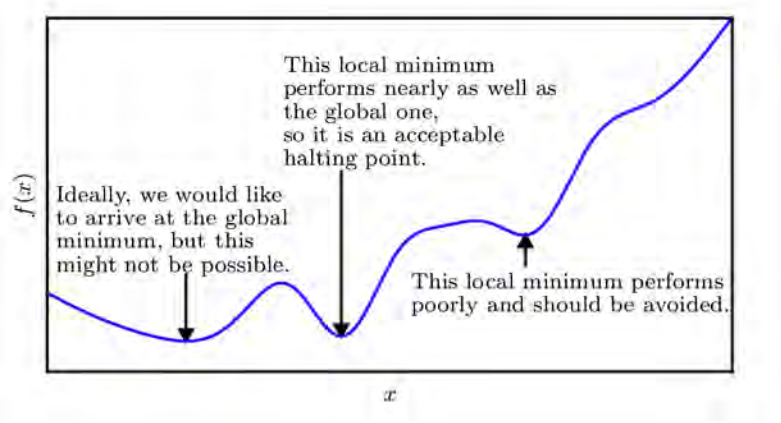

Abbildung 2.7: Optimierungsfunktionen stehen vor der Schwierigkeit das globale Minimum zu finden und nicht bei einem Lokalen zu enden. [23]

Basierend auf dem Gradientenabstiegsverfahren ist der grundlegende Optimierer der Stochastic Gradient Descent (SGD), der im Bereich des Machine Learning Anwendung findet. Auf diesem basierend existieren zahlreiche verschiedene Weiterentwicklungen. Dazu gehört auch der Adaptive Moment Estimation (Adam) Optimierer und Adadelta, welche verwendet wurden. Hierbei ist Adam der Optimierer der aufgrund seiner guten Heuristik häufig verwendet wird. [20] Auf die konkreten Unterschiede und mathematischen Grundlagen dieser Algorithmen wird hier nicht weiter eingegangen und kann in [23] nachgelesen werden. Der primäre Unterschiede ist, dass Weiterentwicklungen, wie Adam oder Adadelta, im Gegensatz zu SGD eine adaptive Lernrate haben, die während des Trainings angepasst wird. [23]

Die Lernrate bestimmt wie stark die Parameter angepasst werden. Ist sie zu hoch, kann es passieren, dass das globale Minimum verfehlt wird. Ist sie jedoch zu klein, kann es sein, dass man in einem lokalen Minimum oder auf einem Plateau hängen bleibt. [47]

2.5.4 Layer Typen

Die verschiedenen Konfigurationseinstellungen, die innerhalb des Netzes getätigt werden und dessen Performence beeinflussen, werden als Hyperparameter bezeichnet. [47] Dazu gehören beispielsweise die Größe der Ebenen, die gewählten Aktivierungs- oder Loss-Funktion. Zudem gibt es verschiedene Arten von Schichten, die innerhalb des Netzes genutzt werden können. Die grundlegendste Variante ist die vollständig verbundene Schicht, welche auch in dem zu Beginn erwähnten vorwärtsgerichteten Netz verwendet wird. Weitere für diese Arbeit relevante Schichten werden im Folgenden kurz erläutert. Hierbei sind die Convolutional Layers neben den vollständig verbundenen Schichten diejenigen, die Neuronen enthalten und somit für das klassische Lernen verantwortlich sind. Die anderen Schichten sind lediglich Zusätze, die für ein besseres Lernen im Allgemeinen sorgen sollen, aber selbst nicht lernen können.

Convolutional Layer

Die Convolutional Layer (Faltungsschichten) bilden die Grundlage von Convolutional Neural Networks (CNN). Auch CNNs sind vorwärtsgerichtete Netzwerke. Zudem ist die grundlegende Funktionsweise die gleiche wie bei Netzen aus vollständig verbundenen Schichten. Der Unterschied besteht darin, dass bei Faltungsschichten nicht jedes Neuron mit jedem Neuron der nachfolgenden Schicht verbunden ist, sondern nur lokale benachbarte Neuronen. Es wird eine Faltung genutzt, um eine Ebene auf die nächste abzubilden. CNNs werden beispielsweise verwendet, wenn der Input aus mehrdimensionalen Daten besteht, wie zum Beispiel Bilder oder Zeitreihen. Denn neben dem Aspekt, dass es schwierig ist strukturierte Daten wie ein Bild in einen eindimensionalen Vektor zu transformieren, da die Anordnung von Pixeln für den Inhalt des Bildes sehr relevant ist, haben vollständig verbundene Netze den Nachteil, dass die Anzahl der Gewichte bei großen Inputs schnell ansteigt. Ein Bild welches aus n Pixeln besteht und bei dem die erste verborgene Schicht auch n Neuronen enthält, hat bereits eine Anzahl von n^2 Gewichten. Entsprechend hat ein Netz bei einem typischen Ein-Megapixel RGB-Bild bereits 9×10^{12}

Gewichte. Da bei Faltungsschichten keine vollständigen Matrix-Vektor-Multiplikationen durchgeführt werden, sondern Faltungen, entsteht dieses Problem dort nicht. [52]

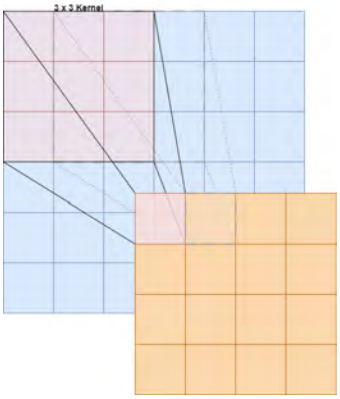

Abbildung 2.8: Visualisierung einer zweidimensionalen Faltung, mit einem 3x3 Kernel, der um eins weiter geschoben wird

Zwischen den Faltungsschichten werden nur lokal benachbarte Neuronen mithilfe eines Musters an Gewichten verknüpft, das auf die gesamte Fläche wiederholt angewendet wird. Das Gewichtsmuster, das mehrmals angewendet wird, wird als *Kernel* und das Anwenden des Kernels auf die Pixel des Bildes wird als *Faltung* bezeichnet. Dies ist in Abbildung 2.8 vereinfacht visualisiert. Dort wird ein Kernel der Größe 3x3 (in Rot) auf die 6x6 große Ebene (in Blau) angewendet. Hierbei wird der Kernel auf das entsprechende Feld in der nächsten Ebene abgebildet. So wird über das gesamte Bild iteriert. [52]

Für die mathematischen Grundlagen wird zur Vereinfachung ein eindimensionaler Datensatz betrachtet. Der Input Vektor x hat eine Größe von n und es wird ein Kernel k der Größe l genutzt. Dann lässt sich die Faltungsoperation $z = x * k$ wie folgt definieren:

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-(l+1)/2} \qquad [52]$$

Das bedeutet, für jede Output Position i wird das Skalarprodukt zwischen dem Kernel k und einem Ausschnitt aus x berechnet. Diese Berechnung ist in Abbildung 2.9 visualisiert.

In Abbildung 2.9 wird der Kernel immer um zwei Einheiten weiter bewegt. Diese Distanz zwischen den Kernel-Mittelpunkten nennt sich *Stride* und beträgt in diesem Fall s = 2. Dadurch wird auch die Anzahl der Einheiten in der nachfolgenden Ebene reduziert. Dies

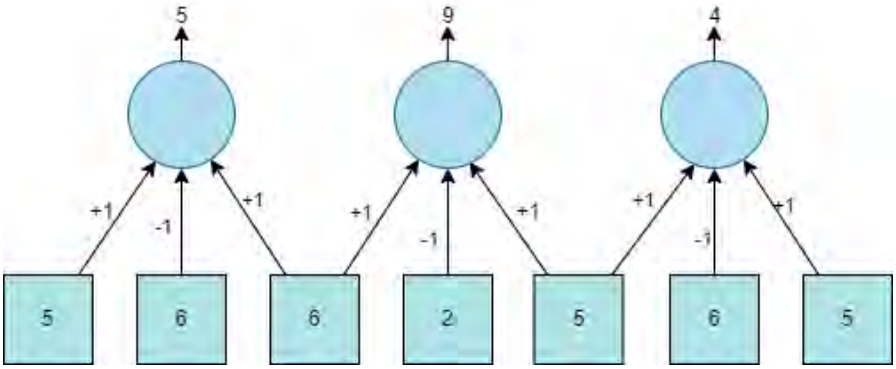

Abbildung 2.9: Visualisierung einer Faltung am Beispiel eines eindimensionalen Datensatzes. Eigene Darstellung basierend auf [52]

gilt nur solange an den Rändern des Bildes gestoppt wird. Es besteht jedoch auch die Möglichkeit, die Ränder der Ebene aufzufüllen. Hierbei können beispielsweise Nullen oder Kopien der Randwerte verwendet werden. [52] Dieser Wert nennt sich *Padding* und er kann entsprechend genutzt werden, um die Anzahl der Einheiten nicht zu reduzieren oder nicht so stark zu reduzieren, wie in Abbildung 2.10 dargestellt. Wenn das Padding immer so gewählt wird, dass die ursprüngliche Größe beibehalten wird, nennt sich das Zero-Padding. [47]

Im Gegensatz zu vollständig verbundenen Netzen muss bei CNNs beachtet werden, dass nicht alle Neuronen zwischen den Schichten verbunden sind. Die einzelnen Neuronen haben nur ein beschränktes Empfangsfeld, abhängig von der Kernelgröße. Die Menge an Inputs, die ein Neuron aktivieren kann, wird als *Receptive Field* bezeichnet. Das Receptive Field ist bei CNNs meist sehr klein und beträgt in der ersten verborgenen Schicht entsprechend der Kernelgröße l Pixel. In den tieferen Schichten wird dieses größer. Wenn Stride $= 1$ ist, hat ein Neuron in der m-ten verborgenen Schicht ein Receptive Field der Größe $(l-1)m+1$. Demnach benötigt es entweder entsprechend tiefere Netze, damit sich relevante Informationen aus dem Input weiter verteilen können, oder größere Kernel. [52]

Ein weiterer Ansatz, um das Receptive Field zu erhöhen, ist *Dilation*. Bei Dilation wird bei der Faltung nicht ein zusammenhänges Fenster genommen, sondern der Kernel wird gespreizt, sodass nur jedes n-te Feld bei einer Dilation von n genommen wird. [41] In Abbildung 2.11 ist beispielhaft eine Dilation von 2 dargestellt.

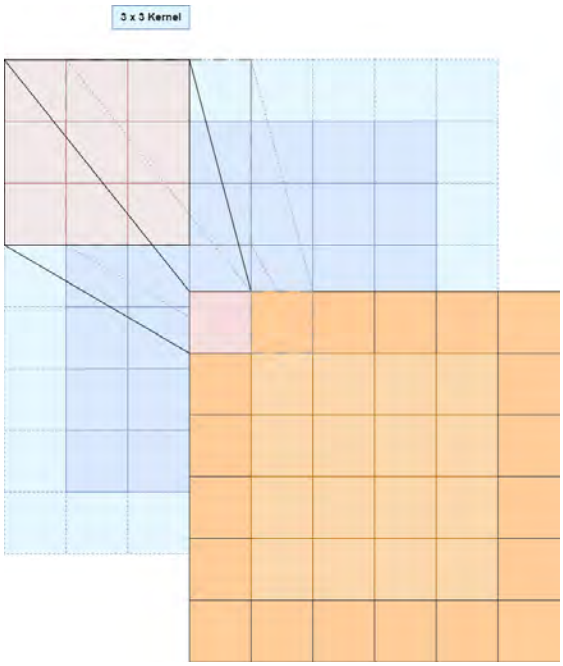

Abbildung 2.10: Visualisierung einer Faltung mit Padding

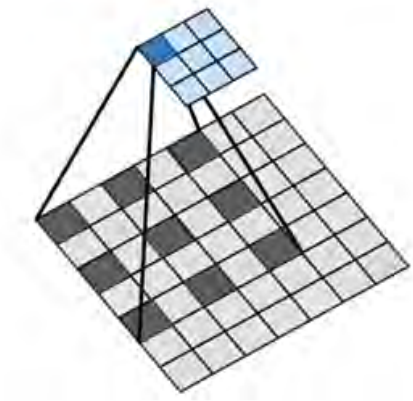

Abbildung 2.11: Visualisierung einer Faltung mit einer Dilation von 2 [41]

In einem CNN können neben Faltungsschichten auch vollständig verbundene Schichten verwendet werden. Es ist üblich mit einigen Faltungsschichten zu starten, die Auflösung mithilfe einiger Pooling Schichten zu reduzieren und am Ende noch vollständig verbundene Schichten zu verwenden. [52]

Pooling Schichten

Mithilfe von Pooling Schichten kann die Auflösung in einem CNN reduziert werden. Dabei werden lokale Gruppen zu einem Wert zusammengefasst, indem, abhängig von der gewünschten Zielgröße, mit einem entsprechenden Kernel über die Daten gegangen wird und die Werte zu einem zusammengefasst werden. Es gibt verschiedene Arten den zusammengefassten Wert zu wählen. Es kann entweder das Maximum, der Durchschnitt oder ein Zufallswert genommen werden. [52]

Batch Normalization

Batch Normalization ist eine Technik, die darauf abzielt, sicherzustellen, dass der Optimierer eine bessere Konvergenz erreichen kann. Hierfür werden die generierten Werte der vorliegenden Schicht innerhalb jedes Minibatches neu skaliert. Dabei standardisiert Batch Normalization den Mittelwert und die Varianz der Werte. [52]

Dropout

Dropout dient hauptsächlich dazu Overfitting zu vermeiden, indem es das Lernen der Trainingsdaten erschwert. Bei jedem Trainingsschritt deaktiviert Dropout zufällig einen Teil der Verbindungen. Dadurch wird das Netz gezwungen mehrere Wege für die gleichen Input Daten zu lernen, was zu einer besseren Generalisierung führt. Allerdings hat dies auch den Nachteil, dass das Training schwieriger wird, was dazu führen kann, dass ein größeres Netz oder längeres Training erforderlich ist. [52]

2.5.5 Autoencoder Architektur

Basierend auf den verschiedenen Komponenten und Parametern haben sich viele verschiedene übergeordnete Architekturen entwickelt, die für verschiedene Aufgaben geeignet sind. Eine davon ist die Autoencoder Architektur. Der klassische Autoencoder besteht aus zwei Teilen. Dem Encoder, der die Input Daten x auf $\hat{z}$ abbildet, und einem Decoder, der die Repräsentation $\hat{z}$ wieder zurück zu x umwandeln soll, wie in Abbildung 2.12 zu sehen ist. Im Allgemeinen ist der Encoder nur eine parametrisierte Funktion f und der Decoder ist eine parametrisierte Funktion g. Das Modell ist dann trainiert, sodass $x \approx g(f(x))$, d.h., der Encoding-Prozess wird quasi vom Decoding-Prozess invertiert. Das Ziel dabei ist, dass $\hat{z}$ so viele Informationen wie möglich komprimiert behält, sodass x wiederhergestellt werden kann. [52]

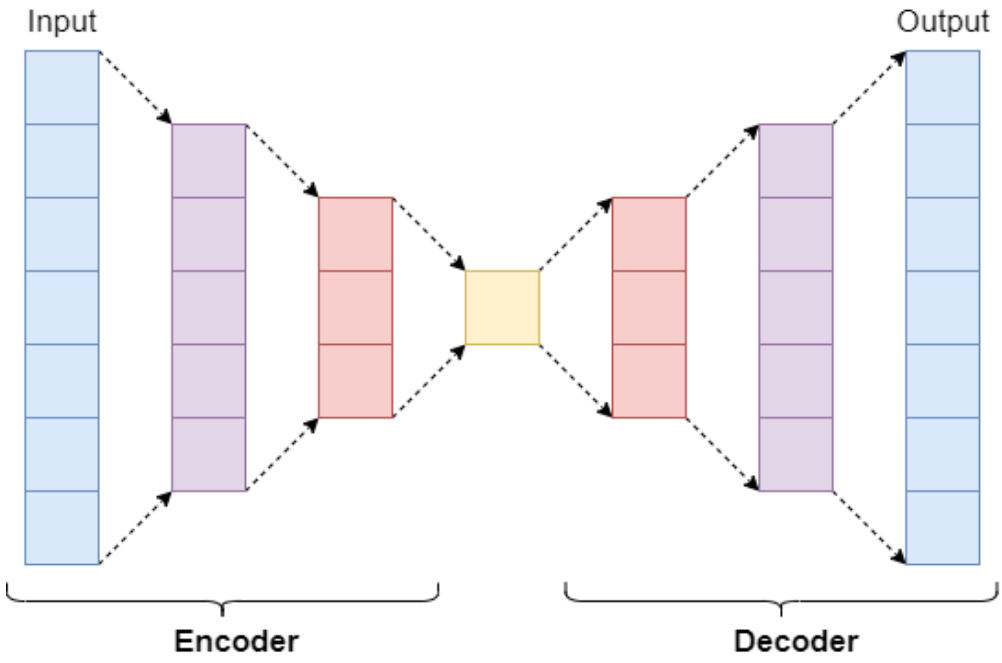

Abbildung 2.12: Visualisierung einer Autoencoder Architektur

Diese Architektur findet in vielen Bereichen Anwendung. Beim unüberwachten Lernen wird sie beispielsweise für die Anomalieerkennung verwendet, indem angenommen wird, dass Datensätze, die der Decoder nicht rekonstruieren kann, Ausreißer sind. [36] Ein weiteres Anwendungsfeld für Autoencoder ist die Bildverarbeitung. Hierbei werden sie meist in abgewandelter Form mit Faltungsschichten genutzt. [12, 4, 9] Mehr Details zu Autoencodern können in dem Buch [9] von L. Rokach et al. nachgeschlagen werden.

2.6 Reinforcement Learning

Das *Reinforcement Learning* (deu.: Verstärkendes Lernen) gehört neben dem *Supervised Learning* (deu.: überwachtes Lernen) und dem *unsupervised Learnig* (deu.: Unüberwachtes Lernen) zu den drei Arten des Lernens, die im Maschinellen Lernen eingesetzt werden. Während beim *überwachten Lernen* ein Datensatz existiert, bei dem zu jedem Input der Output bekannt ist und mit dem dann trainiert wird, ist dies beim *unüberwachtem Lernen* nicht der Fall. Hier geht es darum, dass das Modell eigenständig Muster findet. [56]

Das *Reinforcement Learning* (RL) hingegen ist dem natürlichen Lernen, wie beispielsweise bei Kindern, nachempfunden. Eine Handlung wird ausgeführt und durch Feedback kann die Handlung beurteilt und etwas gelernt werden. Dieses Feedback nennt sich beim RL *Reward* (Belohnung). Dabei bewertet die Umgebung die getätigte Aktion oder Aktionen. Der Agent versucht diese Belohnung zu maximieren, um dadurch die bestmöglichen Entscheidungen zu treffen. Dabei wird nicht vorgegeben, welche Aktion ausgeführt werden soll, sondern der Agent lernt durch Ausprobieren und Erfahrung. Hierbei entsteht ein Trade-off bei der Aktionswahl zwischen der Maximierung des Gewinns und Erforschen neuer Optionen. [56] Auf diesen Trade-off wird in Abschnitt 2.6.1 genauer eingegangen.

Der Agent kennt die Umwelt und kann mit dieser interagieren. Basierend auf dem aktuellen Zustand entscheidet der Agent, welche Aktion ausgeführt werden soll, führt diese dann aus und erhält dann entweder direkt oder nach dem Ausführen mehrerer Handlungen eine Belohnung. Neben der Belohnung, die jede einzelne Handlung im Hinblick auf ihre direkte Qualität bewertet, gibt die *Value-Funktion* (Wert-Funktion) den langfristigen Wert einer Handlung zur Erreichung des übergeordneten Ziels an. Dadurch kann ein spezieller Zustand zwar eine kleine Belohnung haben, aber dennoch einen hohen Wert, da ihm meist ein Zustand mit sehr hoher Belohnung folgt. Diese Wert-Funktion wird anschließend von einer Policy genutzt, um zu entscheiden, welche Handlung ausgeführt wird. Eine *Policy* ist somit quasi eine Verknüpfung zwischen Eingangszuständen der Umgebung und Aktionen. Sie ist der Kern des Reinforcement Learning Agenten. [56]

Das letzte Element, das in manchen RL-Systemen vorkommt, ist das *Model* der Umwelt. Dieses bildet das Verhalten der Umwelt ab und ermöglicht es Schlussfolgerungen darüber zu ziehen, wie die Umgebung reagieren wird. Diese Komponente existiert nicht in jedem System, daher gibt es sowohl Model-based als auch Model-free Ansätze. Entsprechend

sind Model-free Ansätze solche, bei denen ein explizites Lernen durch Ausprobieren genutzt wird. [56]

2.6.1 Exploitation vs. Exploration Problem

Ein Problem, das beim Reinforcement Learning im Gegensatz zu den anderen Arten des Lernens existiert, ist der Kompromiss zwischen Exploitation (Ausbeutung) und Exploration (Erkundung). Es muss ein guter Mittelweg zwischen dem Ziel der reinen Maximierung der Belohnung und dem Erkunden gefunden werden. Wenn der Agent immer nur die Entscheidung trifft, die nach seinem Kenntnisstand die beste ist, führt das dazu, dass er potenziell bessere Entscheidungen, die er noch nicht kennt, gar nicht erst finden kann. Reines Erkunden wiederum kann aber ebenfalls vermeintlich schlecht sein oder auch zum Scheitern führen. Daher muss eine gute Mischung gefunden werden. [56]

In der Umsetzung wird dieser Kompromiss mithilfe eines ϵ-greedy Verfahrens abgebildet. Hierbei wird eine Variable ϵ eingesetzt, die den prozentualen Anteil an Zufallsentscheidungen definiert. Formal bedeutet das, dass im Zustand s mit der Wahrscheinlichkeit ϵ eine zufällige Aktion und mit $1 - \epsilon$ die Aktion mit dem höchsten Wert ausgeführt wird. Dadurch wird sichergestellt, dass eine Mischung aus Exploration und Exploitation vorhanden ist. Hier muss ein geeigneter Wert für ϵ gefunden werden. Üblicherweise wird dieser zu Beginn größer gewählt und dann mit fortschreitendem Training reduziert. [56, 20]

2.6.2 Deep Reinforcement Learning

Ein Teilbereich des Reinforcement Learning ist das Deep Reinforcement Learning (deep RL). Hierbei werden neuronale Netze eingesetzt, um zu entscheiden, welche Aktion gewählt wird. Dieser Ansatz hat den Vorteil, dass auch komplexe Aufgaben gelernt werden können, bei denen beispielsweise die Umgebung sehr komplex ist und dadurch ein großer Zustandsraum entsteht. Dafür gibt es verschiedene Architekturen, die genutzt werden können. Eine sehr häufig eingesetzte ist das Deep Q-Network (DQN). Ein DQN gibt, basierend auf dem Input State, die Q-Values aus. Darauf basierend kann entschieden werden, welche Aktion am geeignetsten ist. Diese Architektur eignet sich, wenn ein Netz benötigt wird, das zwischen mehreren Aktionen wählen soll. [56]

Da eine detaillierte Einführung ins deep RL den Rahmen dieser Arbeit überschreitet, können weitere Erläuterungen zum Thema RL in [56] von R.S. Sutton et al. (2018) nachgelesen werden.

2.7 Potentialfelder

Der Ursprung von Potentialfeldern liegt in der Robotik, wo sie verwendet werden, um eine globale Darstellung des Raumes zu erhalten. Sie unterstützen die Einschätzung von Entfernungen und Formen von Hindernissen, was es Robotern ermöglicht, effizientere Routen zu planen. Zudem erleichtern sie die Kollisionsvermeidung, da aufwendige Berechnungen zur Schnittpunkterkennung auf Grundlage von geografischen Darstellungen vermieden werden. Dies geschieht durch eine geeignete Definition des Potentials eines Punktes wodurch bereits irrelevante Hindernisse für diesen Ort eliminiert werden. [30]

Das konkrete Erscheinungsbild und die genutzten Algorithmen können variieren, aber im Allgemeinen handelt es sich um Karten, in denen verschiedene Bereiche unterschiedliche Potentiale aufweisen. Diese Potentiale wirken entweder anziehend oder abstoßend auf die Roboter, wodurch Kollisionen mit Hindernissen verhindert werden sollen. [32, 30] Formal lässt sich dies wie folgt abbilden:

$$U(p) = U_{att}(p) + U_{rep}(p) \qquad [32]$$

Hierbei ist p der entsprechende Punkt auf der Karte und U_{att} und U_{rep} beschreiben die anziehenden sowie die abstoßenden Kräfte des Potentials, welches auf diesen Punkt einwirken. [32]

2.8 Verwandte Forschungsarbeiten

Da nach bestem Wissen der Autorin keine wissenschaftlichen Arbeiten existieren, die zur Generierung von Potentialfeldern auch auf Reinforcement Learning (RL) zurückgreifen, lassen sich die verwandten Arbeiten in zwei Hauptbereiche unterteilen. Zum einen gibt es Arbeiten, die sich mit derselben Domäne befassen, auf die in Abschnitt 2.8.1 näher eingegangen wird. In diesem Kontext gibt es auch einige Publikationen, die sowohl RL als

auch Potentialfelder nutzen. Zum anderen gibt es Arbeiten, die vergleichbare Methoden einsetzen wie in dieser Arbeit. Diese werden in Abschnitt 2.8.2 genauer betrachtet.

2.8.1 Verwandte Arbeiten zur Thematik Wegefindung in autonomen Systemen

Immer mehr Arbeiten nutzen Methoden des Maschinellen Lernens, um Pfadplanungsalgorithmen zu ersetzen oder mit ihnen zu kombinieren, um deren Nachteile auszugleichen. Davon auch einige, die sich mit der Steuerung von unbemannten Flugobjekten beschäftigen.

B. Li und Y. Wu (2020) [40] haben einen Ansatz entwickelt, der für die Verfolgung von Bodenzielen mit Hindernissen durch Drohnen genutzt wird. Die Herausforderung besteht hierbei in der dynamischen Umgebung und dem Verfolgen von bewegten Zielen. Hierfür wird ein verbesserter Deep Deterministic Policy Gradient (DDPG) Algorithmus als Basis verwendet. Damit die Drohnen das Verhalten des Target Tracking und gleichmäßige Flugbahnen lernen, werden für die Belohnungsfunktion mathematisch berechnete Potentialfelder genutzt. Das Training findet in einer virtuellen Umgebung statt.

Eine weitere Arbeit, die RL und Potentialfelder für die Drohnensteuerung nutzt, stammt von Kong et al. (2023) [37]. Hierbei müssen Drohnen in einer gridbasierten Umgebung vom Start zum Ziel fliegen und dabei Hindernissen ausweichen. Die Autoren entwickeln eine Steuerung, die auf künstlichen Potentialfeldern und Deep Q-Networks (APFDQN) basiert, und vergleichen diese mit einer Steuerung, die auf einem klassischen Deep Q-Network (DQN) basiert. Das DQN der APFDQN-Steuerung gibt neben den Q-Werten, für die verschiedenen Richtungen, auch die Wahrscheinlichkeitsverteilung der Aktionen aus. Der Algorithmus führt zusätzlich eine SA-ϵ-Greedy-Strategie für die Balance zwischen Exploration und das Exploitation ein. Im APFDQN-Ansatz wird für die Wahl der Aktion auch das mathematisch berechnete Potentialfeld genutzt. Dabei wird die Abweichung zwischen der Vorhersage des Netzes und dem Potentialfeld bestimmt. Ist diese Abweichung größer als ein gewisser Schwellenwert, wird basierend auf dem Potentialfeld gehandelt. Dadurch wird das Potentialfeld als Vorwissen eingebunden, wodurch weniger Erforschen nötig ist und der Algorithmus schneller konvergiert. In den Experimenten zeigt sich, dass der APFDQN-Ansatz besser lernt und eine höhere Erfolgsrate als das klassische DQN erzielt.

Neben den Arbeiten, die Potentialfelder nur als Unterstützung für das RL nutzen, verfolgt die Arbeit von Q. Yao et al. (2020) [61] einen alternativen Ansatz. Der Anwendungsfall und der Simulationsaufbau ähneln dem von Kong et al. (2023) [37], jedoch handelt es sich hier nicht um Drohnen, sondern um Agenten in Warenhäusern. Auch hier müssen die Agenten in einer gridbasierten Umgebung ein Ziel erreichen und dabei Hindernissen ausweichen. Im Gegensatz zu den vorherigen Arbeiten liegt der Fokus jedoch eher auf den Potentialfeldern. Es wird sich mit der Problematik von lokalen Stabilitätspunkten beschäftigt, bei denen die Agenten basierend auf den Potentialfeldern handlungsunfähig werden, da sie ein lokales Minimum erreicht haben. Um mit diesem Problem umzugehen, stellen sie eine verbesserte Berechnung der Potentialfelder vor, die sie Black-Hole Potential Field nennen. Diese stellen die Umgebung dar, in der der Agent agiert. Die Potentialfelder werden als Input für ein DQN genutzt, welches über die Bewegungsrichtung entscheidet. Durch die Kombination aus der verbesserten Berechnung und dem DQN soll das Problem der lokalen Stabilitätspunkte gelöst werden. Am Ende zeigt dieser Ansatz eine sehr gute Leistung und kann sogar mit dynamischen Hindernissen umgehen.

Ein weiteres Feld, das sich intensiv mit der Thematik der Wegfindung und dem Einsatz von Potenzialkarten befasst, ist der Bereich der unbemannten Wasserfahrzeuge. S. Guo et al. (2020) [25] nutzen wie Kong et al. (2020) [40] einen DDPG-Ansatz als Basis. Auch in dieser Arbeit wird das Potentialfeld als Vorwissen verwendet und unterstützt die Entscheidungsfindung. Hierbei wird betont, dass durch den Einsatz von Potentialfeldern die Trainingszeit verkürzt wird und das Netz schneller konvergiert. Einen ähnlichen Ansatz verfolgen auch C. Wang et al. (2019) [16] in ihrer Arbeit. Der Unterschied besteht hierbei nur darin, dass ein DQN eingesetzt wird. Dennoch wird auch hier das Potentialfeld als Vorwissen genutzt.

Im Bereich der Steuerung unbemannter Wasserfahrzeuge haben auch X. Wang et al. (2022) [59] eine interessante Arbeit veröffentlicht. In dieser wird kein Reinforcement Learning genutzt, sondern die Potentialfelder mit einer Model Predictive Control (MPC) kombiniert. Der Fokus ist hierbei etwas konkreter auf den Umgang mit Winkelgeschwindigkeiten bei der Steuerung ausgerichtet, als nur auf die allgemeine Pfadplanung. Aber vergleichbar mit dem eigenen Anwendungsfall, werden auch hier Potentialfelder als Basis für die übergeordnete Wegentscheidung genutzt. Die MPC nimmt diese Entscheidung dann lediglich als Basis, um die konkreten Steuerbefehle zu bestimmen.

Eine andere Arbeit, die sich mit der Pfadplanung im Allgemeinen beschäftigt, stammt von J. Wang et al. (2020). [58] Auch in dieser Arbeit soll der Agent an Hindernissen

vorbei zu einem Ziel manövrieren. Der vorgestellte Ansatz ist eine Weiterentwicklung des Rapidly Exploring Random Trees (RRT) Algorithmus. Da dieser eine sehr langsame Konvergenz zu optimalen Pfaden aufweist und dadurch erhöhter Speicher- und Zeitverbrauch entsteht, wird der Neural RRT (NRRT) Algorithmus vorgestellt. Es wird mithilfe überwachtem Lernen ein tiefes Neuronales Netz trainiert. Dieses gibt die Wahrscheinlichkeitsverteilung der optimalen Pfade für die gegebene Aufgabe aus. Das Netz wird mithilfe von Karten und optimalen Pfaden, die mithilfe des A* Algorithmus berechnet werden, trainiert. Dieser Ansatz führt zu bessere Ergebnisse als herkömmliche Verfahren.

Nicht nur im Bereich der Pfadplanung finden Potentialfelder Anwendung. H. Jiang et al. (2023) [32] nutzen diese Kombination für das Spielen des Computerspiels StarCraft II. Hierbei wird das Potentialfeld auf Bildern basierend berechnet. Dieses wird für den aktuellen Zustand und für die Belohnungsfunktion verwendet. Der entwickelte Ansatz weist dabei deutliche Vorteile gegenüber bekannten Alternativen auf.

2.8.2 Verwandte Arbeiten mit Fokus auf den verwendeten Methoden des Maschinellen Lernens

Neben den Arbeiten, die sich mit der Thematik der Steuerung und Pfadplanung befassen, gibt es auch Arbeiten, die im Hinblick auf die eingesetzten Methodiken des Maschinellen Lernens verwandt mit dieser Arbeit sind.

Viele Ansätze jedoch, die RL mit Bildverarbeitung kombinieren, betrachten bei der Belohnung das gesamte Bild. R. Furuta et al. (2020) [21] haben sich hingegen mit einem Vorgehen für pixelweise Belohnungen beschäftigt. Dabei nutzen sie pro Pixel einen Agenten, der das optimale Verhalten lernen soll. Hierbei haben sie sich damit beschäftigt wie ein solcher Ansatz auch bei einer hohen Bildauflösung und entsprechend vielen Agenten möglich sein kann. Der Ansatz wurde mit verschiedenen Methoden zur Bildverarbeitungen wie Bildentrauschung, Farbverbesserung oder Bildrestauration, getestet. Der entwickelte Ansatz hat in allen Bereichen vergleichbare oder bessere Ergebnisse erzielt.

Die Kombination von RL und Bildverarbeitung wird auch von G. Maicas et al. (2019) in [42] betrachtet. In dieser Publikation geht es um die Klassifikation und Identifikation der betroffenen Bereiche von Brustkrebs auf MRT-Bildern. Die Autoren vergleichen zwei Ansätze. Der Post-hoc Ansatz nimmt erst die Klassifikation vor, ob bösartiges Gewebe vorhanden ist oder nicht. Für die positiven Fälle wird anschließend in einem zweiten Schritt versucht die Position zu lokalisieren. Dies geschieht mithilfe eine Autoencoders.

Der Pre-hoc Ansatz geht andersrum vor und lokalisiert erst Läsionen und klassifiziert diese in einem zweiten Schritt als bösartig oder nicht. Für die Lokalisierung in dieser Methode wird ein RL-Ansatz vorgeschlagen, der basierend auf den Bildern die potenziellen Bereiche erkennen soll. Für diesen Ansatz werden ein DQN und ein Residual Network genutzt. Die Bewertung basiert auf der Qualität des gewählten Ausschnitts zur Erkennung von Läsionen. Ein weiteres Neuronales Netz klassifiziert anschließend die Ergebnisse. Beim Vergleich der beiden Ansätze stellte sich heraus, dass der RL-Ansatz bei der Lokalisierung der speziellen Bereiche mit bösartigem Gewebe besser ist.

Besonders im Bereich der Medizin gibt es zahlreiche Einsatzbereiche von Bildverarbeitung und davon auch Ansätze mit RL. [41, 48, 62] Hierbei ist es oft schwierig gelabelte Datensätze zu erstellen, wodurch sich besonders Alternativen zum überwachtem Lernen eignen, wie in [15].

M. Chen et al. (2021) zeigen dabei in [15], dass sich Convolutional Autoencoder für die Bildverarbeitung eignen. In ihrer Arbeit geht es um die Klassifizierung von Lungenknoten auf CT-Bildern. Da kaum gelabelte Daten vorhanden sind, nutzen sie hierfür eine Kombination aus unüberwachtem und überwachtem Lernen. Hierbei wird das Netz primär mit unüberwachtem Lernen trainiert, um in zwei Klassen zu klassifizieren. Mithilfe der gelabelten Daten wird dann noch das Fine-Tuning durchgeführt.

Auch S. Karimpoulia und P. Tahmasebib (2019) nutzen in [34] Convolutional Autoencoder. In diesem Fall, um Gestein zu klassifizieren. Hierbei ist die Herausforderung, dass auf den Graustufenbildern die Klassifikation oft schwierig ist. Daher werden zwei verschieden große Convolutional Autoencoder trainiert, um die Segmentierung auf den Bildern zu verbessern. Da nur wenige Input Daten vorhanden sind, wird Augmentation angewendet, um die Überanpassung des Netzes zu verhindern. Mit dem Verfahren werden gute Ergebnisse erzielt.

Die Eignung von Convolutional Autoencodern zeigen zudem A. Azarang et al. (2019) in [4]. Die Autoren haben eine Convolutional Autoencoder Architektur genutzt, um die Bildqualität von Satellitenbildern zu verbessern. In ihrer Arbeit vergleichen sie ihren Ansatz mit anderen bekannten Methoden und belegt damit die Effektivität ihres Ansatzes.

3 Technischer Aufbau

In diesem Kapitel werden die verschiedenen Komponenten, die erstellt wurden und für die Anwendung relevant sind, genauer beschrieben. Eine Übersicht über die relevanten Komponenten befindet sich in Abbildung 3.1. Da mit verschiedenen Programmiersprachen gearbeitet wurde, ist der Mittelpunkt ein Ordner, in dem alle relevanten Daten liegen. Dieser stellt die Schnittstelle zwischen den verschiedenen Komponenten dar. In dem Ordner sind sowohl alle Eingangsdaten als auch die generierten Daten enthalten und er wird genutzt, um das Training und die Simulationen synchron zu halten. Die Datenstruktur sowie die Aufbereitung und Bereinigung der Daten werden in Abschnitt 3.1 genauer erläutert.

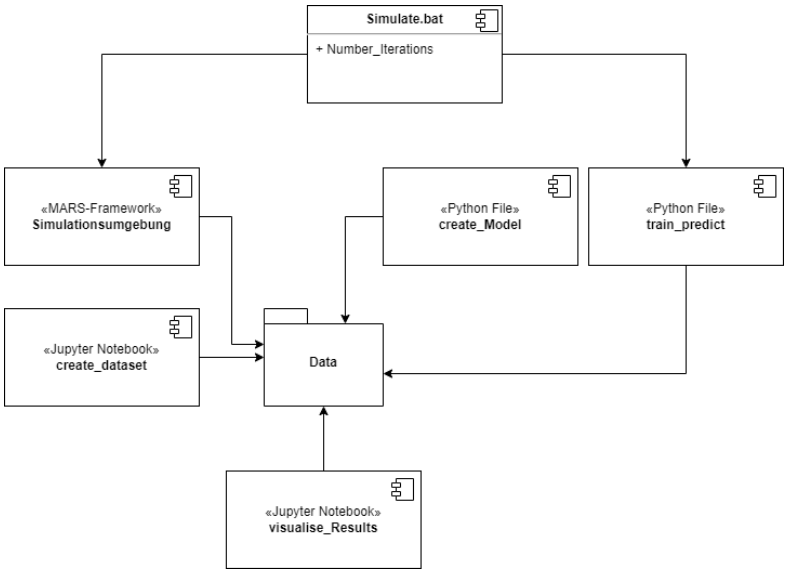

Abbildung 3.1: UML-Komponentendiagramm zur allgemeinen Struktur

Diese Daten werden mithilfe von Python und Tensorflow genutzt, um ein neuronales Netz zu trainieren und Potentialfelder zu generieren, wie in Abschnitt 3.2 beschrieben ist. Die generierten Potentialfelder werden mithilfe von Simulationen bewertet. Hierfür

wird die mit dem MARS Framework erstellte Simulationsumgebung genutzt, auf die in Abschnitt 3.3 genauer eingegangen wird. Um diesen Ablauf über mehrere Iterationen zu automatisieren, wird ein Batch-Skript genutzt, das abwechselnd das Training und die Simulationen startet. Zuletzt werden Kepler GL und Jupyter Notebook genutzt, um die Ergebnisse zu visualisieren und auszuwerten.

3.1 Datensatz

Der Datensatz besteht aus verschiedenen Karten, welche alle Informationen enthalten, die für die Drohnen für die spätere Wegefindung relevant sind. Für das Zusammenstellen und Aufbereiten des Datensatzes wurde Python verwendet. Hierbei werden mehrere zufällige Koordinatenpunkte bestimmt. Basierend auf diesen wird ein Rechteck erstellt welches immer die gleichen Maße hat. Dieser Ausschnitt wird anschließend mit allen Kartentypen abgebildet. Ein Beispieldatensatz ist in Abbildung 3.2 zu sehen. Hier sind die in Rahmen dieser Arbeit verwendeten vier verschiedenen Kartentypen abgebildet. Auf diese wird im Folgenden genauer eingegangen. Diese vier Input Daten stellen hierbei lediglich mögliche Input Daten dar je nach Anwendungsfeld könnten auch andere Informationen als Input genutzt werden.

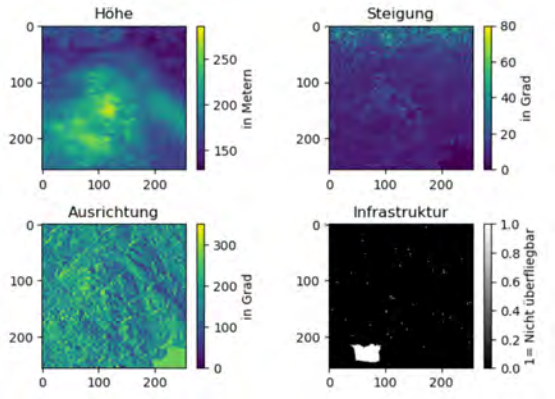

Abbildung 3.2: Beispiel eines Input Datensatzes mit DEM, Slope, Aspect und der Infrastrukturkarte

3.1.1 Höhendaten

Die Grundlage des genutzten Datensatzes bildet das Digital Elevation Model (DEM). Diese Daten stammen von der Shuttle Radar Topography Mission (SRTM), einer gemeinsamen Mission der NASA, der National Geospatial-Intelligence Agency sowie der deutschen und der italienischen Raumfahrtagentur im Jahr 2000. Durch die Verwendung von zwei Radarantennen konnten für fast 80 % der Erdoberfläche topografische Daten erfasst werden. Die gesammelten Daten dienten zur Erstellung gerasteter Karten des erforschten Gebiets. Der Datensatz steht mit einer Auflösung von einer Bogensekunde öffentlich zur Verfügung, was ihn zu einem der detailliertesten DEMs der Erde macht. Diese Auflösung entspricht in Äquatornähe sowohl bei den Breiten- als auch bei den Längengraden einer Entfernung von 30 Metern. Die Höhenangaben in diesem Datensatz sind in Metern angegeben. [19, 53]

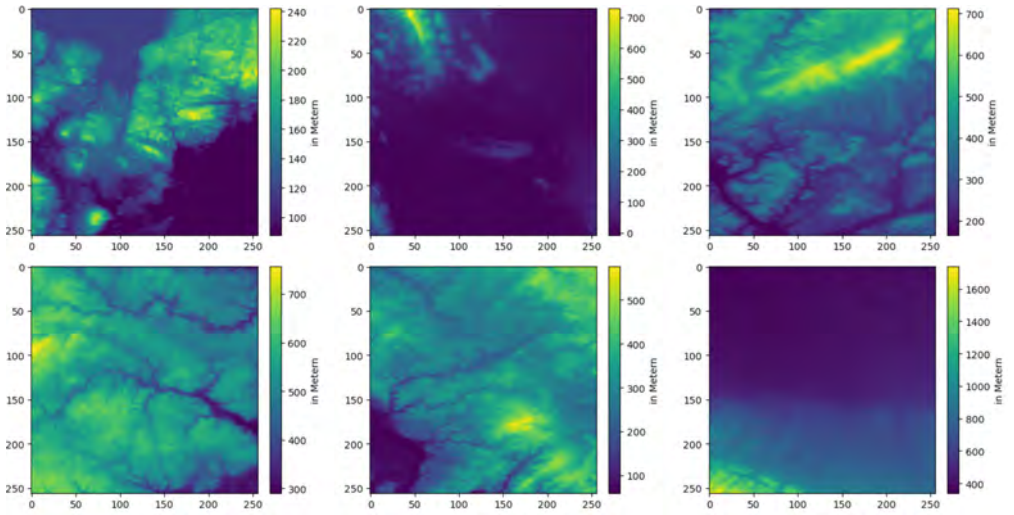

Abbildung 3.3: Verschiedene Höhendaten Beispiele

Um diese Daten herunterzuladen, wird die Elevation-Bibliothek in Python genutzt. Hierbei können über Längen- und Breitengrade definierte Quadrate aus diesem Datensatz in Form einer Rasterdatei heruntergeladen werden. [5] Diese dienen als Basis für die restlichen Daten. In Abbildung 3.3 sind beispielhaft ein Auswahl an Karten zu unterschiedlichen Ausschnitten der Welt abgebildet, die für das Training benutzt werden.

3.1.2 Steigungsausrichtung (Aspect)

Basierend auf den DEMs wird die Ausrichtung der Steigung (Aspect) berechnet. Zusammen mit den Höhendaten ist diese eine wesentliche Information, um das geografische Terrain zu analysieren. Das Überfliegen steiler Anstiege verlängert die Flugstrecke und kann zu ineffizienten Routen führen, im Vergleich zu Umwegen um diese Steigungen herum. Die Berechnung der Ausrichtung erfolgt mithilfe der Richdem-Bibliothek [10], die die von Horn (1981) [29] vorgestellten Algorithmen zur Berechnung von Steigung und Aspect implementiert.

Die konkrete Implementierung ist in 3.1 dargestellt. Hierbei ist erst das Laden der Höhendaten aus dem SRTM Datensatz zu sehen und anschließend die Erstellung des zweidimensionalen Arrays mit den Steigungsausrichtungen mithilfe von Richdem.

Codeblock 3.1: Code für die Erstellung der Aspect-Karten

```
#Download SRTM data with elevation library and save the clipping as .tif file
elevation.clip(bounds= bounds, output=path)
#Load .tif with Richdem
dataset= rd.LoadGDAL(tif_path)
#create Aspect
aspect = rd.TerrainAttribute(dataset, attrib='aspect')
np_aspact=np.array(aspect)
```

Der Aspect spiegelt die Richtung der maximalen Neigung der fokalen Zelle wider und wird in Grad angegeben. In Abbildung 3.4 sind passende Beispiele zu den in Abbildung 3.3 dargestellten DEMs zu sehen. Dabei entspricht eine Ausrichtung von 0 Grad Norden. Die Skala verläuft im Uhrzeigersinn. [29, 10]

3.1.3 Steigung (Slope)

Der dritte Faktor, der für die Beschreibung des geografischen Terrains relevant ist, ist die Steigung (Slope). Auch für diese Analyse wird zunächst die Richdem-Bibliothek [10] verwendet. Der von Horn (1981) [29] vorgestellte Algorithmus berechnet die Steigung einer fokalen Zelle mithilfe einer zentralen Differenzschätzung einer an die fokale Zelle und ihre Nachbarn angepassten Fläche. Die gewählte Steigung ist das Maximum dieser Fläche und kann in verschiedenen Formaten zurückgegeben werden. [29, 10] Bei der

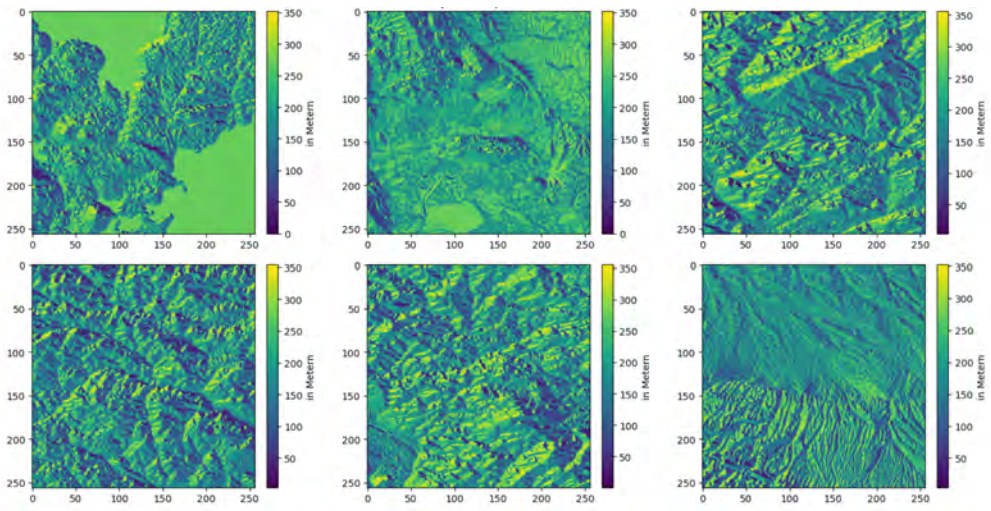

Abbildung 3.4: Verschiedene Aspect Beispiele

Implementierung von Richdem kann gewählt werden, ob die Ergebnisse in Grad, Prozent oder als Rise/Run angezeigt werden sollen.

Bei genauerer Analyse des genutzten Datensatzes zeigte sich jedoch, dass diese Implementierung für die hier verwendeten DEMs nicht geeignet ist. Dies liegt daran, dass die Richdem-Implementierung standardmäßig davon ausgeht, dass die Skalierung in allen drei Dimensionen in der gleichen Einheit vorliegt oder lediglich ein Skalierungsfaktor zum Umrechnen für die Z-Achse nötig ist. [10] In diesem Fall ist das Problem, dass die geografischen Daten in X- und Y-Richtung als Längen- und Breitengrade angegeben sind, die Höhe allerdings in Metern. Dadurch kann kein einheitlicher Skalierungsfaktor für die Z-Achse angegeben werden, da die Längengrade sich an den Polen treffen und nicht parallel, wie die Breitengrade, verlaufen. [7] Infolgedessen werden die Ergebnisse ungenau und unbrauchbar, sobald sich der Kartenausschnitt vom Äquator entfernte. Dies liegt daran, dass sich die Skalierung ändert und es keinen einheitlichen Umrechnungsfaktor für Meter in Längengrad und Meter in Breitengrad gibt. Die Folgen sind in Abbildung 3.5 visualisiert. Hierfür werden zwei Kartenausschnitte an verschiedenen Positionen auf der Erde genommen. Die oberen Bilder zeigen einen Ausschnitt auf der Nordhalbkugel und die unteren einen Ausschnitt in der Nähe des Äquator. Es wird ersichtlich, dass bei der Richdem-Berechnung zwar ein Teil der Bilder mit dem richtigen Umrechnungsfaktor korrekt berechnet werden, sobald sich die Ausschnitte allerdings näher an den Polen befinden werden die Ergebnisse unbrauchbar, da hier der Umrechnungsfaktor von Me-

tern zu Längengraden und zu Breitengraden unterschiedlich ist. Die Folge ist, dass die Ergebnisse nicht stimmen und überall starke Steigerungswerte entstehen.

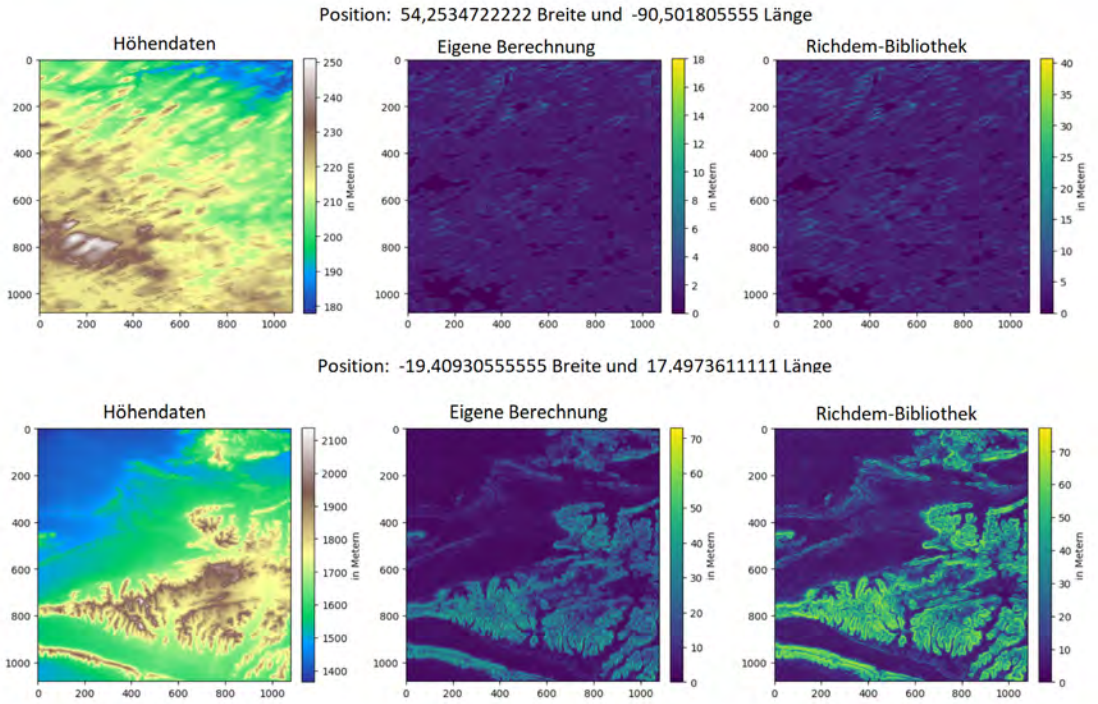

Abbildung 3.5: Vergleich der eigenen Slope Berechnung und der Richdem Methode für verschiedene Kartenausschnitte

Da ähnliche Probleme auch bei anderen Bibliotheken auftreten, wird die Berechnung des Slopes schließlich selbst implementiert. Hierfür wird ein Sliding Window genutzt, um über die gesamte Höhenkarte zu iterieren. Zunächst wird die Steigungsrichtung von der zentralen Zelle des jeweiligen Fensters bestimmt, woraufhin die in 3.1 abgebildete Formeln für die Berechnung der Steigung in Grad benötigt wird.

$$S_{inGrad} = tan^{-}1(\frac{\Delta h}{\Delta dist}) \tag{3.1}$$

$$\Delta dist = \sqrt{(F_{long} * \Delta long)^2 + (F_{lat} * \Delta lat)^2} \tag{3.2}$$

$$F_{long}(lat) = 111,3 \cdot cos(lat) \tag{3.3}$$

In diesem Zusammenhang wird der Tangens des Verhältnisses der Höhendifferenz beider Punkte (Δh) zu ihrer horizontalen Entfernung ($\Delta dist$) berechnet, um die Steigung in Grad zu ermitteln. Zur Berechnung von $\Delta dist$ wird in der vorliegenden Anwendung, aufgrund der vergleichsweise geringen Distanzen zwischen den Punkten, eine vereinfachte Annahme einer ebenen Fläche zugrunde gelegt. Auf dieser Grundlage kann der Satz des Pythagoras angewandt werden, um die horizontale Entfernung gemäß Formel 3.2 zu berechnen. Hierbei ist zu beachten, dass die Unterschiede in Breiten- und Längengraden in Meter umgerechnet werden müssen. Der Abstand zwischen zwei Breitengraden kann als Quotient des Erdumfangs durch 360 Grad beschrieben werden. Aufgrund der leicht elliptischen Form der Erde ergeben sich geringfügige Unterschiede, weshalb ein Durchschnittswert von etwa 111,13 km verwendet wird. Am Äquator beläuft sich der entsprechende Abstand zwischen zwei Längengraden auf circa 111,3 km. Da dieser Abstand an den Polen nahezu null ist, erfordert die Umrechnung stets die Berücksichtigung der Breitengradposition. Hierfür kann die in 3.3 dargestellte Formel verwendet werden, wobei *lat* die entsprechende Breitengradposition zum jeweiligen Längengrad darstellt, für den der Umrechnungsfaktor bestimmt werden soll. Die auf den erläuterten Formeln basierende Implementierung der Berechnung ist im Codeblock A.1 im Anhang abgebildet. Die zu den Daten aus 3.3 und 3.4 resultierenden Ergebnisse, sind in Abbildung 3.6 zu finden.

3.1.4 Infrastruktur und Area of Interest

Neben den geografisch relevanten Daten wurden auch Infrastrukturdaten in die Anwendung einbezogen. Zum einen gibt es Bereiche, über denen ein Flugverbot gilt, und zum anderen müssen möglicherweise sehr hohe Bauwerke, wie beispielsweise Windräder oder hohe Gebäude, umflogen werden. Zudem kann es vorkommen, dass für bestimmte Einsatzfälle gewisse Bereiche ausgeschlossen oder spezifiziert werden müssen. Ein solcher

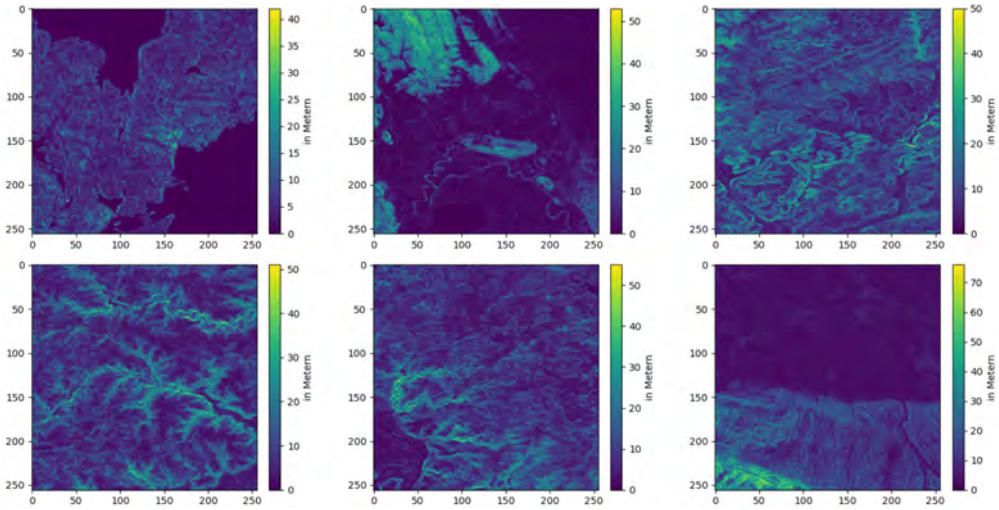

Abbildung 3.6: Verschiedene Slope Beispiele

Fall tritt beispielsweise bei der Suche nach vermissten Personen auf, bei der ein grober Aufenthaltsbereich bekannt ist.

Für diese Anwendungsfälle dient die vierte Eingangskarte, die auf Daten von OpenStreetMap (OSM) [1] basiert. OpenStreetMap ist ein Projekt, das Infrastrukturdaten öffentlich zugänglich macht. Dabei werden von der Community nicht nur Gebäude, sondern auch das öffentliche Verkehrsnetz, Straßen, Flüsse und andere Bauwerke erfasst. [57] Die OSMnx-Bibliothek für Python bietet eine Schnittstelle, um die Rohdaten direkt zu laden. Hierfür können die Daten mithilfe von verschiedenen Tags gefiltert werden. [13] Für den hier vorhandenen Anwendungsfall werden einige Tags ausgewählt, welche relevante Daten enthalten können. Zu diesen Tags werden Daten für den jeweiligen entsprechenden geografischen Bereich geladen und die erhaltenen Daten nach den folgenden Kriterien gefiltert.

- Hohe Gebäude
  - Basierend auf der angegebenen Höhe
  - Basierend auf Stockwerkanzahl
- Andere potentiell hohe Bauwerke
  - Funkmasten
  - Windräder

  - Kräne
  - Schornsteine
  - etc.
- Bereiche mit Flugverbot
  - Flughäfen
  - Atomkraftwerke
  - Militärische Einrichtungen

Die geladenen Daten sind mit geografisch referenzierten Geometrien versehen. Diese werden dann auf ein Raster übertragen und die Höheninformationen werden auf den Bereich von 0 bis 1 skaliert. Objekte, die definitiv nicht überflogen werden dürfen, erhalten den Wert 1. Die Implementierung ist im Codeblock 3.2 gezeigt. Für das Übertragen der geografisch referenzierten Geometrien auf Karten, müssen zunächst die verschiedenen möglichen Geometrietypen unterschieden werden. Anschließend werden die Koordinaten in die passenden Array Positionen umgerechnet. Hierbei werden die entsprechenden Positionen approximiert, sodass sie in die Array Rasterung passen.

Codeblock 3.2: Berechnung der Infrastrukturkarte basierend auf den georeferenzierten Geometrien

```
for index, row in osm_data.iterrows():
    #OSM uses different geometry types in its data which are structured differently so each
        must be treated separately
    if row.geometry.geom_type == 'Point':
        #Calculate the array indices based on the coordinates
        x_position = (x - xllcorner)/cellsize_median
        y_position = (y - yllcorner)/cellsize_median

        #To compensate inaccuracies caused by excessive rasterization, aproximate position
            by using a 2x2 Square instead of one point

        #Write normalized values in array

    elif row.geometry.geom_type == 'Polygon':
        #The polygon consists of multiple X and Y Coordinates which describe the boundarys
```

```
        #Aproximate the min and max array indices based on the coordinates

        if (x_max-x_min)<=1:
            if (y_max-y_min)<=1:
                #The Polygon is not larger den a 2x2 Square so write values to Array
            else:
                #The y-dimension is greater than two fields, so that all fields that lie
                    within the limits are iterated over

        elif ((y_max-y_min)<=1):
            #The x-dimension is greater than two fields, so that all fields that lie within
                the limits are iterated over

        else:
            #Both dimensions are greater than two fields
            #Check for all fileds within the min and max if they are part of the polygon

    elif row.geometry.geom_type =='LineString':
        #A LineString consists of several X and Y coordinates that form a line
        #Iterate over all coordinates and calculate Grid positions

    elif row.geometry.geom_type =='MultiPolygon':
        #A MultiPolygon consists of several polygons that are defined with X and Y
            boundaries
        foreach Polygon in MultiPolygon:
            #Do the same as in the case of a polygon
```

Beispiele für die so entstandenen Karten sind in Abbildung 3.7 dargestellt. Einige Karten sind aufgrund ihrer Lage in weniger besiedelten Gebieten oder in Gebirgsregionen vergleichsweise leer.

Um sicherzustellen, dass auf jeder Karte Hindernisse vorhanden sind und, um den zuvor beschriebenen Anwendungsfall von manuellen Einschränkungen des zu erkundenden Bereichs abzudecken, wird eine Alternative zu den reinen Infrastrukturkarten erstellt. Dabei werden zusätzliche zufällige Polygone auf jeder Karte hinzugefügt, um den zu erkundenden Bereich darzustellen. Dies ermöglicht auch die Erstellung mehrerer Infrastrukturkarten für dieselben geografischen Gebiete, um eine höhere Datenvarianz zu erzielen. Bei diesen Karten wurde, wie in Abbildung 3.8 zu sehen ist, eine Maske über die Infrastruk-

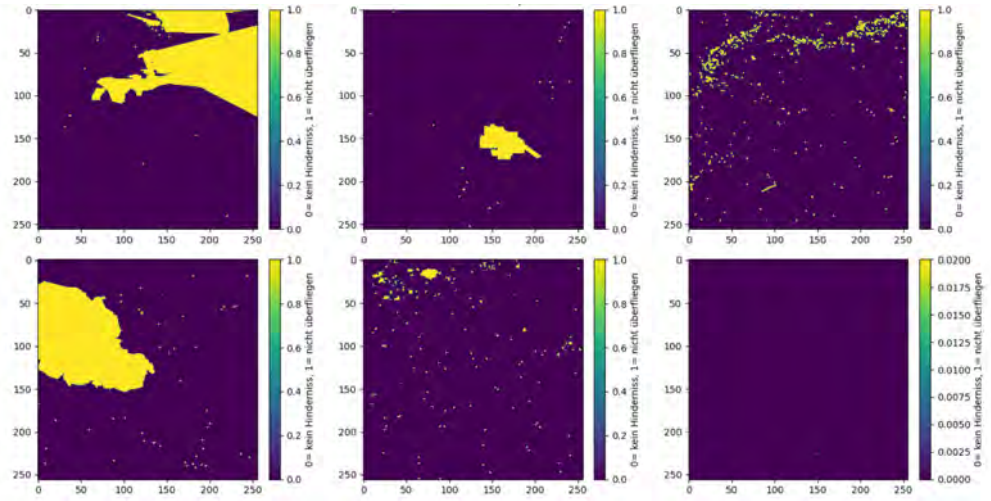

Abbildung 3.7: Verschiedene Infrastruktur Beispiele

turdaten gelegt, um im Bereich der zu erkundenden Region trotzdem die Infrastruktur sichtbar zu lassen.

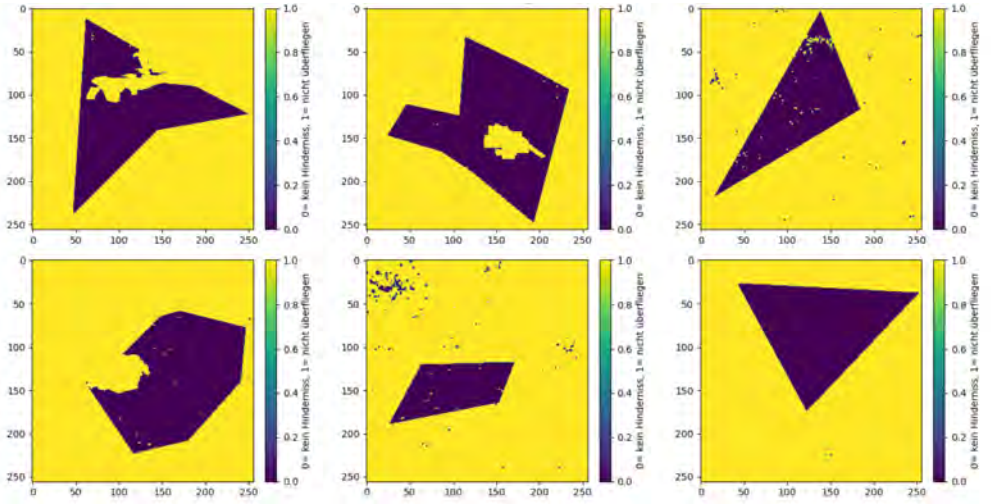

Abbildung 3.8: Verschiedene Infrastruktur Beispiele mit zusätzlichen definierten Interessensbereichen

3.1.5 Verarbeitung und Aufbereitung

Basierend auf den beschriebenen Daten wurde ein Datensatz erstellt. Zu diesem Zweck wurden zufällige rechteckige Ausschnitte mit geografischen Koordinaten generiert und

die entsprechenden Daten wurden geladen. Die hochauflösenden Informationen aus dem SRTM-Datensatz führten zu einer Generierung von Bildern beachtlicher Größe. Jedoch kann die Verwendung derart hochauflösender Daten, insbesondere in Kombination mit mehreren Bildern als Eingabe, selbst auf leistungsstarker Hardware zu signifikanten Leistungsengpässen führen. Dies ergibt sich daraus, dass die Anzahl der Modelleingabe-Parameter mit der Bildauflösung exponentiell anwächst. Um diesem Problem zu begegnen, wurden die Karten auf eine Auflösung von 256x256 Pixel reduziert. Für das Downsampling wird für das DEM und den Slope immer das Maximum genommen, um einzelne Peaks zu erhalten. Um die Steigungsausrichtung beim Downsampling zu approximieren, wird hier immer der Mittelwert genutzt. Die beiden genutzten Methoden sind im Codeblock 3.3 abgebildet. Zusätzlich werden die DEMs, Slopes und Aspects nachträglich normalisiert, um ein effizientes Modelltraining zu ermöglichen.

Codeblock 3.3: Genutzte Methoden zur Reduzierung der Auflösung bei den Input Daten

```
def reduce_resolution_max(grid, target_grid_size, y_resolution, x_resolution):
    resized_grid = np.zeros(target_grid_size)
    for i in range(0, target_grid_size[0]):
        for j in range(0,target_grid_size[1]):
            resized_grid[i,j] = max(grid[(i*y_resolution): ((i*y_resolution)+y_resolution),
                (j*x_resolution): ((j*x_resolution)+x_resolution)].flatten())
    return resized_grid

def reduce_resolution_mean(grid, target_grid_size, y_resolution, x_resolution):
    resized_grid = np.zeros(target_grid_size)
    for i in range(0, target_grid_size[0]):
        for j in range(0,target_grid_size[1]):
            group = grid[(i*y_resolution): ((i*y_resolution)+y_resolution),
                (j*x_resolution): ((j*x_resolution)+x_resolution)].flatten()
            resized_grid[i,j] = sum(group)/len(group)
    return resized_grid
```

3.2 Training

Der gesamte Trainingsprozess des Reinforcement Learnings besteht im Wesentlichen aus zwei Hauptphasen - dem Modelltraining und den Simulationen. Eine detaillierte Darstellung dieses Prozesses ist in Abbildung 3.9 zu erkennen. Zu Beginn wird der Prozess mit der Generierung neuer Potentialfelder initialisiert. Diese können entweder durch das

Modell, wie in der Abbildung dargestellt, oder gemäß der Erläuterungen in Kapitel 4 als leere oder zufällige Karten erstellt werden. Anschließend werden Simulationen auf Basis dieser Karten durchgeführt, wobei die genaue Vorgehensweise in Abschnitt 3.3 ausführlich erläutert wird. Dort wird auch in Abschnitt 3.3.5 die Generierung der Belohnungen erläutert.

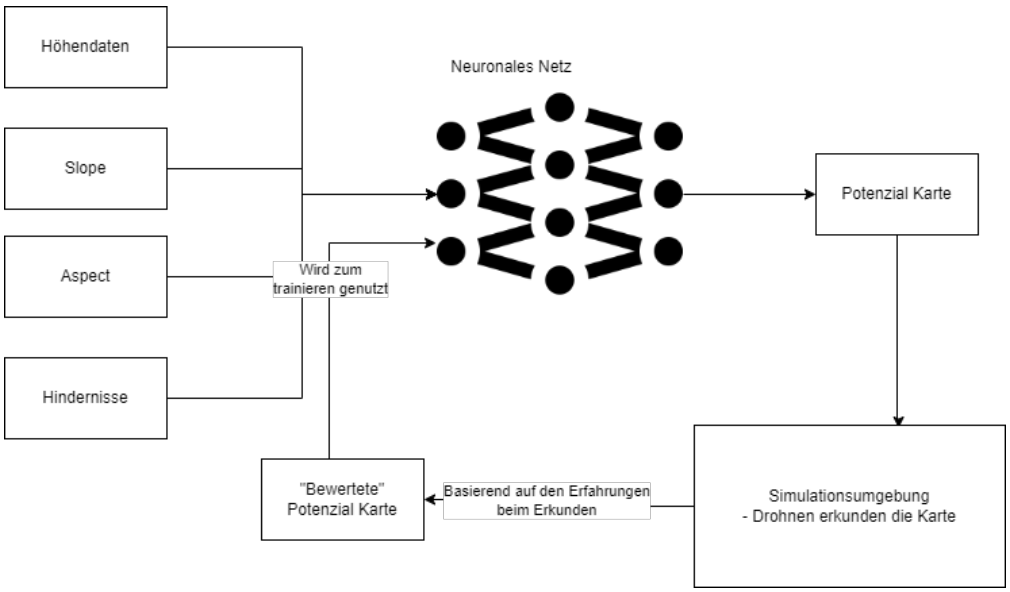

Abbildung 3.9: Darstellung des Reinforcement Learning Prozesses

Basierend auf den Belohnungen, die in den Simulationen erzielt wurden, wird das Modell trainiert, um eine neue Generation zu entwickeln, die bessere Ergebnisse erzielt. Das Modelltraining erfolgt mithilfe von Python, während die Simulationen unter Verwendung von C# durchgeführt werden. Da mit einem festen Satz von Eingabedaten gearbeitet wird und, um ein möglichst effizientes Training zu gewährleisten, wird das sogenannte Batch Reinforcement Learning eingesetzt. Dies ist eine Alternative zum kontinuierlichen Ansatz. Hierbei wird eine größere Menge an Daten auf einmal verarbeitet und nicht nach jedem Schritt eine Belohnung erstellt und verarbeitet. Dies führt zu einem stabileren Training, da das Modell nicht nach jeder Datensatzverarbeitung angepasst werden muss. Der gesamte Prozess wird effizienter, da die Simulationen parallel und das Modelltraining mittels mehrerer Worker durchgeführt werden können. [38] Durch die Verwendung des Batch-Ansatzes entfällt die Notwendigkeit der Kommunikation zwischen den beiden Komponenten und lediglich ein zentraler Ordner wird benötigt, über den die Dateien ausgetauscht werden können, sowie ein Batch-Skript, das beide Prozesse mehrfach hintereinander ausführt.

3.2.1 Ablauf

Dieser Trainingsprozess wird im Folgenden noch einmal zusammengefasst. Hierbei kann der Prozess beliebig häufig durchlaufen werden, bis das gewünschte Ergebnis an Karten entsteht oder keine Verbesserung mehr ersichtlich ist.

1. Initiale Potentialfelder (P) der Größe $m \times n$ mit Qualität $0 \Rightarrow P_0 = \{p_{0,1}, p_{0,2}, \cdots, p_{m,n}\}$ & $p_{m,n} \in [0, 1]$
2. Simulation mit Drohnen auf Potentialfeldern P_0
   - die generierten Bewertungskarten ($B_0 = \{b_{0,1}, b_{0,2}, \cdots, b_{m,n}\}$) werden auf die Potentialfelder (P_0) angewendet $\Rightarrow P_0 + B_0 = L_{PB}$
3. Training des Netzes (N_0) mit bewerteten Potentialfeldern $L_{PB} \Rightarrow$ nächste Generation vom Netz N_1
4. Generieren neuer Potentialfelder (P) der Größe $m \times n$ mit Netz $N_1 \Rightarrow P_1$
5. Simulation mit Drohnen auf Potentialfeldern P_1
   - die generierten Bewertungskarten (B_1) werden auf die Potentialfelder (P_1) angewendet $\Rightarrow P_1 + B_1 = L_{PB}$
6. Training des Netzes (N_1) mit bewerteten Potentialfeldern $L_{PB} \Rightarrow$ nächste Generation vom Netz N_2
7. Generieren der Potentialfelder mit Netz $N_2 \Rightarrow P_2$
8. Simulation mit Drohnen auf Potentialfeldern P_2
   - die generierten Bewertungskarten (B_2) werden auf die Potentialfelder (P_2) angewendet $\Rightarrow P_2 + B_2 = L_{PB}$
9. Training des Netzes (N_2) mit bewerteten Potentialfeldern $L_{PB} \Rightarrow$ nächste Generation vom Netz N_3
10. usw.

3.2.2 Training

Für das Training des Neuronalen Netzes wird Keras von Tensorflow verwendet. Die zu trainierenden Modelle werden in entsprechenden Ordnern abgelegt und von dort geladen. Um die Daten während des Trainings zu laden, wurde eine Sequenz implementiert, die zufällige Batches aus den Dateien lädt. Auf die konkreten Parameter wie Batch Größe, Anzahl an Epochen, etc. wird in Kapitel 4 genauer eingegangen, da hierfür mit verschiedenen Konfigurationen gearbeitet wurde. Zusätzlich werden die Eingabedateien zunächst als NumPy-Binärdateien gespeichert, da das Laden dieser während des Trainings im Vergleich zu ASCII-Dateien performanter ist.

3.3 Simulationsumgebung

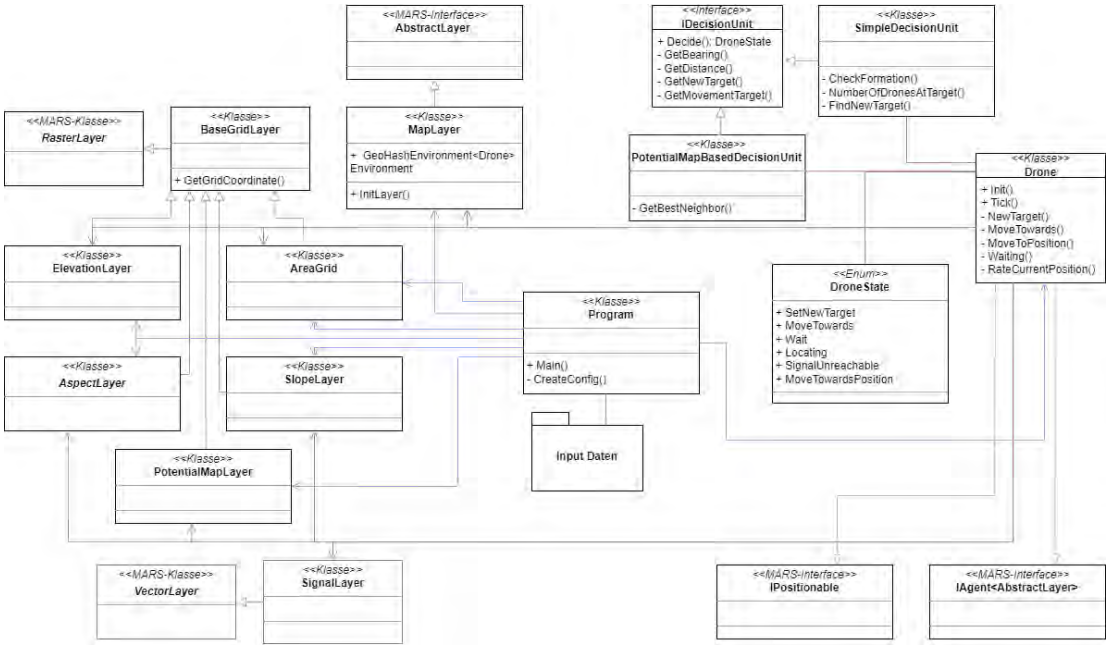

Abbildung 3.10: UML-Klassendiagramm zur Simulationsumgebung

Die Simulationsumgebung ist in C# implementiert, da das MARS Framework eingesetzt wird. Der Aufbau der Simulationsumgebung wird im Folgenden näher erläutert. Einen groben Überblick darüber liefert das Klassendiagramm in Abbildung 3.10. Die Programm-Klasse bildet hierbei das Zentrum der Simulation und umfasst eine allgemeine Beschreibung der Simulation, einschließlich ihrer Dauer und der Eingabedaten. Zu der

Simulation gehören zusätzlich noch die Ebenen und Agenten. Die in dieser Anwendung genutzten Ebenen werden in Abschnitt 3.3.1 genauer erläutert. Die Agenten sind in diesem Fall die Drohnen, auf die in Abschnitt 3.3.2 genauer eingegangen wird. Des Weiteren werden die in Abbildung 3.10 dargestellten Komponenten ebenfalls im Folgenden genauer erläutert.

3.3.1 Ebenen

Innerhalb des MARS Frameworks sind verschiedene Arten von Ebenen verfügbar. Im Rahmen dieser Masterarbeit wurden primär RasterLayer verwendet, da diese sich am besten zur Darstellung der verschiedenen Karten eignen. Für eine RasterLayer wird eine Rasterdatei, wie beispielsweise eine .asc-Datei, benötigt. Daher werden die Daten, wie bereits in Abschnitt 3.1 beschrieben, in dieser Form gespeichert. [43] Die Ebenen basieren auf georeferenzierten Karten. Für die Simulation des Reinforcement Learning wurden insgesamt sechs verschiedene Ebenen genutzt. Zudem gibt es eine weitere Ebene für das Orten von Signalquellen. Dies sind die folgenden:

- ElevationLayer
- SlopeLayer
- AspectLayer
- AreaGrid
- PotentialMapLayer
- MapLayer
- SignalLayer

Einige dieser Ebenen dienten lediglich dazu die Eingabedaten zu simulieren, auf deren Grundlage die Drohnen ihr Feedback generieren. In einer realen Anwendung müssten dafür echte Sensordaten verwendet werden und nicht dieselben Eingabedaten, mit denen das Modell trainiert wird, wie es hier der Fall ist. Zu diesen Ebenen gehören die ElevationLayer, welche die Höhendaten enthält, sowie die SlopeLayer und AspectLayer, die die entsprechenden Steigungswerte und die Neigungsausrichtungen enthalten. Schließlich gibt es das AreaGrid, das relevante Infrastrukturdaten oder andere Hindernisse enthält.

Diese vier Ebenen dienen lediglich als Basis für die Bewertung. Hierfür wären echte Sensordaten, wie beispielsweise reale Flugdaten, besser geeignet.

Des Weiteren gibt es die PotentialMapLayer, diese enthält das entsprechende Potentialfeld und dient als Grundlage für die Routenplanung der Drohnen. Sie bildet das Wissen der Drohnen-Agenten über die Umwelt ab. Wie in Abbildung 3.10 zu sehen ist, erben sowohl die PotentialMapLayer als auch das AreaGrid, die Elevation-, Aspect- und Slope-Layer vom BaseGridLayer. Diese bietet eine Methode zum Umrechnen von internen Koordinaten in georeferenzierte Koordinaten, was für die Positionsbestimmung der generierten Bewertungen auf den Karten erforderlich ist. Des Weiteren verfügt die Simulationsumgebung über die MapLayer, welche lediglich den Lebensraum für die Drohnen-Agenten darstellt.

Zuletzt existiert noch die SignalLayer. Hierbei handelt es sich um eine VectorLayer. Diese Ebene wird für das Orten der Signalquellen benötigt und nicht für die Trainingssimulationen. Die Ebene enthält die ungefähren Positionen der Signalquellen, welche die Drohnen lokalisieren sollen.

3.3.2 Agenten

Für diese Masterarbeit wurde ein Agententyp implementiert, der den digitalen Zwilling einer realen Drohne darstellt. Dieser besitzt dieselben Fähigkeiten wie eine echte Drohne und trifft mithilfe einer Entscheidungseinheit Entscheidungen über seine Handlungen. Zur Wegefindung nutzt er das Potentialfeld und beobachtet dabei, durch Berücksichtigung der verschiedenen Ebenen, die Umgebung. Diese Ebenen simulieren die Erfahrungen, die die Drohnen während eines realen Fluges machen würden, und helfen bei der Bewertung der gewählten Schritte. Ein Agent kann sich mit unterschiedlichen Geschwindigkeiten in verschiedene Richtungen bewegen oder den direkten Weg zu einem vorgegebenen Ziel einschlagen. Zudem ist er neben seiner eigenen Position auch mit den Positionen der anderen Drohnen vertraut.

Im Codeblock 3.4 ist die grundlegende Struktur des Agenten vereinfacht dargestellt. Der Agent implementiert die Tick()-Methode des IAgent MARS Interfaces, die in jedem Zeitschritt aufgerufen wird. Innerhalb dieser Methode erfolgt eine Bewertung der Position auf Grundlage der Umgebungsinformationen. Gleichzeitig werden alle relevanten Informationen an die Entscheidungseinheit übergeben, damit diese den nächsten Schritt entscheiden kann. Diese Informationen umfassen beispielsweise den aktuellen Standort, das Ziel und

das Potentialfeld. Basierend auf dieser Entscheidung wird die nächste Handlung ausgeführt.

Codeblock 3.4: Allgemeine Struktur des Drohnen Zwillings

```
public class Drone : IAgent<MapLayer>, IPositionable{
    [...]  //Declaration of variables
    public void Init(MapLayer layer) [...] // Initialization of variables

    public void Tick(){
        RateCurrentPosition(different_Layers)
        switch (decisionUnit.Decide(different_Layers,Position, [...]) ) {
            case DroneStates.SetNewTarget:
                NewTarget();
            case DroneStates.MoveTowardsPosition:
                MoveToPosition(decisionUnit.GetMovementTarget(),
                    decisionUnit.GetDistance());
            case DroneStates.Wait:
                Waiting();
            case DroneStates.Locating:
                Locating()
            case DroneStates.MoveTowards:
                MoveTowards((decisionUnit.GetBearing()), decisionUnit.GetDistance());
            default:
                Waiting();
        }
    } [...]
```

3.3.3 Steuerung

Die Drohnen werden über eine separate Einheit gesteuert, die das IDecisionUnit-Interface implementieren muss. Diese Einheit erhält alle relevanten Informationen der Drohnen und trifft basierend darauf die Entscheidung, welche Handlung als nächstes ausgeführt werden soll. Die Entscheidung wird mithilfe verschiedener Drohnenzustände zurückgegeben, auf die der Drohnen-Agent entsprechend reagieren kann. Diese Abstraktion der Steuerung der Drohnen wurde gewählt, um die Möglichkeit zu schaffen, die Steuerung der Drohnen modular anzupassen oder auszutauschen. In dieser Anwendung wurde eine Steuerungseinheit entwickelt, die zur Bestimmung der Drohnenroute die Potentialfelder

nutzt. Hierfür wird, basierend auf den umliegenden Potentialfeldwerten, der Distanz zum Ziel und der Distanz zum eigenen Standort, der beste nächste Schritt ausgewählt.

3.3.4 Erkunden

Die Simulationen bestehen aus mehreren Drohnen, die einen gemeinsamen Startpunkt erhalten und ein oder mehrere Zielpunkte anfliegen. Diese werden zufällig generiert. Basierend auf den Werten des Potentialfelds suchen sie ihren Weg zum Ziel. Dazu wird der in Kapitel 2.6.1 erläuterte ε-greedy Ansatz verwendet. Der konkrete Zufallsanteil wurde für die verschiedenen Simulationen variiert. Nachdem das erste Ziel erreicht wurde, setzen die Drohnen ihre Suche zum nächsten Ziel fort. Um verschiedene Bereiche der Karte zu erforschen, wurden zudem mehrere Simulationen pro Karte durchgeführt.

3.3.5 Belohnung

Zur Generierung der Bewertungen, anhand derer das Modell trainiert wird, bewerten die Agenten jeden Standort hinsichtlich der Entscheidung, dorthin zu gehen. Dabei wird nicht nur der eigene Standort, sondern auch die unmittelbare Umgebung berücksichtigt. Wenn beispielsweise in der Nähe starke Steigungen oder infrastrukturelle Hindernisse auftreten, die nicht überflogen werden können, wird ein entsprechender Schritt negativ bewertet. Im Gegensatz dazu wird ein Schritt positiv bewertet, wenn in der Nähe keine Hindernisse, starke Steigungen bzw. Höhenunterschiede vorhanden sind. Die Bewertungen mehrerer Drohnen werden kombiniert und auf die verwendeten Potentialfelder aufaddiert, um damit die Modelle zu trainieren.

3.3.6 Visualisierung

Für die Visualisierung der durchgeführten Simulationen werden die Start- und Zielpunkte für jede Simulation gespeichert. Zudem werden die geflogenen Routen der Drohnen gespeichert, wofür das MARS Framework von Haus aus eine Option bietet. Diese Daten können genutzt werden, um beispielsweise mit Python Plots der geflogenen Routen zu erstellen. Alternativ können die Simulationen mithilfe von Kepler GL abgespielt und visualisiert werden, um sie genauer zu betrachten.

3.4 Anwendungsbeispiel Simulation der Ortung mittels TDoA

Für den einfachen Pfad, bei dem grobe Standorte bekannt sind, wie er im Grundlagenabschnitt 2.1.3 beschrieben ist, wurde eine Simulation in der Simulationsumgebung implementiert. Hierbei wird im folgenden vom Standort des Signals gesprochen, damit ist dieser vermutete Bereich in dem das Signal ist gemeint. Diese Simulation dient als ein späteres Anwendungsbeispiel für die Potentialfelder. Der allgemeine Aufbau und die genutzten Komponenten dieser Simulation entsprechen größtenteils denen der Simulation für das Reinforcement Learning (RL).

3.4.1 Konfiguration

In der Konfiguration der Drohnen ist die gewünschte Anzahl an Drohnen und ihre entsprechenden Startpunkte hinterlegt. Die Anzahl der Drohnen ist variabel, jedoch müssen mindestens vier Drohnen vorhanden sein, da dies die minimale Anzahl für eine hinreichend genaue Lokalisierung ist. Es können hier theoretisch auch mehr Drohnen verwendet werden. Jedoch werden aktuell nicht mehr als vier Drohnen in der Agentenlogik berücksichtigt. Entsprechend würden Drohnen bei der Zielwahl nicht berücksichtigen, ob sich dort bereits vier weitere Drohnen befinden und auch die Positionierung um das Ziel herum ist für einen solchen Fall nicht optimiert.

Neben der Drohnenkonfiguration wird eine GeoJSON-Datei mit den zu lokalisierenden Signalquellen benötigt. Derzeit ist nur der einfache Fall in der Simulation enthalten, bei dem bereits grobe Standorte bekannt sind. Zusätzlich muss eine Grid-Datei hinterlegt sein, die den Bereich enthält, in dem sich die Drohnen aufhalten dürfen. Diese Karte kann auch Hindernisse enthalten, die umflogen werden müssen. Optional kann auch eine Höhenkarte hinterlegt werden. In diesem Fall meiden die Drohnen Bereiche, die über einer konfigurierten Höhe liegen. Das Ergebnis einer solchen Konfiguration mit Hindernissen und Höhenkarte ist in Abbildung 3.11 zu sehen. Dort wurden zehn Signalquellen (in rot) und vier Drohnen (in blau) genutzt.

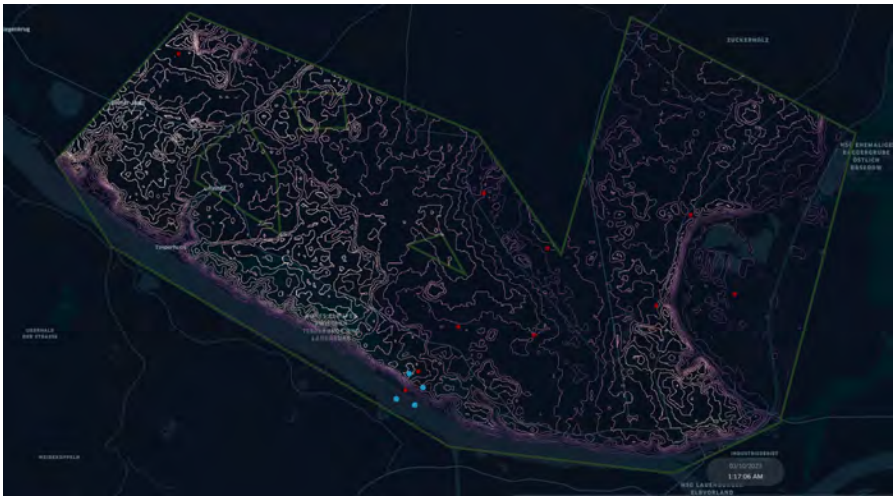

Abbildung 3.11: Visualisierung einer Simulation mit Kepler GL mit hinterlegter Höhenkarte und Hindernissen. In Rot sind die Signalquellen und in Blau die Drohnen eingezeichnet.

3.4.2 Allgemeiner Ablauf

Wird die Simulation gestartet, sind den Drohnen die Standorte der Signalquellen, ihr eigener Standort sowie der Standort der anderen Drohnen bekannt. Dabei haben sie jedoch keine Kenntnis über die Pläne der anderen Drohnen. Über die gesamte Simulation hinweg treffen sie ihre Entscheidungen ausschließlich auf Basis dieser Standortinformationen und Informationen aus den Ebenen. Sie nutzen die SignalLayer, ElevationLayer und das AreaGrid. Auf Grundlage dieser Informationen wählen die Drohnen ein Ziel aus, das sie anfliegen. Nachdem sie angekommen sind, warten sie bis mindestens drei weitere Drohnen eingetroffen sind, um eine optimale Lokalisierung durchführen zu können.

Sobald weitere Drohnen eingetroffen sind, beginnen sie sich möglichst gleichmäßig um die Signalquelle zu verteilen. Nachdem sich vier Drohnen erfolgreich formiert haben, gilt die Signalquelle als lokalisiert und sie fliegen weiter zum nächsten Ziel. Falls keine weiteren Drohnen hinzukommen, fliegen sie nach einer gewissen Zeit zum nächsten Ziel. Die Abbildung 3.12 zeigt die zurückgelegten Routen aller Drohnen am Ende der Simulation.

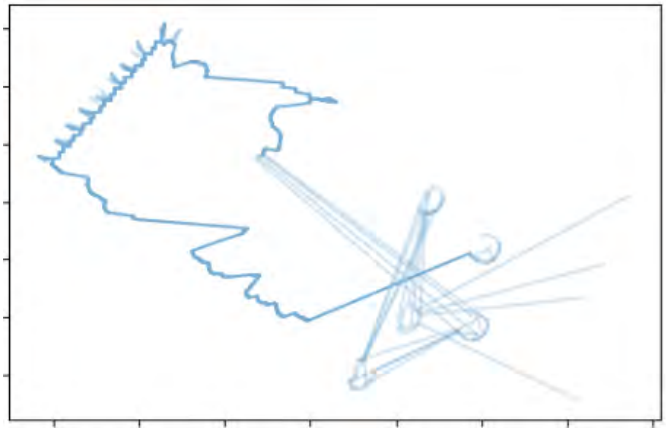

Abbildung 3.12: Geflogene Wege aller Drohnen in einer Simulation

3.4.3 Agentenlogik

Für diese Simulation wird die SimpleDecisionUnit als Entscheidungseinheit verwendet. Generell lassen sich zwei Phasen definieren, in denen sich der Agent befinden kann. In der ersten Phase zeigt der Agent ein nicht-kooperatives Verhalten und wechselt dann in die zweite Phase, in der ein kooperatives Verhalten auftritt. Zwischen diesen beiden Phasen wechselt die Drohne je nach ihrem aktuellen Zustand. Die allgemeine Umsetzung dieser Entscheidungsfindung ist im Codeblock A.2 im Anhang dargestellt.

In der ersten Phase besteht die Aufgabe darin ein Ziel zu suchen und dieses anzusteuern. Der Agent nutzt in diesem Stadium noch keine intelligente Wegplanung, sondern wählt den direkten Weg zum Ziel. Wenn er auf ein Hindernis stößt, beginnt er es zu umfliegen, indem er sich vorbei tastet.

Für die Entscheidung, welches Ziel angeflogen wird, nutzt die Drohne die Standorte aller anderen Drohnen, um sicherzustellen, dass möglichst alle dasselbe Ziel ansteuern. In diesem Zusammenhang wird ein nicht-kooperatives Verhalten angewendet, um die Kommunikation so gering wie möglich zu halten. Zur Bestimmung des Ziels berechnet die Drohne den Mittelwert ($\overline{p}$) aller aktuellen Drohnenpositionen (p_n) der n Drohnen.

$$\overline{p} = \frac{1}{n} \sum_{i=1}^{n} p_i$$

Anschließend wird die Signalquelle basierend auf $\overline{p}$ gewählt. Durch dieses Vorgehen wird sichergestellt, dass die Wahrscheinlichkeit, dass alle Drohnen dasselbe Ziel ansteuern,

sehr hoch ist, ohne dass eine direkte Absprache erforderlich ist. Darüber hinaus wird ein Ziel gewählt, das für alle Drohnen möglichst nah liegt.

Nachdem die Drohne das Ziel erreicht hat, tritt die zweite, die kooperative Phase ein. In dieser Phase geht es darum sich mit anderen Drohnen zu koordinieren und um das Ziel optimal zu positionieren. Damit verhindert wird, dass eine Drohne zu lange wartet, wird ein Timer gestartet. Erreicht der Timer den Wert Null, wird angenommen, dass keine weitere Drohne das gleiche Ziel anfliegt. Die Drohne wechselt darauf wieder in die erste Phase und sucht nach einem anderen Ziel. Die Wartezeit an einem Ziel ist abhängig von der Anzahl der Drohnen, die bereits an diesem Ziel warten. Je mehr Drohnen dort warten, desto länger wird gewartet. Dies soll sicherstellen, dass, wenn sich die Drohnen in zwei Gruppen an unterschiedlichen Zielen befinden, die kleinere Gruppe zur größeren Gruppe fliegt. Wurde ein Signalquelle erfolgreich lokalisiert, wechseln die Drohnen wieder in die erste Phase und beginnen mit der Suche nach einem neuen Ziel.

Sind mehrere Drohnen bei der gleichen Signalquelle, versuchen sie sich um das Ziel zu verteilen, um eine effektive Lokalisierung zu ermöglichen. Auch bei der Positionierung um das Ziel herum wird keine direkte Absprache genutzt. Das Ziel besteht darin, dass sich die Drohnen möglichst gleichmäßig um die Signalquelle anordnen. Der für dieses Verhalten genutzte Algorithmus wurde durch den Boids-Algorithmus von Reynolds (1987) [50] inspiriert. Der Boids-Algorithmus wird genutzt, um das Verhalten von Vogel- und Fischschwärmen nachzubilden. Hierfür setzt der Algorithmus drei einfache Grundprinzipien ein, nach denen die Individuen ihre Bewegungsrichtung koordinieren. Dazu gehört die Separation von anderen Individuen, das Ausrichten basierend auf der Ausrichtung der anderen Individuen und die Anziehung zum Mittelpunkt der Gruppe. [50] Der für die Formierung genutzte Algorithmus kombiniert anziehende und abstoßende Verhaltensweisen, um die Drohnen auf Basis einer Kreisbahn gleichmäßig um das Ziel zu verteilen. Hierbei ist keine aufwendige Kommunikation zwischen den Drohnen erforderlich. Das Ziel besteht darin, dass sich vier Drohnen annähernd in einem Quadrat um die Signalquelle anordnen. Um dies zu erreichen, wirken abstoßende Kräfte von anderen Drohnen auf die betrachtete Drohne, während die Signalquelle anziehend wirkt.

Der allgemeine Prozess ist in Abbildung 3.13 dargestellt. Im ersten Schritt separieren sich die Drohnen von anderen Drohnen. Dabei werden nur die anderen Drohnen betrachtet, die sich in der nähe befinden. In Abbildung 3.13 sind dies d_2 und d_3. Basierend auf deren Positionen wird die Richtung bestimmt, in die sich die Drohne bewegen soll. Bevor diese Richtung als finale Bewegungsrichtung verwendet wird, wird jedoch noch der

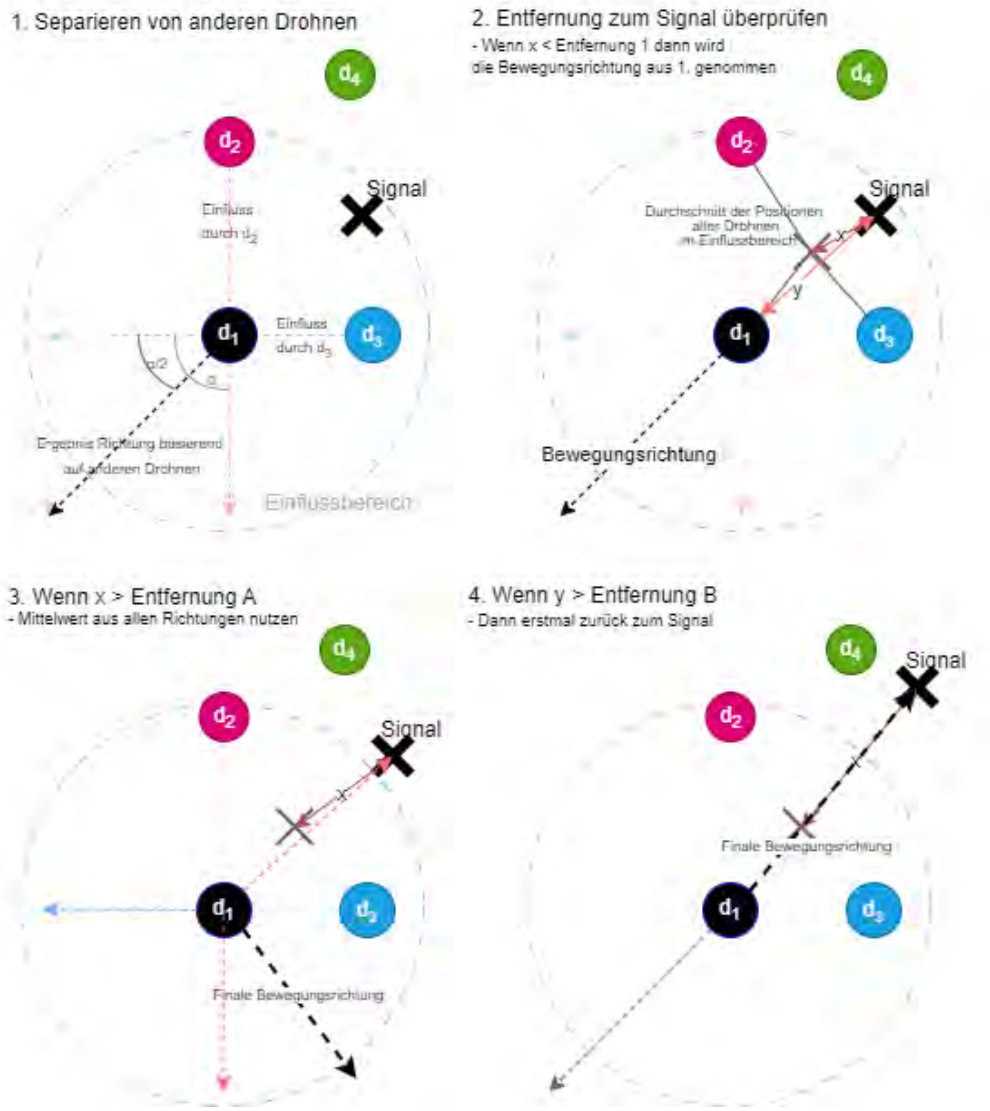

Abbildung 3.13: Ablauf, der zum Formieren um eine Signalquelle genutzt wird

Mittelwert aus den Positionen von $d_{1,2,3}$ gebildet. Falls diese Durchschnittsposition noch in der Nähe der Signalquelle liegt, wird die berechnete Bewegungsrichtung übernommen. Falls der Abstand zwischen dem Mittelwert und der Signalquelle jedoch zu groß ist, wird die Richtung, in die sich d_1 bewegen müsste, um zur Signalquelle zu gelangen, ebenfalls berücksichtigt. In diesem Fall wird der Mittelwert aller Richtungen genommen. In Abbildung 3.13 wären dies die entgegengesetzten Richtungen von d_2, von d_3 und die Richtung der Signalquelle. Ist jedoch die Entfernung zwischen der Drohne und der Signalquelle besonders groß geworden, fliegt die Drohne zunächst zurück zur Signalquelle und versucht sich erneut zu positionieren. In Abbildung 3.13 zeigt dies die vierte Abbildung. Die entsprechende Implementierung ist im Codeblock 3.5 dargestellt.

Da in der Regel vier Drohnen pro Signalquelle vorhanden sein sollen, wird ein Viertel des Kreisumfangs als optimale Distanz festgelegt. Um sicherzustellen, dass die Positionierung auch mit mehr als vier Drohnen funktioniert, wird der Radius bei mehr Drohnen erhöht. Um die Möglichkeit eines stabilen Punktes bei der Positionierung zu erhöhen und gleichzeitig sicherzustellen, dass die Positionierung auch mit mehreren Drohnen funktioniert, wird für die weitere Berechnung 70 % der optimalen Distanz als Schwellenwert genutzt.

Anschließend wird der Abstand zu den anderen Drohnen überprüft und die Winkel werden aufaddiert. Falls keine Drohnen zu nah sind, wird die Position beibehalten. Falls doch, wird der Abstand zum Ziel überprüft und gegebenenfalls wird der Bewegungswinkel angepasst. Um zu verhindern, dass sich die Drohnen gegenseitig so beeinflussen, dass sie alle vom Ziel wegfliegen, wird schließlich noch der Abstand der Drohne zum Ziel überprüft. Wenn die Drohne sich bereits zu weit vom Ziel entfernt hat, fliegt sie zunächst in einem kleinen Bogen zurück zum Ziel. Dies soll sicherzustellen, dass die Drohnen nicht wieder in genau der gleichen Konstellation am Ziel ankommen und idealerweise verteilter sind. Auf diese Weise soll die Chance erhöht werden, dass ein erneuter Formierungsversuch gelingt.

Codeblock 3.5: Berechnung der Bewegungsrichtung

```
double circumference = ((DistanceToTarget +(10*(droneAtSameTarget.Count-4)))* 2.0) *
    Math.PI;
double distanceBetweenDrones = circumference/ 4.0;
foreach (KeyValuePair<string, SimpleDrone> droneValuePair in droneAtSameTarget) {
    SimpleDrone simpleDrone = droneValuePair.Value;
    // Only take the other drones into account.
    if (!droneValuePair.Key.Equals(_droneName)) {
```

```
            // If the distance to the other drone is too small add the opposite bearing, so we
                move away from this drone
            if (simpleDrone.Position.DistanceInMTo(dronePosition) <
                (distanceBetweenDrones*0.7)){
                averageLat += simpleDrone.Position.Latitude;
                averageLong += simpleDrone.Position.Longitude;
                averageBearingCount++;
                bearing += simpleDrone.Position.GetBearing(dronePosition);
            }
        }
    }
    // If there where no bearings added to the average, this instance has enough distance to
        everyone and doesn't need to move.
    if (averageBearingCount >0) {
        // The own position is added and the average values for the positions and bearings are
            calculated.
        averageLat += dronePosition.Latitude;
        averageLong += dronePosition.Longitude;
        averageLat /= (averageBearingCount+1);
        averageLong /= (averageBearingCount+1);

        // The average position indicates if all drones are positioned around the target.
        Position centerPosition = new Position(averageLong, averageLat);
        if (centerPosition.DistanceInMTo(_targetPosition) > 50) {
            bearing += centerPosition.GetBearing(_targetPosition);
            bearing /= (averageBearingCount+1);
        } else {
            bearing /= averageBearingCount;
        }
        // If we move to far away from the target we will first fly back to the target in a
            small circle
        if(dronePosition.DistanceInMTo(_targetPosition) > DistanceToTarget +
            (5*(droneAtSameTarget.Count-4))) {
            bearing = (dronePosition.GetBearing(_targetPosition)+ 80) %360;
        }
            bearing = bearing;
            return DroneStates.MoveTowards;
    } else {
        return DroneStates.Wait;
    }
```

4 Experimente

Unter Verwendung des zuvor beschriebenen Aufbaus wurden verschiedene Konfigurationen getestet, um eine umfassende Evaluation der Leistungsfähigkeit vorzunehmen. Um die Komplexität zu begrenzen und eine solide Grundlage zu schaffen, wurde eine grundlegende Netzkonfiguration gewählt, die in Abschnitt 4.1 ausführlicher erläutert wird. Anschließend werden die relevantesten Experimente und Ergebnisse präsentiert und erläutert. Einige Abbildungen von Ergebnissen, die im Text nur klein dargestellt sind, befinden sich zusätzlich im Anhang, um eine detailliertere Betrachtung zu ermöglichen.

4.1 Grundlegende Netzkonfiguration

Abbildung 4.1: Visualisierung des grundlegenden Netzaufbaus, der in den Experimenten verwendet wird

Die grundlegende Netzkonfiguration nutzt einen Convolutional Autoencoder. Die Entscheidung ein Convolutional Neural Network zu verwenden, wurde getroffen, da die Bildgröße von 256x256 Pixeln und die vier Input Ebenen bereits zu einer Input Dimension von 262.144 führt. Bei der Verwendung mehrerer vollständig verbundener Schichten würde die Anzahl der Parameter stark ansteigen, was schnell zu Ressourcenproblemen führen kann. [52] Darüber hinaus lässt sich dieser Anwendungsfall gut mit der klassischen Bildverarbeitung vergleichen, da es abstrakt betrachtet nur um die Verarbeitung mehrerer Bildkanäle zu einem Bild mit derselben Größe und nur einem Kanal geht. Daher eignet

sich besonders eine Autoencoder Struktur, bei der der Output genauso groß ist wie der Input. Außerdem eignet sich eine solche Architektur gut für die Bildverarbeitung und die Reduktion der Dimensionalität der Input Daten. [4, 9] Als eine solche Kombination lässt sich der hier vorliegende Fall beschreiben. In Abbildung 4.1 ist die allgemeine Struktur noch einmal dargestellt. Es werden sieben Hidden-Layer verwendet. Wie zu erkennen ist, wird das Bild mithilfe von drei Blöcken aus Convolutional Layern und MaxPooling zunächst auf eine Größe von 32x32 Pixeln reduziert und anschließend mithilfe von drei Convolutional Layern und Up-Sampling wieder auf 256x256 Pixel vergrößert. Die Reduktion auf 32x32 Pixel hat den Vorteil, dass dadurch ein größeres Receptive Field entsteht und auch globalere Informationen erfasst werden können. [52]

Für die Hidden-Layer wird die Rectified Linear Unit (ReLU) als Aktivierungsfunktion verwendet. Diese Wahl wurde getroffen, da die Berechnung mit ReLU sehr schnell ist und es seit 2010 der De-facto-Standard ist. [28] Für die Ausgabeschicht wird hingegen die Logistische Aktivierung (Sigmoid) verwendet, um einen kontinuierlichen Wertebereich zwischen 0 und 1 als Ausgabe zu erhalten. Des Weiteren wird in der grundlegenden Konfiguration Adam als Optimierer verwendet, da es sich hierbei um einen weit verbreiten Optimierer handelt, der sehr robust bei verrauschten Daten und recheneffizient sein soll. [54]

4.2 Training basierend auf initialen Netzvorhersagen

Zu Beginn werden initiale Vorhersagen mithilfe des untrainierten Netzwerks erstellt, um anschließend mit den Simulationen zu beginnen. Im Folgenden werden die ersten durchgeführten Experimente präsentiert, die auf diesem Vorgehen basieren. Es werden die angewendeten Konfigurationen sowie die Trainingsergebnisse vorgestellt und evaluiert. Basierend auf diesen Ergebnissen wird erläutert, welche Anpassungen für die darauf folgenden Durchläufe vorgenommen werden.

Für die nachfolgend beschriebenen Experimente wird in Codeblock 4.1 die grundlegende Struktur des verwendeten Python-Skripts dargestellt, das für das Training des neuronalen Netzes genutzt wird.

Codeblock 4.1: Erste Implementierung des Trainings des Neuronalen Netzes

```
model_version = int(np.loadtxt(basepath+ 'iteration.asc'). tolist ())

#...Load data from csv file ...
```

```
#...Reformatting the data into a multidimensional array...

model = tf.keras.models.load_model(model_path)
history = model.fit(train_data_array, train_labels, batch_size, epochs, validation_split)
#...save model and history

for file in input files:
    #...Load input files...
    #... Make prediction and save result to new files...

#Update Iteration number
iterationfile .write(model_version+1)
```

4.2.1 Experiment 1

In diesem Experiment werden anfänglich grundlegende Tests durchgeführt, um einen umfassenden Einblick in die Ergebnisse zu gewinnen.

Konfiguration

Es wird die in Abbildung 4.2 gezeigte Konfiguration verwendet, welche die grundlegende Struktur darstellt, wie sie im vorherigen Abschnitt beschrieben wurde. In der ersten Ebene werden 32 Filter verwendet. Die Anzahl der Filter nimmt mit abnehmender Layer-Größe zu und nimmt anschließend mit zunehmender Layer-Größe wieder ab. Darüber hinaus wird mit einer Minibatch-Größe von 32 und über 100 Epochen trainiert.

Ergebnisse

Zu Beginn wird das Netz zunächst über eine Iteration trainiert. Zur Veranschaulichung des Trainingsergebnisses zeigt Abbildung 4.3 zwei beispielhafte Datensätze und in Abbildung 4.4 sind die zugehörigen Vorhersagen aus der ersten Trainingsiteration. Dabei ist zu beachten, dass das obere Bild einen Testdatensatz darstellt und das untere ein Trainingsdatensatz.

Hierbei ist einmal ein Zwischenstand nach 50 Epochen und dann nochmal am Ende des Trainings nach 100 Epochen zu sehen. Bei genauerer Betrachtung der Eingangsdaten

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      1184

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 128)       147584

 conv2d_4 (Conv2D)           (None, 32, 32, 128)       147584

 up_sampling2d (UpSampling2D  (None, 64, 64, 128)      0
 )

 conv2d_5 (Conv2D)           (None, 64, 64, 128)       147584

 up_sampling2d_1 (UpSampling  (None, 128, 128, 128)    0
 2D)

 conv2d_6 (Conv2D)           (None, 128, 128, 64)      73792

 up_sampling2d_2 (UpSampling  (None, 256, 256, 64)     0
 2D)

 conv2d_7 (Conv2D)           (None, 256, 256, 32)      18464

 conv2d_8 (Conv2D)           (None, 256, 256, 1)       33

=================================================================
Total params: 628,577
Trainable params: 628,577
Non-trainable params: 0
_________________________________________________________________
Optimizer:  <keras.optimizers.optimizer_v2.adam.Adam object at 0x0000024E76045AB0>
Loss:  <function mean_squared_error at 0x0000024C9DF8A7A0>
```

Abbildung 4.2: Die Netzkonfiguration, die im ersten und zweiten Experiment verwendet wird

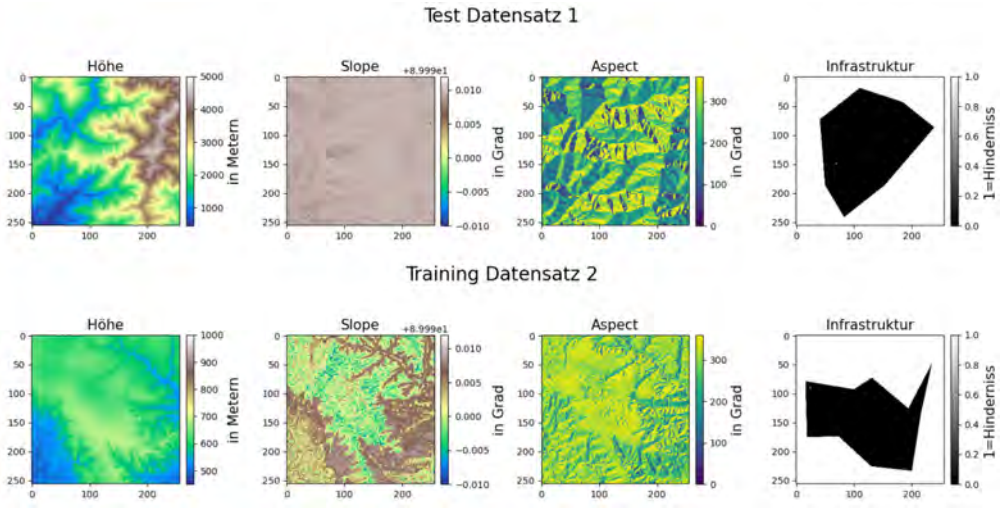

Abbildung 4.3: Beispieldaten für das ersten Experiment

wird deutlich, dass noch mit den fehlerhaft berechneten Slope-Werten von Richdem gearbeitet wird, wie in Abschnitt 3.1.3 ausgeführt ist. Dieser Fehler wurde nach diesem Experiment behoben, und es wurde ein neuer Datensatz erstellt, bei dem die Berechnung eigenständig implementiert wurde. Das Ziel bei den Potentialfeldern ist, dass die positiven Bereiche, deren Überfliegen sicher oder vorteilhaft wären, mit einer höheren Zahl (Potential) bewertet werden als die Bereiche, die gemieden oder gar nicht überflogen werden sollen. Der hierbei verwendete Wertebereich ist null bis eins.

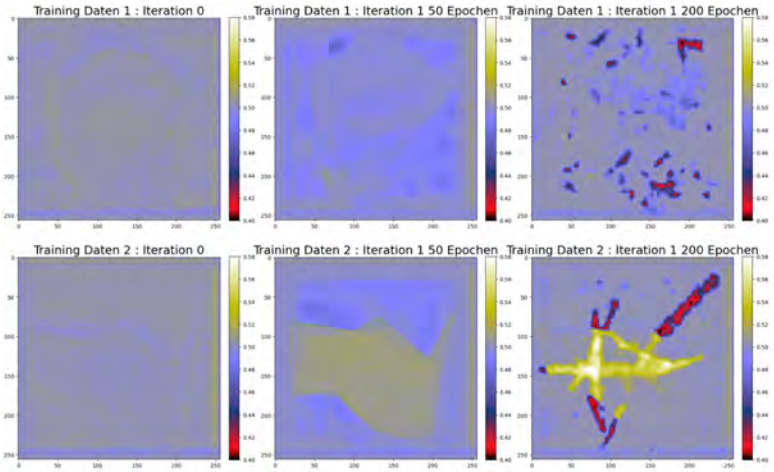

Abbildung 4.4: Die generierten Potentialkarten der initialen und der ersten Iterationen nach 50 und 200 Epochen Training [Skala: 0,4 - 0,58]

Bei Betrachtung der Vorhersage (Abbildung 4.4) fällt auf, dass bereits die initiale Vorhersage, bei der das Modell noch nicht trainiert wurde (Iteration 0), leicht die Infrastrukturkarte abbildet. Nach 50 Trainingepochen ist diese bei dem Trainingsdatensatz sehr deutlich zu erkennen. Bereits hier wird der Unterschied zwischen den Trainings- und Testdaten ersichtlich, da diese bei der Testkarte kaum zu erkennen ist. Nach 100 Epochen ist die Karte deutlich schwächer zu erkennen und das Bild enthält allgemein kaum Details. Insbesondere bei den Testdaten wird ersichtlich, dass das Modell hier nichts gelernt hat, da die Flecke keinen erkennbaren Zusammenhang mit den Eingangsdaten aufweisen. Bei dem Trainingsdatensatz lässt sich zumindest noch der Bezug zwischen den Flecken und den generierten Belohnungen, die in Abbildung 4.5 abgebildet sind, vermuten. Diese Belohnungskarten gehören zu den Datensatz 2 der in Abbildung 4.3 dargestellt ist und werden für das Training genutzt, aus dem die in Abbildung 4.4 dargestellte Potentialkarte hervorgeht. Sie zeigen die fünf Simulationsrunden mit den verschiedenen Start- und Zielpunkten, die durchgeführt wurden. Diese Karten werden anschließend für das Training verwendet. Zudem fällt auf, dass der Wertebereich auch in der ersten Iteration noch sehr klein ist und durch die initiale untrainierte Vorhersage ein Bias entstanden ist, den das Modell beibehalten hat.

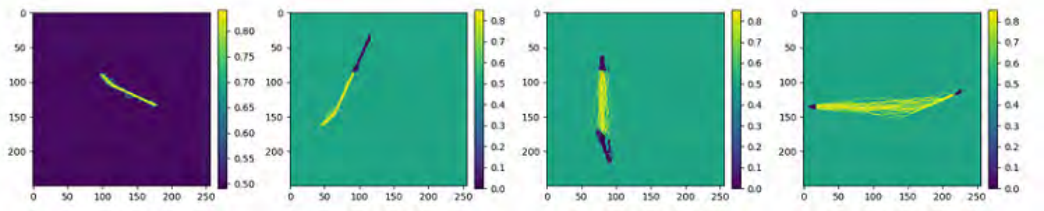

Abbildung 4.5: Beispiel-Belohnungskarten aus dem ersten Experiment

Des Weiteren verdeutlicht die Betrachtung der Accuracy und des Loss, dass es zu Overfitting kommt. Zudem ist der Datensatz deutlich zu klein, sodass allgemein nur eine kaum messbare Accuracy erreicht wird.

4.2.2 Experiment 2

Aufgrund des zu kleinen Datensatzes und der fehlerhaften Slope-Werte wird ein zweiter Durchlauf mit einem neuen, umfangreicheren Datensatz durchgeführt, bei dem die Slope-Werte eigenständig berechnet werden.

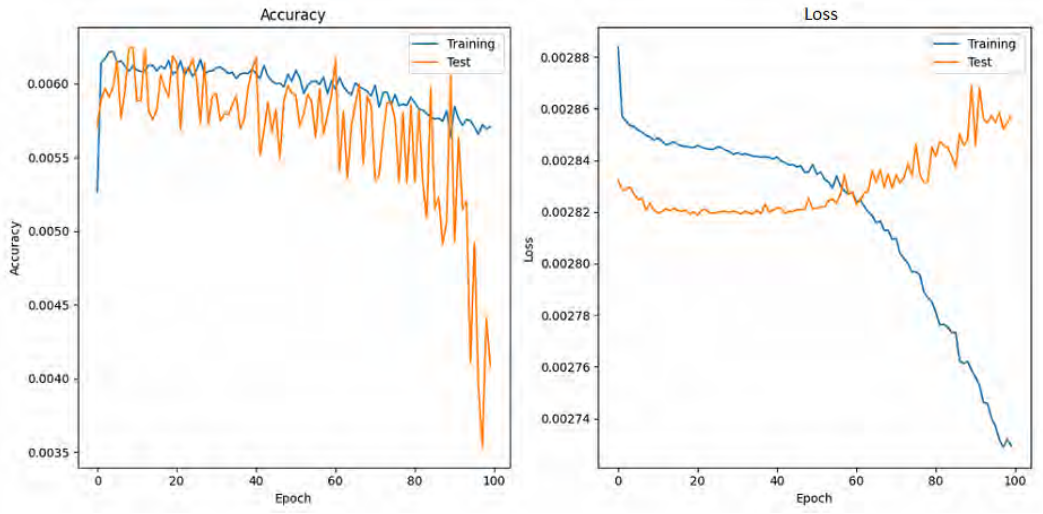

Abbildung 4.6: Accuracy und Loss von der ersten Trainingsiteration

Konfiguration

Bei diesem Durchlauf wird dieselbe Konfiguration wie im vorherigen Beispiel genutzt. Es werden lediglich reine Infrastrukturkarten, ohne einen eingezeichneten Interessensbereich, verwendet. Dies soll die Evaluierung erleichtern, um festzustellen, inwieweit die anderen Karten in die Ergebnisse einfließen, wenn der Bereich nicht so stark eingeschränkt ist wie durch die Infrastrukturkarte.

Ergebnisse

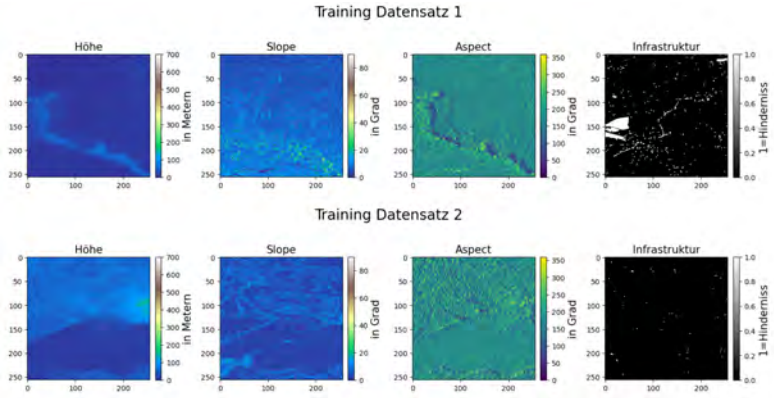

Abbildung 4.7: Beispieldaten aus dem zweiten Experiment

Es wird insgesamt über drei Iterationen simuliert, um auch die Veränderungen nach der ersten Iteration betrachten zu können. Zwei beispielhafte Datensätze sind in 4.7 und die dazugehörigen Ergebnisse in Abbildung 4.8 dargestellt. Hier wird ersichtlich, dass die Ergebnisse sich nicht sonderlich von denen aus dem ersten Experiment unterscheiden. Auch hier sind die Wertebereiche immer noch sehr klein und es kommt auch über die zusätzlichen Trainingsiterationen kaum zu einer Vergrößerung des Wertebereichs. Zudem ist auch hier das einzige, was ansatzweise gelernt wird, die Infrastrukturkarte. Allerdings auch nur sehr schwach, und sobald keine größeren Hindernisse vorhanden sind, sind keine erkennbaren Details in den Ergebnissen vorhanden, die in Zusammenhang mit den Input Features gebracht werden können.

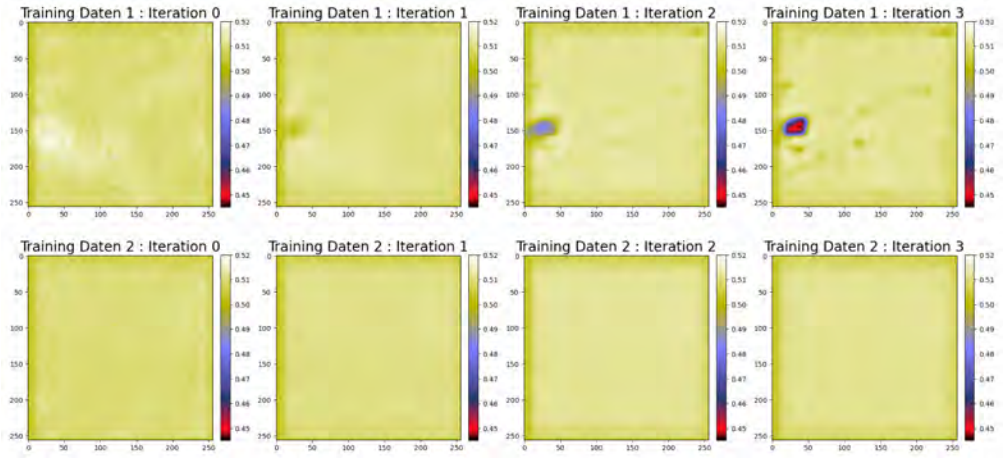

Abbildung 4.8: Die generierten Potentialkarten über drei Iterationen im zweiten Experiment [Skala: 0,445 - 0,52]

Des Weiteren lässt sich über die verschiedenen Trainingsiterationen hinweg keine wesentliche Veränderung feststellen, außer dass immer weniger Details in den Ergebnissen enthalten sind. Dies deutet somit eher auf eine Verschlechterung hin.

4.2.3 Experiment 3

Beim dritten Experiment wird die Netzwerkkonfiguration verändert, um zu testen, ob das Netz dadurch besser lernt. Das Ziel ist es, zum einen den Wertebereich (0 bis 1) auch in den generierten Karten zu erhalten. Zum anderen soll über mehrere Trainingsiterationen eine Verbesserung erkennbar sein.

Konfiguration

In diesem Experiment werden im Vergleich zum vorherigen zwei Veränderungen vorgenommen. Zum einen wird die Netzwerkgröße reduziert, indem die Anzahl der Filter halbiert wird. Zum anderen wird nach jedem Convolutional Layer eine Batch-Normalization-Schicht hinzugefügt. Dies soll für eine bessere Regularisierung sorgen und somit Overfitting verhindern. [52] Diese geänderte Netzkonfiguration ist auch nochmal im Anhang in Abbildung A.1 dargestellt.

Ergebnisse

Auch bei diesem Experiment wird über mehrere Iterationen simuliert, um zu evaluieren, ob dies zu einer Veränderung führt. Zwei beispielhafte Datensätze sind in 4.9 und die Ergebnisse in Abbildung 4.10 dargestellt.

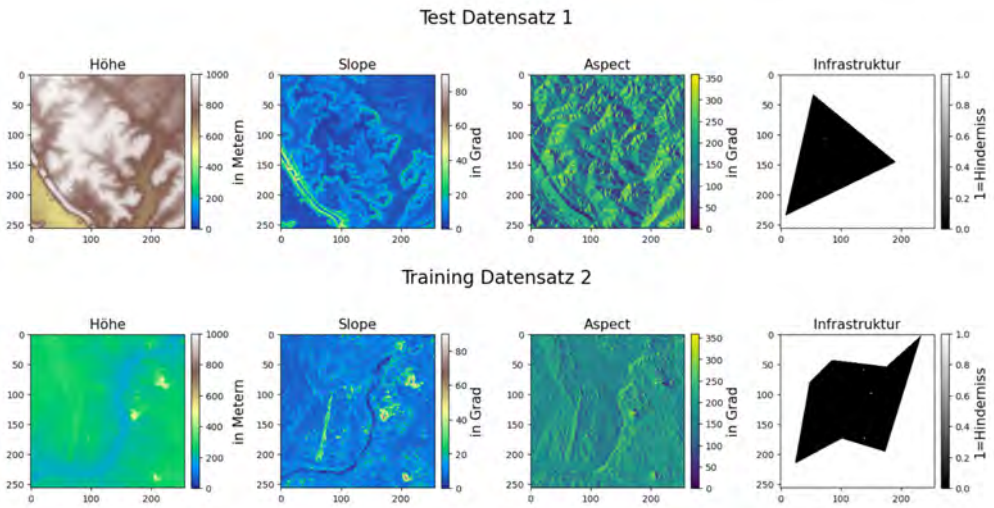

Abbildung 4.9: Beispieldaten aus dem dritten Experiment

Hierbei stellt das obere Beispiel einen Testdatensatz und das untere einen Trainingsdatensatz dar. Beim Betrachten der Ergebnisse fällt kein bedeutender Unterschied zu den vorherigen Beispielen auf. Der Wertebereich ist weiterhin sehr klein und verzerrt. Allerdings wird der Wertebereich mit mehreren Iterationen etwas größer, wie es zu erwarten war.

Inhaltlich haben sich die Ergebnisse der Potentialfelder nicht wesentlich verbessert. Besonders beim Testdatensatz wird die Infrastrukturkarte selbst bei zunehmendem Training

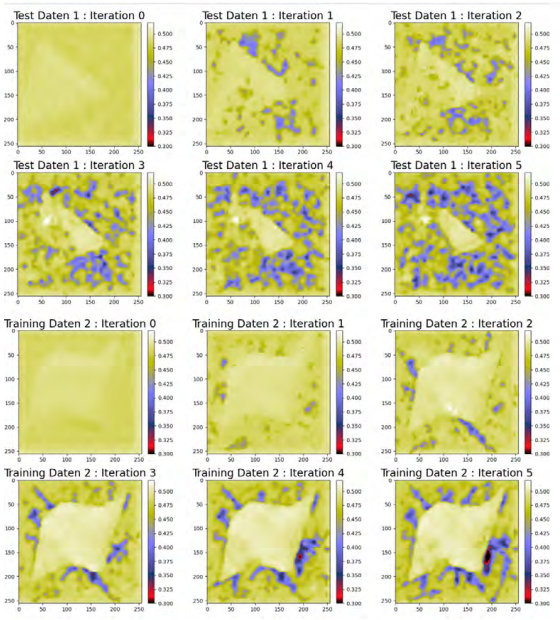

Abbildung 4.10: Die generierten Potentialkarten über fünf Iterationen im dritten Experiment [Skala: 0,3 - 0,52]

schlechter vorhergesagt. Dies verdeutlicht erneut, dass es auch hier wieder zu Overfitting kommt.

Ein weiteres Beispiel, welches dies gut aufzeigt, ist in Abbildung 4.11 zu sehen. Dort wird deutlich, dass die auf der Vorhersage entstandenen Flecken und Striche lediglich die auswendig gelernten Belohnungen der erkundeten Bereiche sind. Das Netz schafft es hierbei nicht, den Zusammenhang zwischen den Features in den Eingangsdaten und den schlechteren oder besseren Werten zu erschließen. Dadurch scheitert das Modell an einer erfolgreichen Generalisierung, was dazu führt, dass die Ergebnisse bei den Testdaten lediglich Flecken aufweisen.

4.2.4 Auswertung

Basierend auf diesen ersten Experimenten lassen sich gewisse Schlüsse ziehen, die als Grundlage dienen, um potenzielle Verbesserungsmöglichkeiten zu erkennen. Es fällt insbesondere auf, dass lediglich die Infrastrukturkarte in den Vorhersagen erkennbar ist. Dies lässt sich darauf zurückführen, dass sie hauptsächlich auf den Maximal- und Minimalwerten basiert, die während des Trainings besonders stark gewichtet werden und

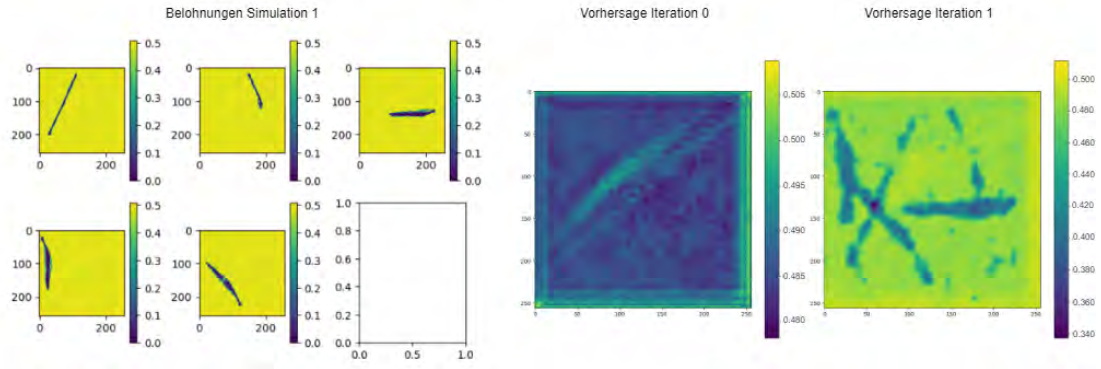

Abbildung 4.11: Links im Bild sind die in der Simulation generierten Belohnungen zu sehen. Rechts sowohl die initiale Vorhersage als auch die Vorhersage nach dem Training unter Berücksichtigung der Belohnungen

somit bereits frühzeitig vom Netzwerk erlernt werden. Ein weiteres auffälliges Ergebnis ist die Konzentration der Werte in einem engen Bereich um 0,5. Diese Anfangswerte entstammen dem untrainierten Netzwerk. Jedoch sollten sich die Werte im Verlauf des Trainings von diesem Bereich entfernen, da durch die Einbeziehung verschiedener Bewertungen der Zahlenbereich erweitert wird und das Netzwerk entsprechend lernen sollte, dies ist jedoch nur sehr eingeschränkt geschehen. Die bestehende Problematik liegt möglicherweise in der begrenzten Größe der erkundeten Bereiche. Diese begrenzte Erkundung führt zu einem Ungleichgewicht zwischen erkundeten und nicht erkundeten Gebieten, wodurch das Netzwerk weiterhin stark von den vorherigen Werten beeinflusst wird und eine Verzerrung entsteht. Zudem besteht das Hauptproblem darin, dass es sehr schnell zum Overfitting kommt und das Netz dadurch die Zusammenhänge zwischen den Input Features und den generierten Karten nicht erkennt.

Zusammengefasst lässt sich ableiten, dass das Netzwerk nicht optimal lernen kann, sodass lediglich die Infrastrukturkarte mit den starken Werten teilweise gelernt wird. Das Overfitting und das damit zusammenhängende unzureichende Lernen des Modells kann einerseits an schlechten oder unzureichenden Eingangsdaten liegen. Andererseits kann es auf eine zu geringe Exploration hinweisen. Basierend auf diesen Ergebnissen werden die folgenden Maßnahmen abgeleitet.

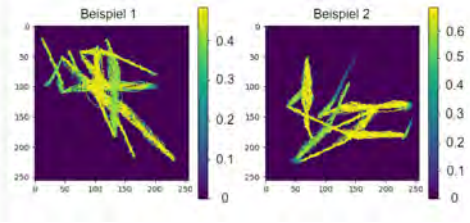

Abbildung 4.12: Belohnungskarten mit mehreren Simulationen pro Karte und ϵ-Wert von 30 %

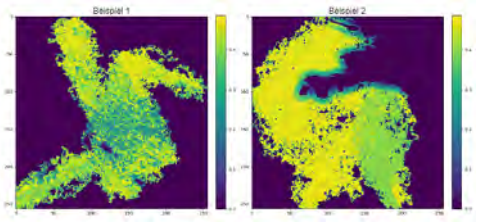

Abbildung 4.13: Belohnungskarten mit mehreren Simulationen pro Karte und ϵ-Wert von 90 %

Größere erforschte Bereiche

Um mit dem Bias der Vorhersagen umzugehen, müssen mehr Bereiche auf der Karte erforscht werden. Hierfür werden zwei Maßnahmen umgesetzt. Zunächst wird die durchgeführten Simulationen zu einer Karte zusammengefasst. Das Ergebnis hiervon ist in Abbildung 4.12 dargestellt.

Zudem wird zur Steigerung der Exploration der ϵ-Wert bei der Wegeplanung von anfänglich 30 % auf 90 % erhöht. Auf diese Weise entsteht ein deutlich größerer erforschter Bereich. Das Ergebnis mit größerem ϵ-Wert ist am Beispiel von drei Belohnungskarten in Abbildung 4.13 zu sehen. Dieser Wert kann über mehrere Trainingsiterationen immer weiter reduziert werden.

Erhöhung der Datenvarianz

Um den Lernerfolg des Netzes zu erhöhen und mit dem Overfitting umzugehen, wird die Varianz der Input Daten erhöht. Hierzu wird die Anzahl an Trainingsdaten erhöht, wofür der Modell Trainingsprozess nochmal angepasst werden muss. Zu Beginn wurden hierfür die gesamten Trainingsdaten in den Arbeitsspeicher geladen, um dann das Modell zu trainieren. Um die Menge an Input Daten zu erhöhen, ohne Ressourcenprobleme zu bekommen, wird eine Sequence genutzt, welche die Daten immer nur Batchweise in den Arbeitsspeicher lädt.

Die Erhöhung an Trainingsdaten ist nur begrenzt möglich, da andernfalls die Simulationszeiten stark erhöht wären, daher wird zusätzlich Data Augmentation eingeführt.

Diese lässt sich für den hier vorliegenden Anwendungsfall sehr gut einsetzen, da die Ausrichtung der Karten irrelevant ist und die Bilder in alle Richtungen gespiegelt werden können, ohne dass das einen Einfluss auf die Validität dieser Daten hat. Entsprechend kann jedes Originalbild auf sieben Arten gespiegelt werden. Dies sorgt dafür, dass der Trainingsdatensatz mit geringem Aufwand stark vergrößert werden kann, da so für jeden Datensatz insgesamt acht verschiedene Optionen entstehen, mit denen trainiert werden kann.Ein Beispiel hierfür ist in Abbildung 4.14 zu sehen.

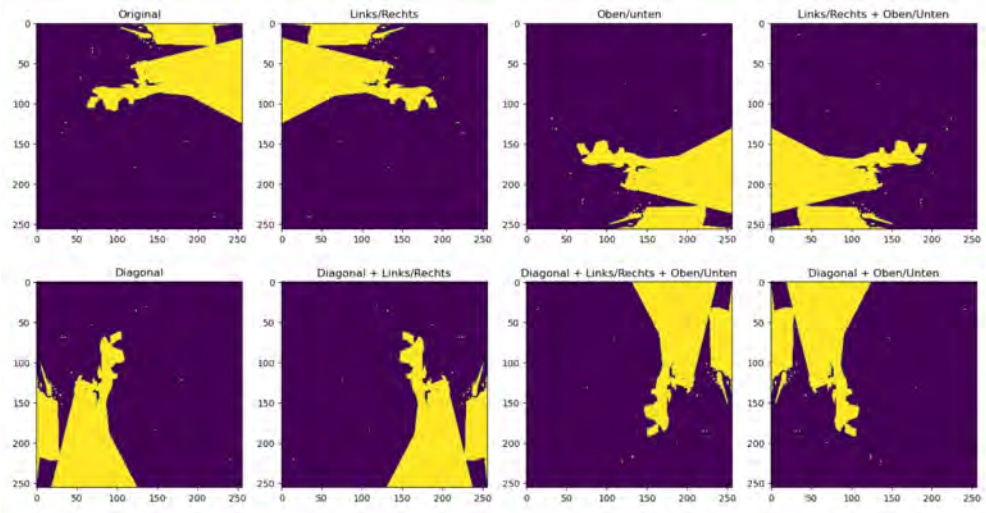

Abbildung 4.14: Beispieldarstellung von allen genutzten Data Augmentation Möglichkeiten

Die Data Augmentation wird implementiert, indem beim Laden der Trainingsbatches, mit der Sequence, für jedes Bild zufällig entschieden wird, ob eine Veränderung vorgenommen wird und gegebenenfalls welche. Die angepasste Implementierung ist in Codeblock 4.2 grundlegend dargestellt.

Codeblock 4.2: Implementierung des Trainierens des Neuronalen Netzes mit einer Sequence

```
class OwnSequence(tf.keras.utils.Sequence):
    #... Initialization of relevant Variables ...
    def __init__(...):
        ... Declaration of relevant Variables
    def __len__(...):
        return number of iterations per epoch
    def load_and_augment_data(..):
```

```
        #...Load files from numpy binarys...
        #...Augment data based on random selection
        #...Reformatter into multidimensional Array

    def __getitem__(self, idx):
        train_labels = np.empty([self.batch_size,256,256])
        train_data_array= np.empty([self.batch_size,256,256,4])
        for i in range(batch_size):
            file_name = random(file_names)
            train_data_array[i] = load_and_augment_data(..)
            train_labels[i] = corresponding reward map
        return train_data_array, train_labels

if __name__ == "__main__":
    model_version = load from file
    model = tf.keras.models.load_model(model_path)
    sequence = OwnSequence(...)
    history = model.fit(sequence, batch_size, epochs, worker)
    #...save model and history...
    #... Make prediction and save result to new files ...
    #...Update Iteration number...
```

Testen alternativer initialer Karte

Als weitere Maßnahme sollen noch alternative Startkarten getestet werden. Hierfür bietet es sich an zu Beginn mit einer leeren Karte zu starten. Dies ermöglicht es genau zu sehen, was das Netz gelernt hat. Eine weitere Möglichkeit ist es mit zufällig initialisierten Karten zu starten. Dadurch wird bereits der gesamte Wertebereich abgedeckt sein und es kann keine Verzerrung entstehen.

4.3 Training basierend auf leeren Karten - Experiment 4

Im nächsten Experiment werden unerforschten Karten untersucht. Dieses Vorgehen soll dafür sorgen, dass kein Bias entsteht, wie beim untrainierten Netz.

In diesem Experiment sind die in Abschnitt 4.2.4 erläuterten Maßnahmen umgesetzt. Hierbei soll primär die Tauglichkeit dieses Ansatzes überprüft und evaluiert werden, ob

die oben genannten Nachteile aufgehoben werden können. Zudem werden unterschiedliche Trainingslängen verglichen.

4.3.1 Konfiguration

Wie auch im vorherigen Experiment, wird hier wieder mit der kleineren Netzkonfiguration gearbeitet, da diese deutlich schneller trainiert und nicht zu Ressourcenproblemen führt. Im vorherigen Experiment hat Batch-Normalization zu besseren Ergebnissen geführt. Daher wird mit dieser Konfiguration weitergearbeitet, wie im Anhang in Abbildung A.3 visualisiert. Zudem werden die ersten zwei Iterationen mit einer Batchgröße von 64 und über 200 Epochen trainiert. Die Trainingszeit sind hierbei im Vergleich zu den vorherigen Experimenten aufgrund der größeren Datenkomplexität erhöht. In der dritten Iteration wird dann insgesamt über 800 Epochen trainiert und in regelmäßigen Zwischenschritten werden neue Vorhersagen generiert, um das Ergebnis an den verschiedenen Zwischenzuständen vergleichen zu können. Dieses Experiment sollte evaluieren, welche Trainingslänge am besten geeignet ist, da in den vorherigen Experimenten auffiel, dass besonders nach den ersten Iterationen kaum noch Veränderungen in den Ergebnissen zu finden waren.

Bei den Eingangsdaten wird wieder mit reinen Infrastrukturkarten gearbeitet, da dies die Auswertung deutlich vereinfacht, da nicht so große Bereiche automatisch schlecht bewertet werden.

4.3.2 Ergebnisse

In Abbildung 4.15 sind beispielhafte Datensätze dargestellt. Dabei handelt es sich bei den oberen zwei Datensätzen um Testdaten, die nicht für das Training verwendet werden und bei den unteren zwei Beispielen um Daten, auf denen die Belohnungen generiert werden und mit denen das Modell trainiert wird.

Die zu diesen Daten generierten Potentialkarten sind in Abbildung 4.16 dargestellt. Dort sind für die vier Datensätze jeweils die Initiale Karte und dann die Ergebnisse der Ersten beiden Trainingsiterationen zu sehen.

Es fällt auf, dass die Ergebnisse für alle vier Datensätze vergleichbar sind und keine gravierenden Unterschiede zu finden sind, die auf Overfitting hindeuten würden. Zudem ist

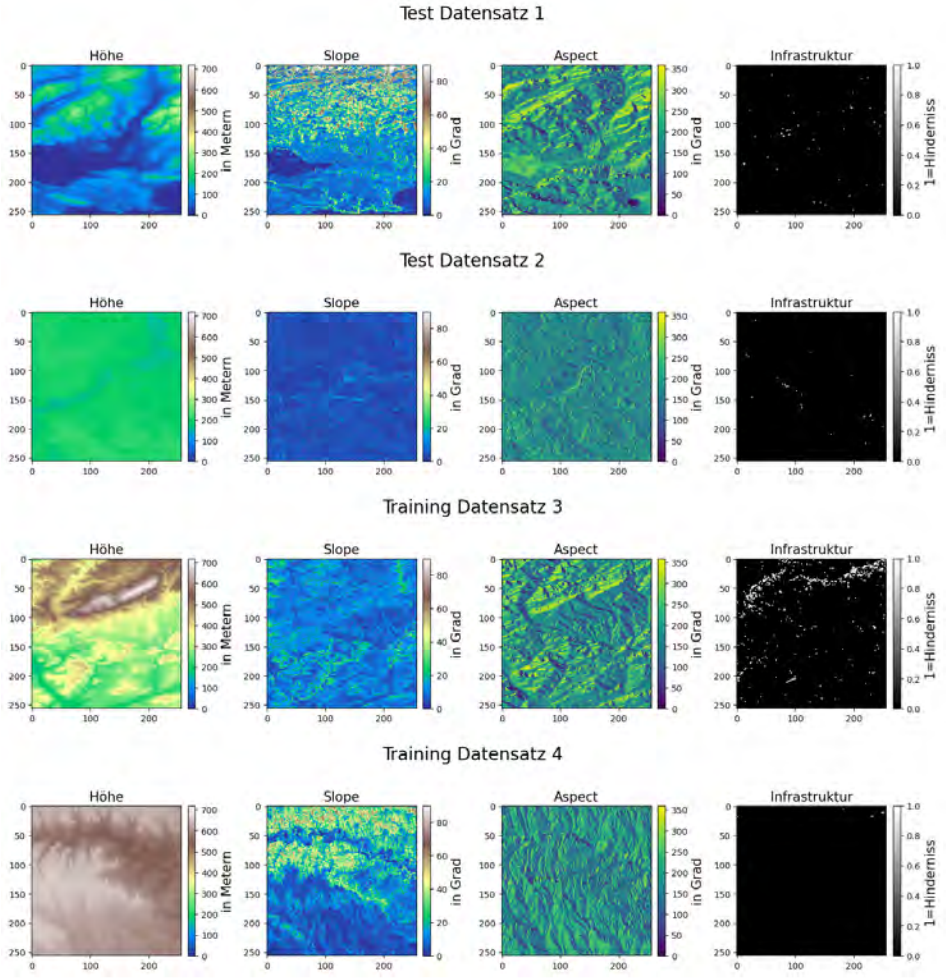

Abbildung 4.15: Beispieldaten für das vierte Experiment

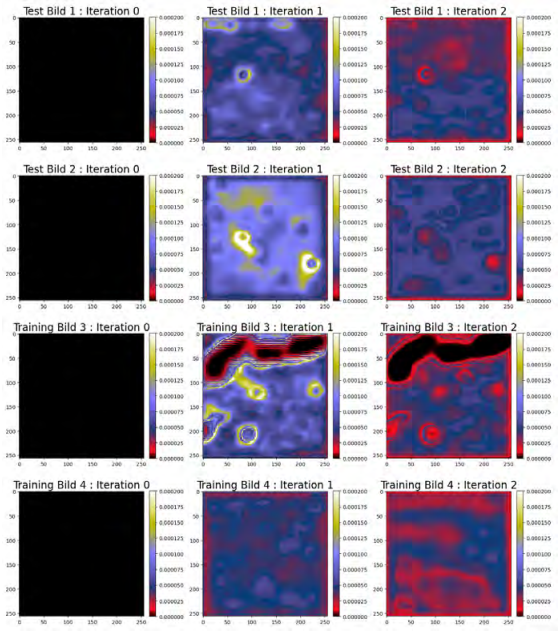

Abbildung 4.16: Die generierten Potentialkarten über zwei Iterationen im vierten Experiment [Skala: 0 - 0,0002]

nun auch eine deutliche Verbesserung zwischen der ersten und der zweiten Trainingsiteration sichtbar, da in der zweiten Iteration mehr unterschiedliche Bereiche erkennbar sind. Wie auch in den früheren Experimenten, sind die Infrastrukturhindernisse diejenigen, die sich am stärksten auf die Ergebniskarten auswirken. Allerdings wird hier auch die Steigung teilweise korrekt erkannt und in der zweiten Iteration schlechter bewertet, wie im zweiten Beispiel zu sehen ist, bei dem die obere Hälfte der Karte schlechter bewertet wird als die untere. Dennoch sind auch hier die Wertebereiche der Vorhersagen stark verzerrt und sehr klein.

Dies wird auch bei der Betrachtung des Lernerfolgs des Netzes deutlich. In Abbildung 4.17 sind die Trainings-Accuracy und das Loss für beide Iterationen dargestellt. Dabei wird ersichtlich, dass bei der ersten Iteration noch eine annähernd gut aussehende Kurve entsteht. Allerdings fällt hier schon auf, dass eine kürzere Trainingszeit vermutlich ausgereicht hätte, da die Kurve recht schnell stagniert. Daher wird dieser Faktor bei der dritten Trainingsiteration genauer untersucht, was im folgenden Abschnitt näher erläutert wird. Bei der zweiten Trainingsiteration ist allerdings kaum ein Lernerfolg sichtbar. Zudem sind in dieser Kurve die besonders starken Schwankungen auffällig.

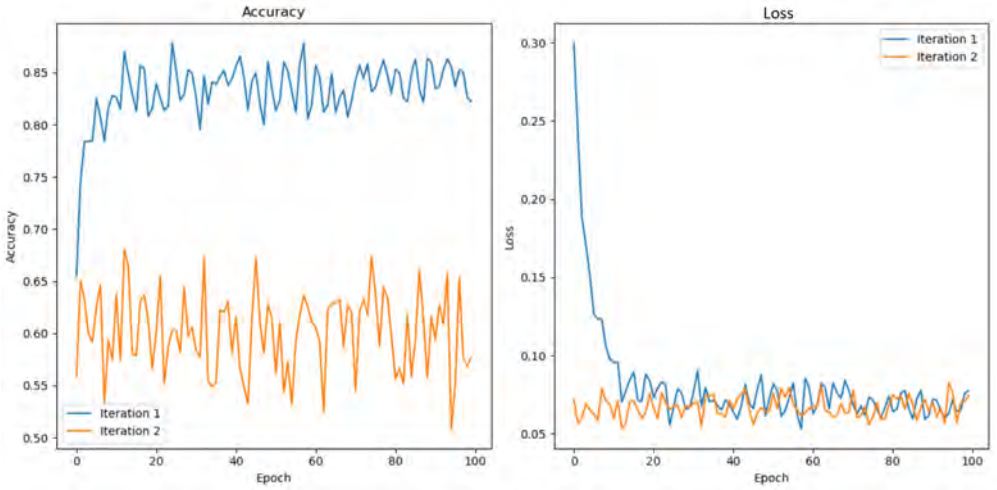

Abbildung 4.17: Accuracy und Loss Kurven aus dem vierten Experiment

4.3.3 Vergleich verschiedener Trainingslängen

Um noch einmal zu evaluieren, welche Trainingsdauer den größten Erfolg bietet und wie sich die Ergebnisse unterscheiden, wird in der dritten Iteration insgesamt über 800 Epochen trainiert. Die Ergebnisse werden an mehreren Zwischenschritten gespeichert. Die Accuracy- und Loss-Kurve dazu ist in Abbildung 4.18 zu finden. Auch hier ist zu Beginn noch eine starke Verbesserung zu erkennen, die jedoch relativ schnell stagniert. Zudem sind ebenfalls sehr starke Schwankungen zu sehen.

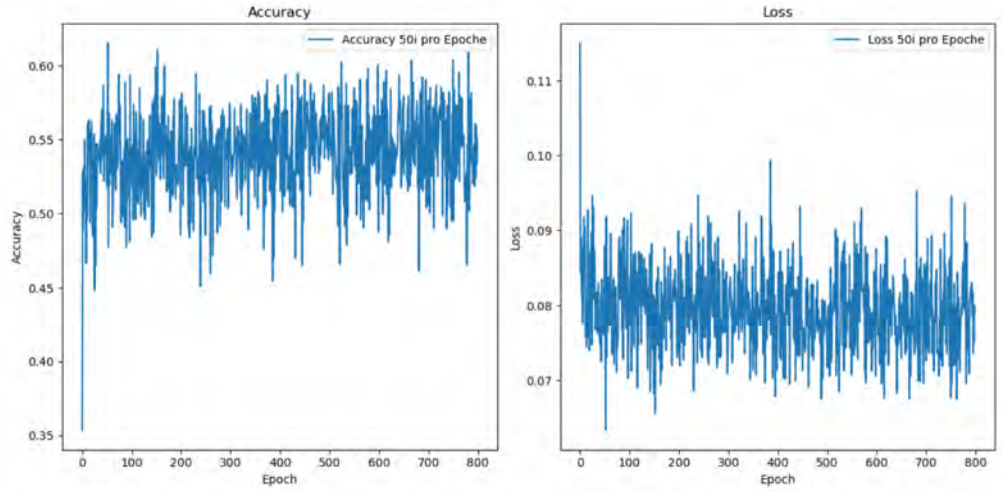

Abbildung 4.18: Accuracy und Loss der 3. Trainingsiteration

Es ist jedoch auch interessant die Zwischenstände der einzelnen Bilder zu betrachten. In Abbildung 4.19 sind die verschiedenen Vorhersagen für den Input Datensatz 1, aus Abbildung 4.15 zu sehen. Die Vorhersagen sind nach den unterschiedlichen Anzahlen an Trainingsepochen abgebildet. Das Gleiche ist in Abbildung 4.20 für den Datensatz 3 zu finden.

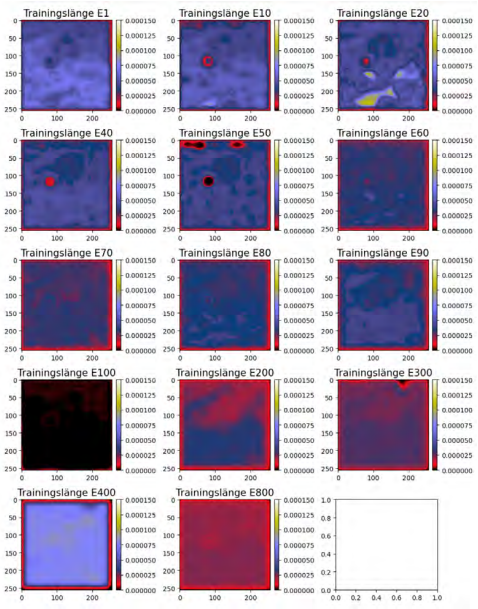

Abbildung 4.19: Verschiedene Trainingslängen Test Datensatz 1 [Skala: 0 - 0,00015]

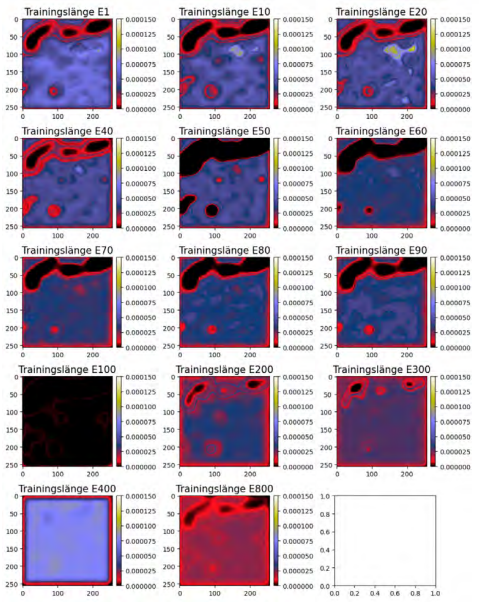

Abbildung 4.20: Verschiedene Trainingslängen Traings Daten 3 [Skala: 0 - 0,00015]

Bei dem linken Datensatz (Abbildung 4.19) handelt es sich um einen Test- und bei dem rechten (Abbildung 4.20) um einen Trainingsdatensatz. Hierbei wird ersichtlich, dass die Vorhersagen für den Trainingsdatensatz allgemein deutlich detaillierter sind. Zudem wird bei den Testdaten klar, dass bereits nach 90 Epochen Overfitting auftritt und auf den Testdaten nichts mehr zu erkennen ist. Spannend ist auch, dass die Bilder bei 100 Epochen keinerlei Details enthalten. Während die Trainingsdaten danach wieder Details in den Potentialfeldern aufweisen, bleiben die Testdaten leer. Des Weiteren besteht auch hier noch das Problem, dass die Werte in einem sehr kleinen Bereich bleiben und dieser kleiner wird, je länger trainiert wird. Allgemein lässt sich beim Vergleich der Trainings- und Testdaten festhalten, dass die Ergebnisse bei bis zu 40 Epochen vom Detailgrad her noch relativ ähnlich sind. Dies ändert sich jedoch bei weiterem Training sehr schnell.

Daher lässt sich zusammenfassend sagen, dass hierbei bereits ein Training von 30 bis 40 Epochen ausreichend ist.

4.3.4 Auswertung

Basierend auf den Ergebnissen dieses Experiments lässt sich festhalten, dass trotz Data Augmentation und einer Erhöhung der Anzahl der Input Daten die Tendenz zum Overfitting bestehen bleibt. Es ist weiterhin erkennbar, dass Unterschiede zwischen Trainings- und Testdaten bestehen, wenn auch nicht mehr so stark wie bei den vorausgegangenen Experimenten zuvor. Dennoch hat der erhöhte Erkundungsbereich bei den Simulationen dazu geführt, dass mehr Parallelen zu den Input Features auf den Potentialfeldern zu erkennen sind als in den vorherigen Beispielen. Allerdings kommt es weiterhin zu einer Verzerrung des Wertebereichs. Dies könnte entweder daran liegen, dass der erkundete Bereich nicht ausreicht oder dass zu viele Bereiche schlecht bewertet werden. Dies führt dazu, dass das Netz neben den initialen Nullwerten verstärkt mit Nullen aus den Bewertungen lernt.

Des Weiteren fallen die nur mäßige Accuracy und die starken Schwankungen dieser auf. Die schlechte Accuracy im Allgemeinen deutet darauf hin, dass die Konfiguration und die Hyperparameter noch einmal angepasst werden sollten, um zu evaluieren, ob es hierfür besser geeignete Einstellungen gibt. Eine mögliche Ursache für die starken Schwankungen der Kurve könnte sein, dass sowohl die Batches als auch die Data Augmentation zufällig bestimmt werden. Dies könnte zu einem Ungleichgewicht führen, zum Beispiel indem einige Datensätze phasenweise häufiger auftreten und dadurch die Accuracy erhöhen, oder dass einige Elemente seltener auftreten, sodass diese dann kurzfristig zu Verschlechterungen führen.

Zusätzlich muss beachtet werden, dass bei den Belohnungskarten immer nur Teile erkundet sind. Dies könnte beispielsweise dazu führen, dass, wenn das Netz es schafft zu generalisieren und zu lernen, dass eine gewisse Ausprägung in den Features schlecht ist, dieser Bereich auf dem Ergebnis als schlecht vorhergesagt wird, obwohl er gar nicht in der Belohnungskarte bewertet ist. Dadurch entsteht eine Differenz zwischen der Vorhersage und der tatsächlichen Bewertung.

Zusammengefasst lässt sich ableiten, dass im nächsten Experiment primär verschiedene Hyperparameter getestet werden sollten, um zu evaluieren, mit welchen die Ergebnisse

verbessert werden können. Zusätzlich sollten zufällige Karten genutzt werden, um die Verzerrung des Wertebereichs zu korrigieren.

4.4 Training basierend auf zufälligen Karten - Experiment 5

Zuletzt wird mit zufällig initialisierten Karten gestartet, um zu Evaluieren, welche Variante sich am besten eignet.

4.4.1 Konfiguration

Als Basis wird hier die grundlegende kleinere Netzkonfiguration, wie in den vorherigen Experimenten, genutzt. Zusätzlich werden verschiedene Anpassungen der Netzwerkarchitektur und -konfiguration getestet, um zu evaluieren, welchen Einfluss diese auf das Ergebnisse haben und was besser geeignet ist. Außerdem werden auch hier verschiedene Trainingslängen ausprobiert, um zu evaluieren wie sich das Zusammenspiel zwischen den Konfigurationen und der Trainingslängen verhält.

Die Untersuchung und Evaluierung verschiedener Konfigurationen sowie Trainingslängen ist ein wesentlicher Schritt, um das Verständnis für den Zusammenhang zwischen den verschiedenen Parametern und Hyperparametern zu vertiefen. Dies kann helfen, die optimalen Einstellungen für das Training und die Konfiguration des Neuronalen Netzes zu ermitteln.

Die getesteten Konfigurationsoptionen umfassten eine Vielzahl von Parametern, die isoliert und in Kombination miteinander ausprobiert werden. Im Folgenden sind die verschiedenen Komponenten aufgelistet.

1. Optimierer
    - Adam
    - Adadelta
2. Zusätzliche Layer
    - Batch Normalization

- Dropout mit 25% und 50%

3. Loss Funktion
    - Mean Squared Error (MSE)
    - Mean Absolute Error (MAE)
4. Dilation 2 und 3
5. Netzkonfiguration
    - Größere Konfiguration: min. 32 Filter, max. 128 Filter
    - Kleinere Konfiguration: min. 8 Filter, max. 32 Filter
    - Andere Konfiguration: Start mit einem 1x1x1 Filter, sodass zu Beginn, erstmal die Ebenen zusammegefasst werden

Die verschiedenen Netzwerke werden insgesamt über 150 Epochen getestet. Zusätzlich werden die Zwischenstände nach 20, 60 und 100 Epochen gespeichert.

4.4.2 Ergebnis

Zu Beginn wird eine Simulation mit zufällig generierten Potentialfeldern durchgeführt. Anschließend werden verschiedene Netzwerkkonfigurationen mithilfe der resultierenden Bewertungskarten trainiert und neue Vorhersagen generiert, die daraufhin verglichen werden können.

Als Grundlage für diesen Vergleich dienen die in Abbildung 4.21 dargestellten Datensätze. Dabei repräsentieren die Datensätze 1 und 2 die Testdaten, während die Datensätze 3 und 4 die Trainingsdaten darstellen. Bei den Vorhersagen wird erwartet, dass die Hindernisse aus der Infrastrukturkarte bereits erkennbar sind. Zudem sind beispielsweise bei dem zweiten und dritten Datensatz im oberen Bereich des Bildes stärkere Steigungen erkennbar. Diese sollten idealerweise auch schlechter vorhergesagt werden.

In der Abbildung 4.22 sind die zugehörigen Belohnungskarten zu den unteren beiden Trainingsdatensätzen abgebildet. Hier fällt direkt auf, dass ein erster Nachteil dieses Vorgehens darin besteht, dass die Belohnungskarten deutlich ungenauer sind als die aus den vorangegangenen Experimenten. Da die Belohnungen auf die vorherigen Werte addiert werden, bleibt das Rauschen bestehen, was das Training erschweren kann.

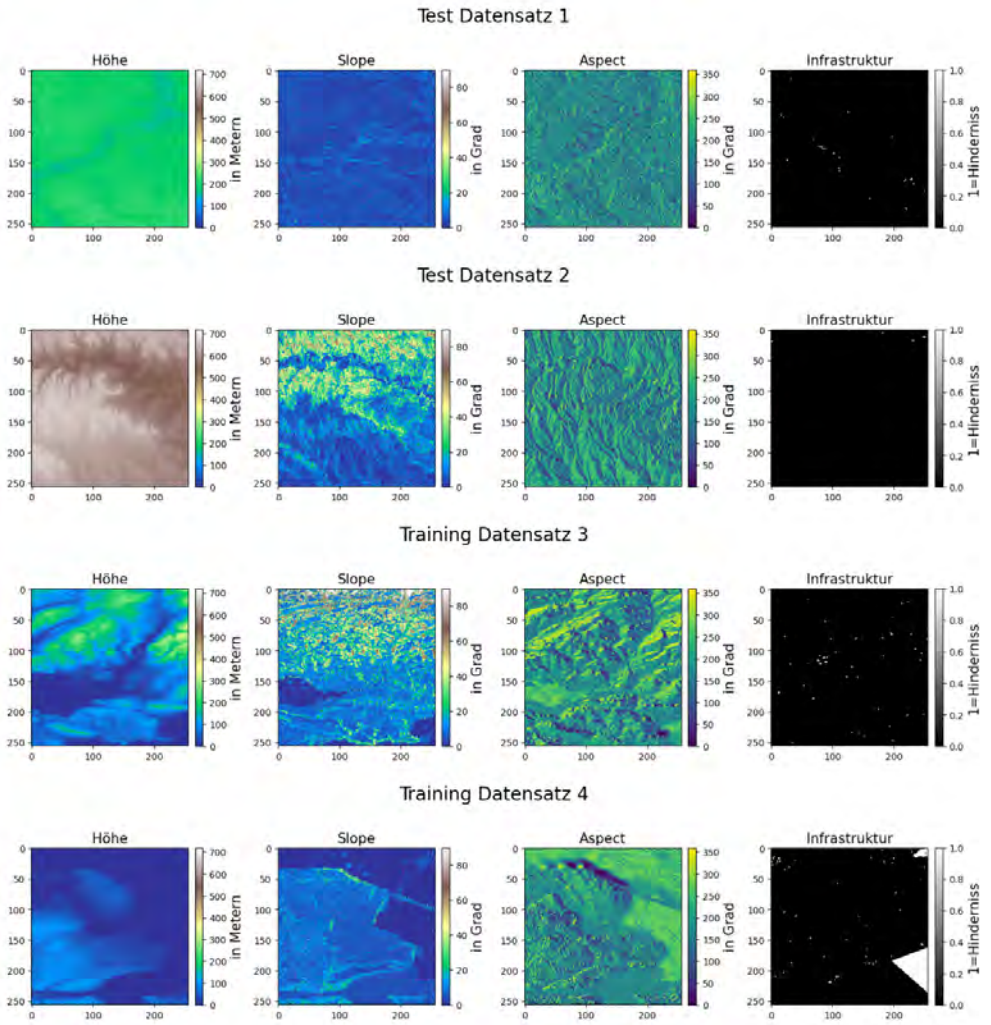

Abbildung 4.21: Input Daten für die Evaluierung der Ergebnisse aus dem fünften Experiment

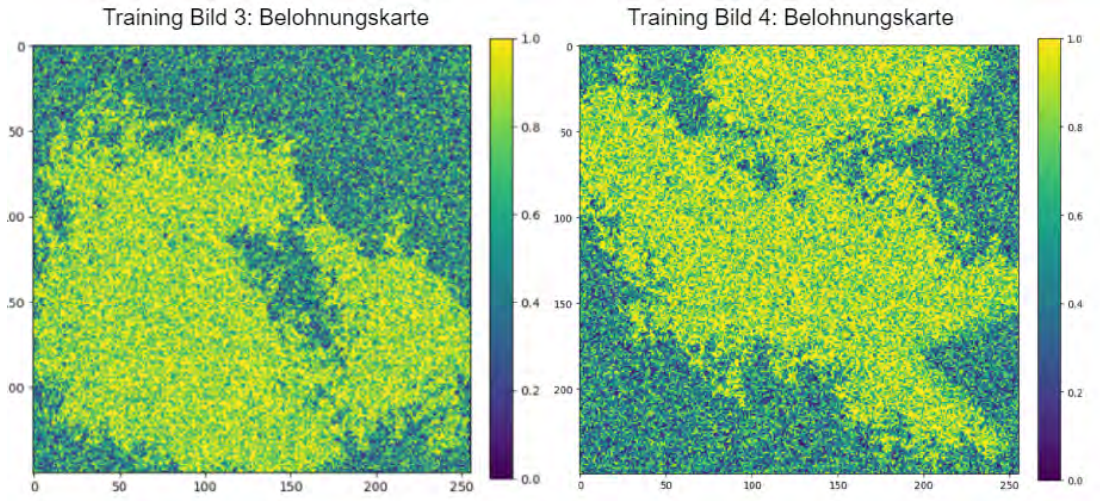

Abbildung 4.22: Belohnungskarten zu den Trainingsdatensätzen 3 und 4

Im Anhang unter A.2.2 befindet sich eine Übersicht über die Accuracy- und Loss-Kurven aller Modelle sowie alle Ergebnisse zu den Beispieldaten in groß. Beim Betrachten der Loss-Kurven des größeren Modells in Anhang Abbildung A.11 fällt auf, dass zwei der Kurven bereits nach 20 Epochen enden. Dies resultiert aus einem Auftreten von Exploding Gradients, infolgedessen das Modell nur noch NaN-Werte ausgibt. Es werden auch weitere Modelle mit der großen Architektur und einer kleineren Architektur trainiert, bei denen es ebenfalls sehr schnell zu diesem Phänomen kam. Daher sind diese Modelle nicht in den Abbildungen enthalten.

Allgemein führt der Einsatz der zufälligen Startkarten zu dem erwünschten Erfolg, dass bei den meisten Vorhersagen kein so starker Bias entsteht und ein deutlich besserer Wertebereich erreicht wird. Allerdings wird auch ersichtlich, dass dem Modell das korrekte Generalisieren auf Basis der Belohnungen schwerer fällt und die Ergebnisse für verschiedene Datensätze häufig stark variieren. Es zeigt sich, dass einige Modelle lediglich einige Kombinationen von Merkmalen gut lernen. Diese Beobachtung wird durch die Betrachtung der Accuracy-Kurven unterstützt, da viele davon stark schwanken. Zudem zeigen viele der Kurven kaum Verbesserungen, sondern bleiben konstant.

Bei den Loss-Kurven lassen sich zwei Hauptgruppen erkennen, in denen die meisten Kurven liegen. Lediglich die Konfiguration mit Batch Normalization, Adadelta und Mean Absolute Error (MAE) (orange in A.7) sowie die alternative Architektur mit Dilation,

Batch Normalization, Adam als Optimierer und MAE (grün in A.10) fallen besonders schlecht aus.

Auffällig ist auch, dass die Modelle mit Dilation im Allgemeinen nicht so gut abschneiden, was sowohl beim Betrachten der Kurven als auch der Bilder deutlich wird. Die Loss-Kurven dieser Modelle zeigen zwar eine stetige Verbesserung, doch die Accuracys bleiben im Durchschnitt weitgehend gleich und weisen besonders starke Schwankungen auf. Die Bilder bestätigen diese Vermutung größtenteils. Es wird ersichtlich, dass die Vergrößerung der Kernelgröße vermutlich das Problem darstellt. Diese führt dazu, dass ein zu großer Bereich betrachtet wird und Details verloren gehen. Ein größerer Kernel hat lediglich in Kombination mit einer Verkleinerung der Anzahl an Filtern zu akzeptablen Ergebnissen geführt. Die beiden Modelle mit Dilation und einer Kernelgröße von 3 liefern bessere Ergebnisse als die anderen Dilation-Modelle.

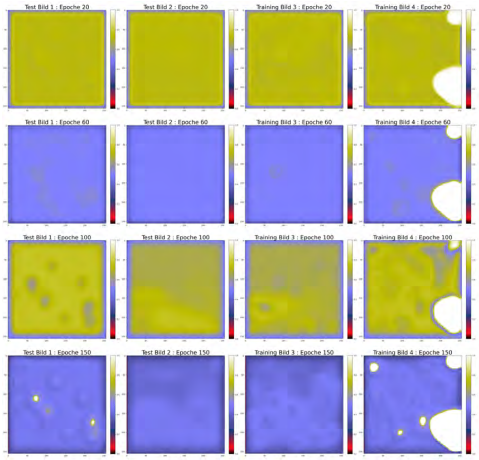

Abbildung 4.23: Potentialkarten Netz: Dilation 2, Batch Normalization, Adam und MSE [Skala: 0 - 1]

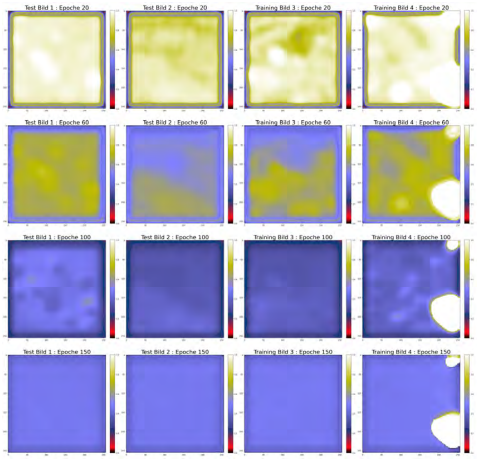

Abbildung 4.24: Potentialkarten Netz: Dilation 2, Batch Normalization, Adam und MAE [Skala: 0 - 1]

Bei der Analyse der Beispielbilder in Abbildung 4.23 wird deutlich, dass die Modellergebnisse im Verlauf des Trainings an Genauigkeit gewinnen. Zudem sind keine signifikanten Unterschiede zwischen den Trainings- und Testdaten erkennbar. Insbesondere bei den Datensätzen mit nur wenigen Infrastrukturhindernissen (Datensatz 2 und Datensatz 3 in Abbildung 4.21) lassen sich im Verlauf des Trainings Zusammenhänge mit den Input Daten wahrnehmen, wodurch die Bereiche mit steilen Steigungen in der letzten Vorhersage präzise antizipiert werden. Hingegen werden die Infrastrukturhindernisse der anderen

beiden Datensätze fälschlicherweise als positive Bereiche identifiziert. Bei der Verwendung der gleichen Konfiguration mit MAE als Loss-Funktion, anstelle von Mean Squared Error (MSE), konvergierten die Ergebnisse nach der 150. Epoche zu einem Wert von nur 0,5, wodurch keine spezifischen Merkmale mehr erkennbar sind, wie in Abbildung 4.24 zu sehen ist. In diesem Fall hat das Netzwerk keinen klaren Zusammenhang mit den zugrunde liegenden Merkmalen gelernt, sondern gibt lediglich Durchschnittswerte aus. Bei diesem Modell ist eine geringere Trainingsdauer von Vorteil und die Ergebnisse bis zur Epoche 100 sind signifikant besser.

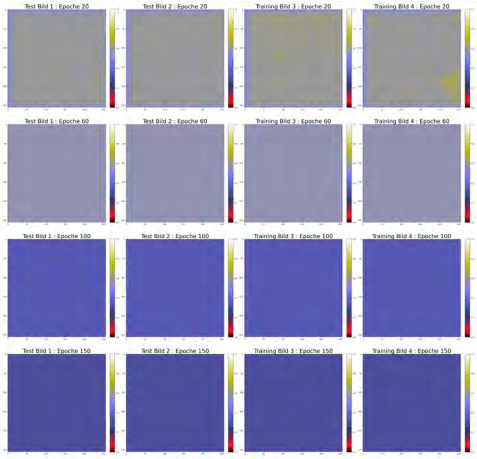

Abbildung 4.25: Potentialkarten mit der grundlegenden Netzkonfiguration und Adam [Skala: 0 - 1]

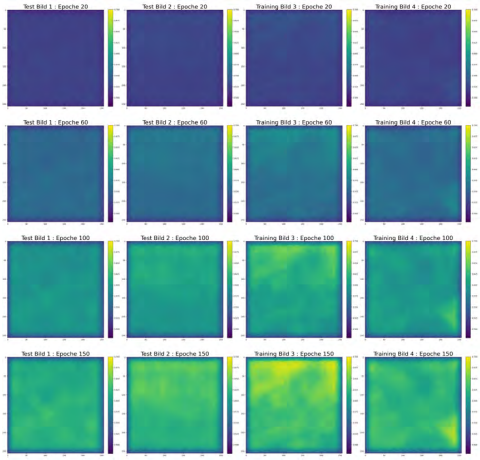

Abbildung 4.26: Potentialkarten grundlegende Netzkonfiguration und Adadelta [Skala: 0 - 1]

Angesichts der Tatsache, dass Modelle, die eine Vielzahl von verschiedenen Elementen enthalten, größtenteils schlecht abschneiden, liegt die Vermutung nahe, dass die grundlegende Konfiguration möglicherweise effektiver ist. Diese Basis-Konfiguration wird sowohl mit Adam als auch mit Adadelta als Optimierer getestet. Die Ergebnisse in den Abbildungen 4.25 und 4.26 verdeutlichen, dass beide Modelle nur mäßige Leistungen erbringen, wobei insbesondere das Adam-Modell keine verwertbaren Ergebnisse liefert. Adadelta scheint bessere Vorhersagen zu generieren.

Die Auswirkungen des Wechsels des Optimierers zwischen Adam und Adadelta werden auch bei der Kombination mit Batch Normalization deutlich. Auch hier ist die Veränderung der Ergebnisse, im Vergleich zum vorherigen initialen Zufallsbild (RL Iteration 0), bei Adadelta weniger stark ausgeprägt als bei Adam (vgl. Abbildung 4.28). Hierbei

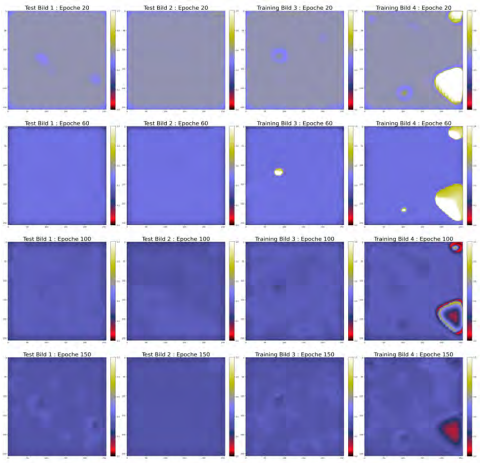

Abbildung 4.27: Potentialkarten Netz: Batch Normalization, Adam und MSE Loss [Skala: 0 - 1]

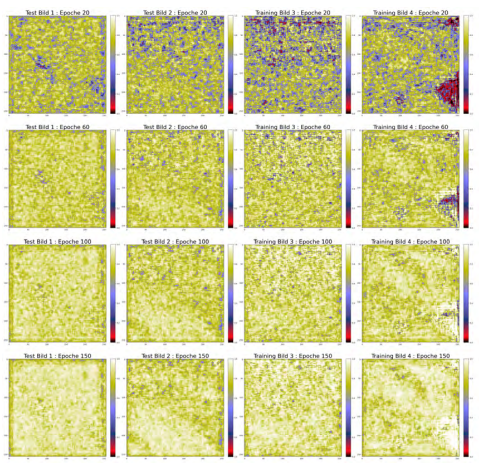

Abbildung 4.28: Potentialkarten Netz: Batch Normalization, Adadelta und MSE [Skala: 0 - 1]

sind die zufälligen Werte vergleichbar mit Rauschen. In dieser Kombination mit Batch Normalization gelingt es Adam deutlich besser das Rauschen zu entfernen als Adadelta. Wie in Abbildung 4.27 zu sehen ist, erreicht diese Kombination nochmal eine deutliche Verbesserung zu der Variante ohne Batch Normalization. Allerdings benötigt Adam auch deutlich mehr Zeit, um zu lernen. Das Ergebnis ist allerdings signifikant besser.

Ein Nachteil von Adam liegt weiterhin in dem begrenzten Wertebereich. Diese Einschränkung kann in dieser Konfiguration durch die Verwendung von MAE als Loss-Funktion verbessert werden. Wie in Abbildung 4.29 zu sehen ist, führt dies zu einem größeren Wertebereich. Im Zusammenhang mit Adadelta führt dies jedoch dazu, dass die gesamten Werte bei längeren Trainingszeiten in Richtung des Maximums streben, wie in Abbildung 4.30 dargestellt.

Des Weiteren werden zu dieser Netzwerkkonfiguration verschiedene Modelle mit Dropout getestet. Die reinen Dropout-Modelle in Verbindung mit Adam schnitten dabei nicht so gut ab. Dies wird auch bei Betrachtung der Accuracy deutlich, die eine auffällige Stetigkeit aufweist. Dabei fällt die Korrelation zwischen den Bildern und der Kurve erneut auf. Beispielsweise sind die Bilder von Dropout 25 mit Adam nach 150 Epochen einfarbig geworden (vgl. Abbildung 4.31) und die Kurve zeigt kurz zuvor einen starken Abfall. Im

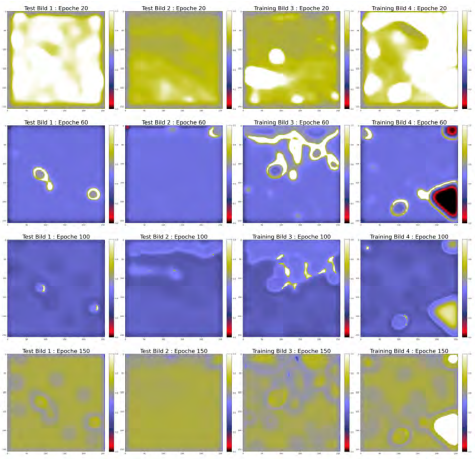

Abbildung 4.29: Potentialkarten Netz: Batch Normalization, Adam und MAE [Skala: 0 - 1]

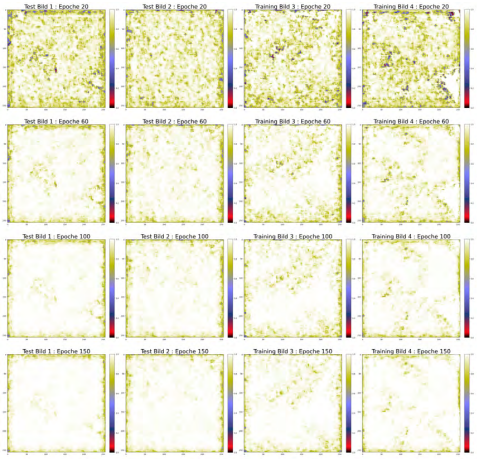

Abbildung 4.30: Potentialkarten Netz:Batch Normalization, Adadelta und MAE [Skala: 0 - 1]

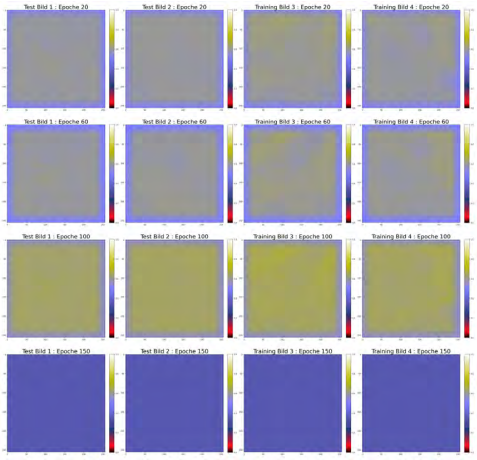

Abbildung 4.31: Potentialkarten Netz: Dropout mit Adam und MSE [Skala: 0 - 1]

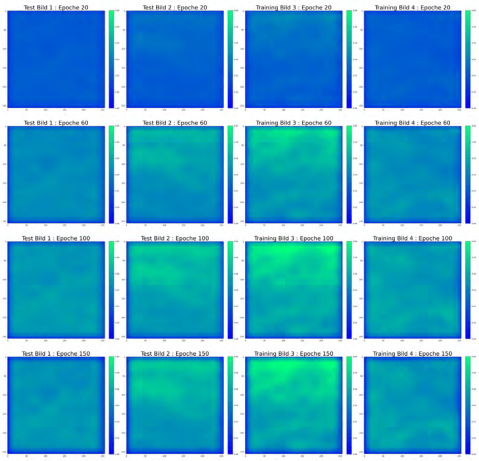

Abbildung 4.32: Potentialkarten Netz: Dropout mit Adadelta und MSE [Skala: 0.48 - 0.6]

Gegensatz dazu ähnelt Dropout mit Adadelta (Abbildung 4.32) der gleichen Variante ohne Dropout (Abbildung 4.26) und weist keine markanten Unterschiede auf.

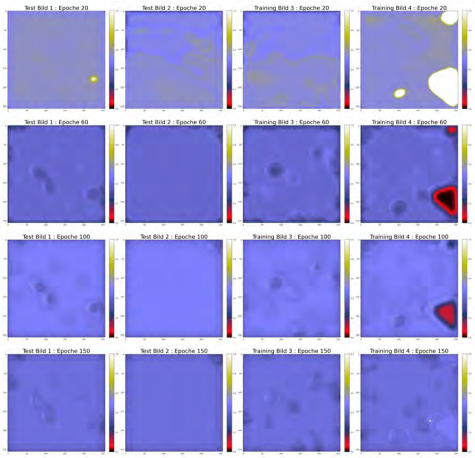

Abbildung 4.33: Potentialkarten Netz: Batch Normalization, Dropout mit Adam [Skala: 0 - 1]

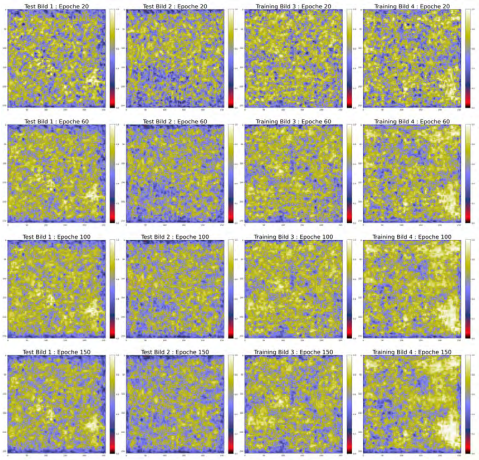

Abbildung 4.34: Potentialkarten Netz: Batch Normalization, Dropout mit Adadelta [Skala: 0 - 1]

Beim Vergleich zwischen reinem Dropout und Dropout in Kombination mit Batch Normalization ist es interessant zu beobachten, dass während dies bei Adam zu einer Verbesserung der Ergebnisse führt, bei Adadelta der gegenteilige Effekt auftritt. Bei Adadelta sind deutlich weniger Kontraste erkennbar und die Ergebnisse enthalten fast ausschließlich Rauschen (vgl. Abbildung 4.34). In Abbildung 4.33 ist es spannend zu sehen, wie sich die Vorhersagen während des Trainings stark verändern. Während die Steigungswerte nach 20 Epochen nochdeutlich erkennbar sind, ändert sich dies nach 60 Epochen und es werden wieder nur die Infrastrukturhindernisse in der Vorhersage berücksichtigt.

Die angepasste Architektur hat sich als nicht sinnvoll erwiesen. Hier hat lediglich die Version mit Adam und Mean Squared Error (MSE) valide Ergebnisse generiert, was mit der Accuracy-Kurve übereinstimmt, da diese (Abbildung A.10 in blau) zwar starke Schwankungen aufweist, aber insgesamt am besten abschneidet. Auch die großen und kleinen Konfigurationen konnten sich aufgrund von Problemen mit den explodierenden Gradienten nicht durchsetzen.

4.4.3 Auswertung

Basierend auf den in Abschnitt 4.4.2 präsentierten Ergebnissen lassen sich einige Tendenzen feststellen. Besonders auffällig ist der Unterschied zwischen Adam und Adadelta. Die Ergebnisse, die mit Adam erzielt wurden, erscheinen weichgezeichneter im Vergleich zu jenen mit Adadelta, wo das Rauschen und somit der Zusammenhang mit den zufällig generierten Basis-Karten stärker zum Ausdruck kommt. Diese Beobachtung unterstreicht die Robustheit von Adam im Umgang mit Rauschen, wie auch in anderen Zusammenhängen bereits ausgewertet wurde. [54, 22] Dieser Punkt wird auch besonders in Kombination mit Batch Normalization deutlich, wo die Bilder mit Adadelta sehr stark verrauscht sind. Allgemein hat Batch Normalisation oftmals zu besseren Ergebnissen geführt.

Des Weiteren sind auch teilweise starke Unterschiede zwischen Mean Absolute Error (MAE) und Mean Squared Error (MSE) zu erkennen. Hierbei lässt sich die Tendenz erkennen, dass die Verwendung von MAE zu einem größeren Wertebereich führt bzw. mehr Werte in Richtung Minimum oder Maximum tendieren als bei MSE. Dies könnte damit korrelieren, dass Ausreißer bei MSE stärker bestraft werden als bei MAE. Dies führt in diesem Fall zu einem größeren Wertebereich, in dem die Werte deutlich stärker in Richtung des Maximums und Minimums tendieren.

Dropout und Dilation hingegen haben zumeist nicht zu einer Verbesserung der Ergebnissen geführt. Das bestätigen sowohl die stark schwankenden Accuracy Kurven als auch die visuelle Analyse. Bei Dropout kann dies ein Indiz sein, dass das Netz zu klein ist. Durch das Deaktivieren von Teilen des Netzes bei Dropout wird das Lernen erschwert. Daher ist es möglich, dass größere Netzwerke für gute Ergebnisse benötigt werden. [52] Bei Dilation besteht die Möglichkeit, dass eine Konfiguration mit weniger Dilation besser gewesen wäre. Denn die wenigen Details und das verwaschene Aussehen der Ergebnisse könnte ein Hinweis darauf sein, dass das Receptive Field zu groß ist und dadurch eine zu globale Sicht entsteht, weswegen die Details verloren gehen. In anderen Bereichen der Bildverarbeitung wurden hingegen bessere Ergebnisse mit Dilation erzielt als ohne. [48, 41]

5 Diskussion

Basierend auf den Ergebnissen aus dem letzten Kapitel lässt sich festhalten, dass, wie zu erwarten war, selbst geringfügige Veränderungen der Hyperparameter zu teilweise unerwarteten und signifikanten Änderungen in den Ergebnissen führen können. Dennoch lassen sich Tendenzen zu geeigneteren und weniger geeigneteren Modellen feststellen. Obwohl die gewählten Netzwerkarchitektur des CNN-Autoencoders in den hier durchgeführten Experimenten insgesamt nicht herausragend gute Ergebnisse erzielt, handelt es sich oftmals um valide Ergebnisse, die korrekte Zusammenhänge mit den Input Featuren abbilden. Daher lässt sich diese Architektur für diesen Anwendungsfall weiter als geeignet bezeichnen.

Des Weiteren gibt es einige Tendenzen bei den Hyperparametern und Feinheiten der Netzwerkarchitektur, die sich als vorteilhaft erwiesen haben. Demnach hängt der Erfolg jeder Netzwerkanpassung auch sehr stark von anderen Einflussfaktoren, wie den Eingangsdaten oder allgemein dem Trainingsmaterial ab, wodurch es keine allgemeingültige ideale Konfiguration gibt. Da das Training der Karten stets eine Erkundung durch die Drohnen erfordert, was wiederum bedeutet, dass jede weitere generierte Bewertungskarte potenzielle Kosten verursacht, sollte der Datensatz so klein wie nötig gehalten werden. Daher stellt Overfitting ein großes Problem dar. Aufgrund dessen ist insbesondere die Anwendung von Data Augmentation von essentieller Bedeutung. Dies ermöglicht es, die Menge der Input Daten künstlich mit geringem Aufwand zu erhöhen, was zu einer signifikanten Verbesserung im Training führen kann, wie die Experimente gezeigt haben. Neben Spiegelungen könnten auch Verzerrungen oder Rauschen getestet werden. Die Ergebnisse zeigen insgesamt, dass insbesondere Anpassungen im Netzwerk, die zu einer besseren Generalisierung führen, hilfreich sind und eine reine Netzwerkstruktur nur aus CNN-Layern nicht empfehlenswert ist. Hierbei hat sich insbesondere Batch Normalization als oftmals vorteilhaft erwiesen. Zudem hat auch Dropout in manchen Kombination eine sinnvolle Ergänzung darstellt.

Abhängig von den initialen Startkarten und den Inhalten sollte auch beachtet werden, welcher Loss und welcher Optimierer verwendet wird. Insbesondere bei initialen Karten, die bereits einen Bias enthalten, könnte die Verwendung von MAE von Vorteil sein, da dies eher zu einer Konvergenz in Richtung des angestrebten Wertebereichs führt. Außerdem hat sich Adadelta in manchen Kombinationen als vorteilhaft erwiesen, da zum einen keine manuelle Anpassung der Lernrate notwendig ist und es zum anderen ein sehr robuster Algorithmus ist.

Der starke Bias in den Versuchen mit initial generierten und leeren Karten könnte darauf hinweisen, dass für diesen Anwendungsfall eine individuellere Verlustfunktion besser geeignet wäre. Diese sollte nicht die gesamten Karten vergleichen, sondern nur die erforschten Bereiche in den Fehler einbeziehen. Dadurch könnte das Modell die Qualität, die durch die Belohnungen vorgegeben ist, in ihrer vollen Gänze erreichen, ohne dass der Einfluss der unerforschten Bereiche vorhanden wäre. Entsprechend würde vielleicht das Pixel-weise-lernen, wie von R. Furuta et al. (2020) in [21] beschrieben, besser funktionieren. Zusätzlich könnte dann, wie es klassischerweise der Fall ist, der Anteil der Exploration in den Simulationen mit jeder Iteration reduziert werden, ohne dass der Rest der Karte einen zu starken Einfluss hätte.

Neben der richtigen Netzwerkkonfiguration ist jedoch auch eine gute Belohnungsfunktion und die Konfiguration des Erforschens von entscheidender Bedeutung für den Erfolg. Insbesondere muss genau evaluiert werden, wie viel der Karte erkundet und wie genau dies umgesetzt wird. Im Rahmen dieser Arbeit wurde aus Mangel an Alternativen dieselben Quellen sowohl für die Bewertung als auch für die Input-Daten verwendet. Dieses Vorgehen ist nicht optimal. Zudem stellen diese einfachen statischen Daten nicht die ideale Quelle für sinnvolle Bewertungen dar. Es wäre besonders vorteilhaft neben realen Umweltdaten auch reale Flugdaten zu verwenden, die live oder aufgezeichnet sein könnten. Hierbei könnte es sich beispielsweise um Daten handeln, die zeigen, wie Profis die Drohnen manuell durch verschiedene Gelände fliegen würden. Dadurch könnte nicht nur basierend auf statischen Daten entschieden werden, welche Wege gut und welche schlecht sind, sondern es könnte auch von realen Daten abstrahiert werden, wann es beispielsweise angemessen ist, eine Steigung zu überfliegen und wann dies ein problematischer Weg wäre. Zudem können nicht nur die allgemeinen Routenwahlen berücksichtigt werden, sondern auch die idealen Flughöhen und Geschwindigkeiten einbezogen werden. Eine weitere vorteilhafte Ergänzung für die Bewertungskarten könnten aktuelle Wetter- und Winddaten zu den relevanten Bereichen darstellen. Es ist vorstellbar, dass beispielsweise in dicht besiedelten oder hügeligen Gebieten Windschneisen entstehen, die das Fliegen

der Drohnen erschweren. Wenn die aktuellen Verhältnisse einen Einfluss auf die Bewertung haben, kann das Modell beispielsweise statische Wetterkarten als Input erhalten und möglicherweise lernen, die Zusammenhänge zwischen Wetter und Infrastruktur/DEM zu erkennen und in die Potentialfelder mit einzubeziehen.

Besonders die Aspekte, welche die Simulation betreffen, sind für zukünftige Arbeiten relevant, da diese im Rahmen dieser Arbeit nur wenig betrachtet wurden. Insbesondere im Hinblick auf die Trainingsergebnisse wäre die Entwicklung und Verfeinerung einer besseren Bewertungsfunktion sinnhaft. Diese hat einen entscheidenden Einfluss auf die Qualität des Modells, da die generierten Karten nicht besser werden können als durch die Bewertungsfunktion vorgegeben. Im Rahmen dieser Arbeit wurde, lediglich eine einfache Bewertungsfunktion zu beginn entwickelt und genutzt. Daher wäre es ratsam diese zu verbessern und unterschiedliche Konfigurationen zu testen.

Zudem wurden relevante Faktoren in den Simulationen vernachlässigt, die wiederum Einfluss auf die Bewertungen hätten. Dazu gehören auch die physikalischen Aspekte der Drohnen. So sollten die Simulation im dreidimensionalen Raum durchgeführt werden, da dies schließlich der entscheidende Vorteil von Drohnen gegenüber Fahrzeugen ist. Außerdem sind auch Faktoren wie Beschleunigung und Energieverbrauch relevant, welches zu einer realistischeren Abbildung des Drohnenverhaltens führen würden. Dazu gehört auch die Einführung von variablen Geschwindigkeiten in der Simulation, welche so vernachlässigt wurden. Dadurch würde es zu mehr Dynamik in den Simulationen kommen und es würde neue relevante Faktoren für die Bewertungsfunktion schaffen. Des Weiteren müsste dann auch nochmal ein besserer Algorithmus für die Wegefindung genutzt werden, damit Kollisionen vermieden werden.

6 Fazit und Ausblick

Abschließend lässt sich argumentieren, dass die hier aufgezeigte Kombination aus dem Trainieren eines Potentialfeldes mittels RL und einem digitalen Zwilling ein vielversprechender Weg ist, um eine Methode zu entwickeln, mit der Drohnen ohne zusätzliche Sensorik für die Erfassung der Umwelt autonom fliegen können. Dafür spricht, dass bereits in den vorgestellten Ergebnissen deutlich wurde, dass das Modell in den Bewertungskarten Bereiche als potenziell gefährlich identifizieren konnte, was dazu führen kann, dass die spätere Routenplanung der Drohne diese Bereiche umfliegt. Es gibt zudem noch viel Potential für weitere Verbesserungen und es ist anzunehmen, dass komplexere Eingangsdaten in Kombination mit einer ausgefeilten Belohnungsstrategie mit realen Daten zu einem robusten Modell führen, das präzise Potentialfelder generiert. In diesem Zusammenhang besteht auch die Möglichkeit, dass, wenn sinnvolle reale Daten aufgezeichnet wurden, die Exploration weiterhin lediglich in einer simulierten Umgebung erfolgt. Ein weiterer Vorteil liegt darin, dass die gewählte Architektur mit dem digitalen Zwilling sehr adaptiv ist, sodass sowohl reale Sensordaten als Eingangsinformationen für die Simulationen dienen können als auch eine echte Drohne damit verbunden werden kann.

Zusätzlich zum Erweitern der Input Daten existiert auch die Möglichkeit den Output des Modells um zusätzliche Informationen zu ergänzen, die über ein einfaches Potentialfeld hinausgehen. Zum Beispiel könnte ein weiterer Kanal hinzugefügt werden, der optimale Flughöhen beinhaltet. Dadurch könnte das Modell nicht nur lernen Hindernisse in einem zweidimensionalen Raum vorherzusagen und sichere Routen zu gewährleisten, sondern es könnte auch eine Ebene generieren, die sicherstellt, dass die gewählte Flughöhe möglichst effizient ist.

Zukünftig könnte neben den Weiterentwicklungen bei der Generierung der Potentialfelder auch im Hinblick auf die Pfadplanung, basierend auf den Potentialfeldern, geforscht werden. Es bietet sich an nicht nur einfache Pfadplanungsalgorithmen für die Navigation zu nutzen, sondern ein Modell zu trainieren, das den Weg basierend auf der Karte

wählt. Dies ist insbesondere in Kombination mit der kooperativen Aufgabe der Signallokalisierung, die die Drohnen durchführen sollen, ein spannender Bereich. Zudem zeigen Arbeiten wie [61] bereits Erfolg mit einem solchen Ansatz.

Literaturverzeichnis

[1] FOSSGIS e.V.: *OpenStreetMap*. – URL https://www.openstreetmap.de/. – Zugriffsdatum: 07.12.2023

[2] Abdelkader, Mohamed ; Guler, Samet ; Jaleel, Hassan ; Shamma, Jeff: Aerial Swarms: Recent Applications and Challenges. In: *Current Robotics Reports* 2 (2021), 09, S. 1–12

[3] Albani, Dario ; Manoni, Tiziano ; Arik, Arikhan ; Nardi, Daniele ; Trianni, Vito: Field Coverage for Weed Mapping: Toward Experiments with a UAV Swarm. In: *Bio-inspired Information and Communication Technologies*, Springer International Publishing, 2019. – ISBN 978-3-030-24202-2

[4] Azarang, Arian ; Manoochehri, Hafez E. ; Kehtarnavaz, Nasser: Convolutional Autoencoder-Based Multispectral Image Fusion. In: *IEEE Access* 7 (2019), S. 35673–35683

[5] B-Open Solutions: *Elevation*. 2021. – URL https://elevation.bopen.eu/en/stable/. – Zugriffsdatum: 17.10.2023

[6] Baker, Chris A. B. ; Ramchurn, Sarvapali ; Teacy, W.T. L. ; Jennings, Nicholas R.: Planning Search and Rescue Missions for UAV Teams. In: *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, IOS Press, 2016 (ECAI'16), S. 1777–1778. – URL https://doi.org/10.3233/978-1-61499-672-9-1777. – ISBN 9781614996712

[7] Bäker, Martin: *Geodäten*. S. 79–106. In: *Isaac oder Die Entdeckung der Raumzeit*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2019. – ISBN 978-3-662-57293-1

[8] Baldazo, David ; Parras, Juan ; Zazo, Santiago: Decentralized Multi-Agent Deep Reinforcement Learning in Swarms of Drones for Flood Monitoring. In: *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, S. 1–5

[9] Bank, Dor ; Koenigstein, Noam ; Giryes, Raja: *Autoencoders*. S. 353–374. In: *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*. Cham : Springer International Publishing, 2023. – URL https://doi.org/10.1007/978-3-031-24628-9_16. – Zugriffsdatum: 01.12.2023. – ISBN 978-3-031-24628-9

[10] Barnes, Richard: *RichDEM: Terrain Analysis Software*. 2016. – URL http://github.com/r-barnes/richdem. – Zugriffsdatum: 17.10.2023

[11] Batty, Michael: Digital twins. In: *Environment and Planning B: Urban Analytics and City Science* 45 (2018), 09, S. 817–820

[12] Bhalodia, Riddhish ; Elhabian, Shireen Y. ; Kavan, Ladislav ; Whitaker, Ross T.: A Cooperative Autoencoder for Population-Based Regularization of CNN Image Registration. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Cham : Springer International Publishing, 2019, S. 391–400. – ISBN 978-3-030-32245-8

[13] Boeing, Geoff: OSMNX: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. In: *Computers Environment and Urban Systems* 65 (2017). – URL https://osmnx.readthedocs.io/en/stable/. – Zugriffsdatum: 30.10.2023

[14] Borole, Yogini ; Borkar, Pradnya ; Raut, Roshani ; Balpande, Vijaya P. ; Chatterjee, Prasenjit: *Digital Twins*. De Gruyter, 2023. – URL https://doi.org/10.1515/9783110778861. – ISBN 9783110778861

[15] Chen, Min ; Shi, Xiaobo ; Zhang, Yin ; Wu, Di ; Guizani, Mohsen: Deep Feature Learning for Medical Image Analysis with Convolutional Autoencoder Neural Network. In: *IEEE Transactions on Big Data* 7 (2021), Nr. 4, S. 750–758

[16] Cheng-bo, WANG ; Xin-yu, ZHANG ; Jia-wei, ZHANG ; Zhi-guo, DING ; Lan-xuan, AN: Navigation behavioural decision-making of MASS based on deep reinforcement learning and artificial potential field. In: *Journal of Physics: Conference Series* 1357 (2019), Nr. 1, S. 012026. – URL https://dx.doi.org/10.1088/1742-6596/1357/1/012026

[17] Clemen, Thomas ; Ahmady-Moghaddam, Nima ; Lenfers, Ulfia A. ; Ocker, Florian ; Osterholz, Daniel ; Ströbele, Jonathan ; Glake, Daniel: Multi-Agent Systems and Digital Twins for Smarter Cities. In: *Proceedings of the 2021 ACM*

SIGSIM Conference on Principles of Advanced Discrete Simulation, Association for Computing Machinery, 2021, S. 45–55. – URL https://doi.org/10.1145/3437959.3459254. – ISBN 9781450382960

[18] Fadil, Rabie ; Abou El Majd, Badr ; ghazi, Hassan el ; Hicham, Rahil: Optimizing the Multi-Objective Deployment Problem of Mlat System, 01 2018, S. 00014

[19] Farr, Tom G. ; Rosen, Paul A. ; Caro, Edward ; Crippen, Robert ; Duren, Riley ; Hensley, Scott ; Kobrick, Michael ; Paller, Mimi ; Rodriguez, Ernesto ; Roth, Ladislav ; Seal, David ; Shaffer, Scott ; Shimada, Joanne ; Umland, Jeffrey ; Werner, Marian ; Oskin, Michael ; Burbank, Douglas ; Alsdorf, Douglas: The Shuttle Radar Topography Mission. In: *Reviews of Geophysics* 45 (2007), Nr. 2. – URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2005RG000183. – Zugriffsdatum: 01.12.2023

[20] Frochte, J.: *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. Hanser, 2021 (Plus.Hanser-Fachbuch). – URL https://books.google.de/books?id=J_J1zwEACAAJ. – ISBN 9783446461444

[21] Furuta, Ryosuke ; Inoue, Naoto ; Yamasaki, Toshihiko: PixelRL: Fully Convolutional Network With Reinforcement Learning for Image Processing. In: *IEEE Transactions on Multimedia* 22 (2020), Nr. 7, S. 1704–1719

[22] Geng, Alexander ; Moghiseh, Ali ; Redenbach, Claudia ; Schladitz, Katja: Comparing optimization methods for deep learning in image processing applications. In: *tm - Technisches Messen* 88 (2021), Nr. 7-8, S. 443–453. – URL https://doi.org/10.1515/teme-2021-0023

[23] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – URL http://www.deeplearningbook.org

[24] Gron, Aurlien: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. – ISBN 1491962291

[25] Guo, Siyu ; Zhang, Xiuguo ; Zheng, Yisong ; Du, Yiquan: An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning. In: *Sensors* 20 (2020), Nr. 2. – URL https://www.mdpi.com/1424-8220/20/2/426. – ISSN 1424-8220

[26] Gustafsson, F. ; Gunnarsson, F.: Positioning using time-difference of arrival measurements. 6 (2003), S. VI–553

[27] He, Shuai ; Dong, Xiaodai: High-Accuracy Localization Platform Using Asynchronous Time Difference of Arrival Technology. In: *IEEE Transactions on Instrumentation and Measurement* 66 (2017), Nr. 7, S. 1728–1742

[28] Hinton, Nair: Rectified Linear Units Improve Restricted Boltzmann Machines, 2010. (2010)

[29] Horn, B.K.P.: Hill shading and the reflectance map. In: *Proceedings of the IEEE* 69 (1981), Nr. 1, S. 14–47

[30] Hwang, Yong K. ; Ahuja, Narendra u. a.: A potential field approach to path planning. In: *IEEE transactions on robotics and automation* 8 (1992), Nr. 1, S. 23–32

[31] Hüning, Christian ; Adebahr, Mitja ; Clemen, Thomas ; Dalski, Jan ; Clemen, Ulfia ; Grundmann, Lukas ; Dybulla, Janus ; Kiker, G.: Modeling Simulation as a Service with the Massive Multi-Agent System MARS, 04 2016

[32] Jiang, Hao ; Li, Shengze ; Zhang, Jieyuan ; Zhu, Yuqi ; Xu, Xinhai ; Liu, Donghong: Efficient state representation with artificial potential fields for reinforcement learning. In: *Complex Intelligent Systems* 9 (2023). – URL https://doi.org/10.1007/s40747-023-00995-8

[33] Julian, Kyle D. ; Kochenderfer, Mykel J.: *Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning*. 2019. – URL https://doi.org/10.2514/1.G004106

[34] Karimpouli, Sadegh ; Tahmasebi, Pejman: Segmentation of digital rock images using deep convolutional autoencoder networks. In: *Computers Geosciences* 126 (2019), S. 142–150. – URL https://www.sciencedirect.com/science/article/pii/S0098300418303911

[35] Kaune, Regina: Accuracy studies for TDOA and TOA localization. In: *2012 15th International Conference on Information Fusion*, 2012, S. 408–415

[36] Kieu, Tung ; Yang, Bin ; Jensen, Christian: Outlier Detection for Multidimensional Time Series Using Deep Neural Networks, 06 2018, S. 125–134

[37] Kong, Fuchen ; Wang, Qi ; Gao, Shang ; Yu, Hualong: B-APFDQN: A UAV Path Planning Algorithm Based on Deep Q-Network and Artificial Potential Field. In: *IEEE Access* 11 (2023), S. 44051–44064

[38] Lange, Sascha ; Gabel, Thomas ; Riedmiller, Martin: *Batch Reinforcement Learning*. S. 45–73. In: Wiering, Marco (Hrsg.) ; Otterlo, Martijn van (Hrsg.): *Reinforcement Learning: State-of-the-Art*, Springer Berlin Heidelberg, 2012. – URL https://doi.org/10.1007/978-3-642-27645-3_2. – Zugriffsdatum: 01.12.2023. – ISBN 978-3-642-27645-3

[39] Lenfers, Ulfia A. ; Weyl, Julius ; Clemen, Thomas: Firewood Collection in South Africa: Adaptive Behavior in Social-Ecological Models. In: *Land* 7 (2018), Nr. 3. – URL https://www.mdpi.com/2073-445X/7/3/97

[40] Li, Bohao ; Wu, Yunjie: Path Planning for UAV Ground Target Tracking via Deep Reinforcement Learning. In: *IEEE Access* 8 (2020), S. 29064–29074

[41] Li, Xiang ; Zhai, Mengyao ; Sun, Junding: DDCNNC: Dilated and depthwise separable convolutional neural Network for diagnosis COVID-19 via chest X-ray images. In: *International Journal of Cognitive Computing in Engineering* 2 (2021), S. 71–82. – URL https://www.sciencedirect.com/science/article/pii/S2666307421000097

[42] Maicas, Gabriel ; Bradley, Andrew P. ; Nascimento, Jacinto C. ; Reid, Ian ; Carneiro, Gustavo: Pre and post-hoc diagnosis and interpretation of malignancy from breast DCE-MRI. In: *Medical Image Analysis* 58 (2019), S. 101562. – URL https://www.sciencedirect.com/science/article/pii/S1361841518306893. – ISSN 1361-8415

[43] MARS Group: *MARS*. – URL https://www.mars-group.org/. – Zugriffsdatum: 25.10.2023

[44] Murphy, Sean O. ; Sreenan, Cormac ; Brown, Kenneth N.: Autonomous Unmanned Aerial Vehicle for Search and Rescue Using Software Defined Radio. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019

[45] Nyambo, Devotha G. ; Luhanga, Edith T. ; Yonah, Zaipuna O. ; Mujibi, Fidalis D. ; Clemen, Thomas: Leveraging peer-to-peer farmer learning to facilitate better strategies in smallholder dairy husbandry. In: *Adaptive Behavior* 30 (2022), Nr. 1, S. 51–62. – URL https://doi.org/10.1177/1059712320971369

[46] O'Keefe, Brian: Finding location with time of arrival and time difference of arrival techniques. In: *ECE Senior Capstone Project* (2017)

[47] Patterson, Josh ; Gibson, Adam: *Deep Learning A Practitioners Approach*. O'Reilly Media, Inc., 2017. – ISBN 978149194250

[48] Perry, Jonathan ; Fernandez, Amanda: MinENet: A Dilated CNN for Semantic Segmentation of Eye Features. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019

[49] Raj, Ahnirudh Y. ; Venkatraman, Akshaya ; Vinodh, Anish ; Kumar, Hariharank: Autonomous Drone for Smart Monitoring of an Agricultural Field. In: *2021 7th International Engineering Conference "Research Innovation amid Global Pandemic"(IEC)*, 2021

[50] Reynolds, Craig W.: *Flocks, Herds, and Schools: A Distributed Behavioral Model*. In: *Seminal Graphics: Pioneering Efforts That Shaped the Field*, Association for Computing Machinery, 1998. – URL https://doi.org/10.1145/280811.281008. – ISBN 158113052X

[51] Russell, Stuart J. ; Norvig, Peter: *Artificial intelligence a modern approach*. Pearson, 2010. – ISBN 9780136042594

[52] Russell, Stuart J. ; Norvig, Peter: *Artificial intelligence a modern approach*. Pearson, 2021. – ISBN 97801346109993

[53] Shuttle Radar Topography Mission: *The Shuttle Radar Topography Mission (SRTM) Collection User Guide*. Web. 2015. – URL https://lpdaac.usgs.gov/documents/179/SRTM_User_Guide_V3.pdf. – Zugriffsdatum: 17.10.2023

[54] Soydaner, Derya: A Comparison of Optimization Algorithms for Deep Learning. In: *International Journal of Pattern Recognition and Artificial Intelligence* 34 (2020), Nr. 13, S. 2052013. – URL https://doi.org/10.1142/S0218001420520138. – Zugriffsdatum: 01.12.2023

[55] Sun, Yimao ; Ho, K. C. ; Wan, Qun: Solution and Analysis of TDOA Localization of a Near or Distant Source in Closed Form. In: *IEEE Transactions on Signal Processing* 67 (2019), Nr. 2, S. 320–335

[56] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction, Second Edition*. The MIT Press, 2018. – ISBN 9780262039246

[57] Vargas-Munoz, John E. ; Srivastava, Shivangi ; Tuia, Devis ; Falcão, Alexandre X.: OpenStreetMap: Challenges and Opportunities in Machine Learning and Remote Sensing. In: *IEEE Geoscience and Remote Sensing Magazine* 9 (2021), Nr. 1

[58] Wang, Jiankun ; Chi, Wenzheng ; Li, Chenming ; Wang, Chaoqun ; Meng, Max Q.-H.: Neural RRT*: Learning-Based Optimal Path Planning. In: *IEEE Transactions on Automation Science and Engineering* 17 (2020), Nr. 4, S. 1748–1758

[59] Wang, Xinwei ; Liu, Jie ; Peng, Haijun ; Qie, Xiwang ; Zhao, Xudong ; Lu, Chen: A Simultaneous Planning and Control Method Integrating APF and MPC to Solve Autonomous Navigation for USVs in Unknown Environments. In: *Journal of Intelligent Robotic Systems* 105 (2022), Nr. 36. – URL https://doi.org/10.1007/s10846-022-01663-8

[60] Weyl, Julius ; Glake, Daniel ; Clemen, Thomas: Agent-Based Traffic Simulation at City Scale with MARS. In: *Proceedings of the Agent-Directed Simulation Symposium*, Society for Computer Simulation International, 2018 (ADS '18). – ISBN 9781510860131

[61] Yao, Qingfeng ; Zheng, Zeyu ; Qi, Liang ; Yuan, Haitao ; Guo, Xiwang ; Zhao, Ming ; Liu, Zhi ; Yang, Tianji: Path Planning Method With Improved Artificial Potential Field—A Reinforcement Learning Perspective. In: *IEEE Access* 8 (2020), S. 135513–135523

[62] Zhou, S. K. ; Le, Hoang N. ; Luu, Khoa ; V Nguyen, Hien ; Ayache, Nicholas: Deep reinforcement learning in medical imaging: A literature review. In: *Medical Image Analysis* 73 (2021), S. 102193. – URL https://www.sciencedirect.com/science/article/pii/S1361841521002395. – ISSN 1361-8415

[63] Zhou, Yongkun ; Rao, Bin ; Wang, Wei: UAV Swarm Intelligence: Recent Advances and Future Trends. In: *IEEE Access* 8 (2020), S. 183856–183878

A Anhang

A.1 Technischer Aufbau - Code

A.1.1 Berechnungen zu den Erstellungen der Datensätze

Codeblock A.1: Berechnung der Slope Karten basierend auf Höhendaten und Aspect

```
# Mapping from aspact to the corresponding array indices --> Aspact=0 means facing Nord
view_to_index = { 0: [0,1], 1: [0,2], 2: [1,2], 3: [2,2], 4: [2,1], 5: [2,0], 6: [1,0], 7:
    [0,0], 8: [0,1]}

slope = np.zeros_like(elevation_map)
for line in range(len(elevation_map)):
    for value in range(len(elevation_map[line])):
        #Create sliding window
        window = elevation_map[max(0, line-1):min((len(elevation_map)-1), line+2),
            max(0,value-1):min((len(elevation_map[line])-1), value+2)]

        #Get direction of slope
        aspect = aspect_map[line,value]
        facing = math.ceil((aspect-22.5)/45)
        index = view_to_index[facing]

        #Get height values aund calculate difference
        neighbor = (window[min(index[0],(window.shape[0]-1)),
            min(index[1],(window.shape[1]-1))])
        height_in_km = abs(elevation_map[line,value]-neighbor)/1000

        #Calculate latitude and longetude values for both fields
        own_pos_long = (value * scale_long) + long_min
        own_pos_lat = (line * scale_lat)+ lat_min
        target_pos_long = ((value + (index[1]-1)) * scale_long) + long_min
```

```
target_pos_lat = ((line + (index[0]-1)) * scale_lat) + lat_min

#Calculate distances in Km
factor_long = (111.3 * math.cos((own_pos_lat+target_pos_lat)/2))
dx = factor_long * abs(own_pos_long - target_pos_long)
dy = 111.13 * abs(own_pos_lat- target_pos_lat)

#Calculate slope
dist  = math.sqrt((dx**2+dy**2))
val  = np.arctan((height_in_km/dist))
slope[line,value]  = val*(180/math.pi)
```

A.1.2 Agentenlogik

Codeblock A.2: Allgemeine Struktur der entscheidungslogik des Agenten zum Orten von Signalen

```
public DroneStates Decide(SignalLayer signalLayer, AreaGrid areaGrid, ElevationLayer
    elevationLayer, Position dronePosition, Dictionary<string,SimpleDrone>
    allDronePositions){
    if we are in phase 1 {
        if we need a new target {
            target = FindNewTarget(allDronePositions, signalLayer);
            return DroneStates.SetNewTarget;
        }
        if the drone Position is not inside of area Grid {
            bearing = GetBearingToMoveBackInArea();
            return  DroneStates.MoveTowards;
        }
        if elevationLayer is used {
            foreach surrounding elevation{
                if elevation > maximum elevation to fly above {
                    bearing = getAvoidObstacleBearing();
                    return DroneStates.MoveTowards;
                }
            }
        }

        if we reached the target {
            changePhaseTo( 2 );
```

```
            return DroneStates.Wait;
        }

        bearing = getBearing(target)
        return DroneStates.MoveTowards();
    } else { //This is Phase 2
        if maximumWaitingTime == 0 {
            changePhaseTo( 1 );
            return DroneStates.Wait;
        } else {
            if numberOfDronesAtTarget >= 4 {
                if position is not correct {
                    bearing = calculatedBearingForFormation;
                    return DroneStates.MoveTowards();
                } else{
                    changePhaseTo( 1 );
                    return DroneStates.Locating;
                }

            } else{
                maximumWaitingTime -= (4 - numberOfDronesAtTarget);
                bearing = calculatedBearingForFormation;
                return DroneStates.MoveTowards();
            }
        }
    }
```

A.2 Ergänzende Bilder zu den Experimenten

A.2.1 Experiment 3 und 4

Im folgenden sind sowohl die Netzkonfiguration als auch die Ergebnisse des dritten und vierten Experiments abgebildet.

Im folgenden sind sowohl die Netzkonfiguration als auch die Ergebnisse des dritten Experiments abgebildet.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 256, 256, 16)      592

 batch_normalization_8 (Batc  (None, 256, 256, 16)     64
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 128, 128, 16)     0
 2D)

 conv2d_10 (Conv2D)          (None, 128, 128, 32)      4640

 batch_normalization_9 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_11 (Conv2D)          (None, 64, 64, 64)        18496

 batch_normalization_10 (Bat  (None, 64, 64, 64)       256
 chNormalization)

 max_pooling2d_5 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_12 (Conv2D)          (None, 32, 32, 64)        36928

 batch_normalization_11 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 conv2d_13 (Conv2D)          (None, 32, 32, 64)        36928

 batch_normalization_12 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 up_sampling2d_3 (UpSampling  (None, 64, 64, 64)       0
 2D)

 conv2d_14 (Conv2D)          (None, 64, 64, 64)        36928

 batch_normalization_13 (Bat  (None, 64, 64, 64)       256
 chNormalization)

 up_sampling2d_4 (UpSampling  (None, 128, 128, 64)     0
 2D)

 conv2d_15 (Conv2D)          (None, 128, 128, 32)      18464

 batch_normalization_14 (Bat  (None, 128, 128, 32)     128
 chNormalization)

 up_sampling2d_5 (UpSampling  (None, 256, 256, 32)     0
 2D)

 conv2d_16 (Conv2D)          (None, 256, 256, 16)      4624

 batch_normalization_15 (Bat  (None, 256, 256, 16)     64
 chNormalization)

 conv2d_17 (Conv2D)          (None, 256, 256, 1)       17

=================================================================
Total params: 159,025
Trainable params: 158,321
Non-trainable params: 704
_________________________________________________________________
Optimizer:  <keras.optimizers.optimizer_v2.adam.Adam object at 0x000001F43F386830>
Loss:  <function mean_squared_error at 0x000001F43CA413F0>
```

Abbildung A.1: Die Netzkonfiguration vom dritten Experiment, anders sind die kleineren Ebenen und Batch-Normalization

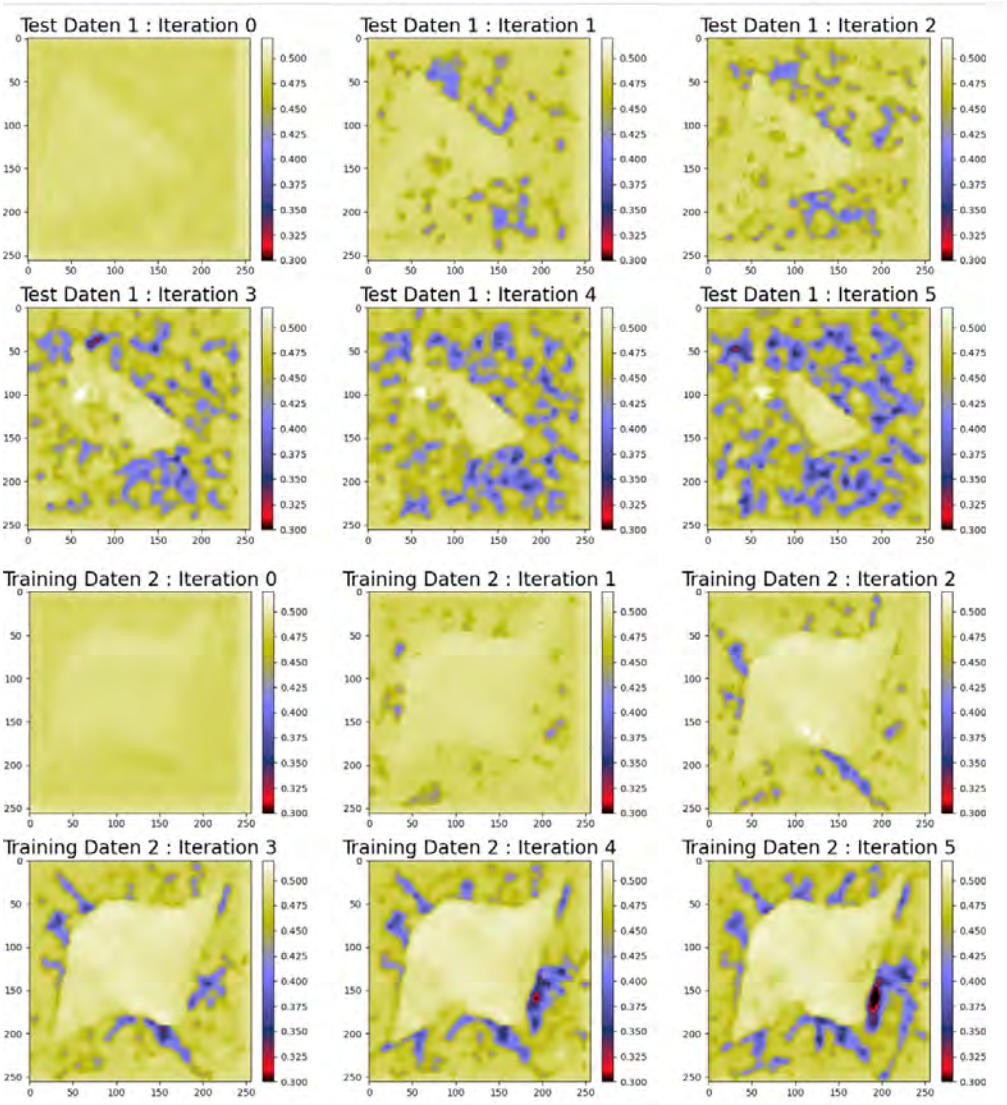

Abbildung A.2: Die generierten Potentialkarten über fünf Iterationen im dritten Experiment [Skala: 0,3 - 0,52]

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      592

 batch_normalization (BatchN  (None, 256, 256, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      4640

 batch_normalization_1 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

 batch_normalization_2 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 64)        36928

 batch_normalization_3 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 conv2d_4 (Conv2D)           (None, 32, 32, 64)        36928

 batch_normalization_4 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 up_sampling2d (UpSampling2D  (None, 64, 64, 64)       0
 )

 conv2d_5 (Conv2D)           (None, 64, 64, 64)        36928

 batch_normalization_5 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 up_sampling2d_1 (UpSampling  (None, 128, 128, 64)     0
 2D)

 conv2d_6 (Conv2D)           (None, 128, 128, 32)      18464

 batch_normalization_6 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 up_sampling2d_2 (UpSampling  (None, 256, 256, 32)     0
 2D)

 conv2d_7 (Conv2D)           (None, 256, 256, 16)      4624

 batch_normalization_7 (Batc  (None, 256, 256, 16)     64
 hNormalization)

 conv2d_8 (Conv2D)           (None, 256, 256, 1)       17

=================================================================
Total params: 159,025
Trainable params: 158,321
Non-trainable params: 704
_________________________________________________________________
Optimizer:  <keras.optimizers.optimizer_v2.adam.Adam object at 0x0000016B5488A350>
Loss:  <function mean_absolute_error at 0x0000016B51146B90>
```

Abbildung A.3: Netzkonfiguration die im vierten Experiment verwendet wurde

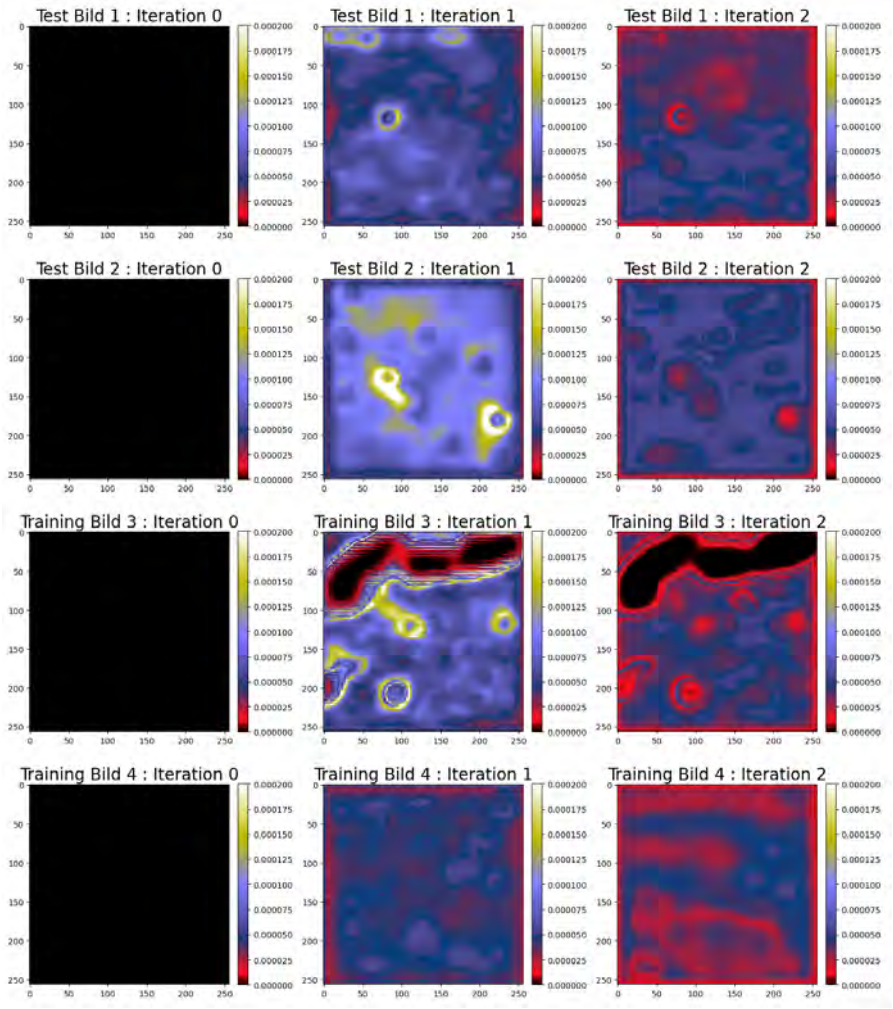

Abbildung A.4: Die generierten Potentialkarten über zwei Iterationen im vierten Experiment [Skala: 0 - 0,0002]

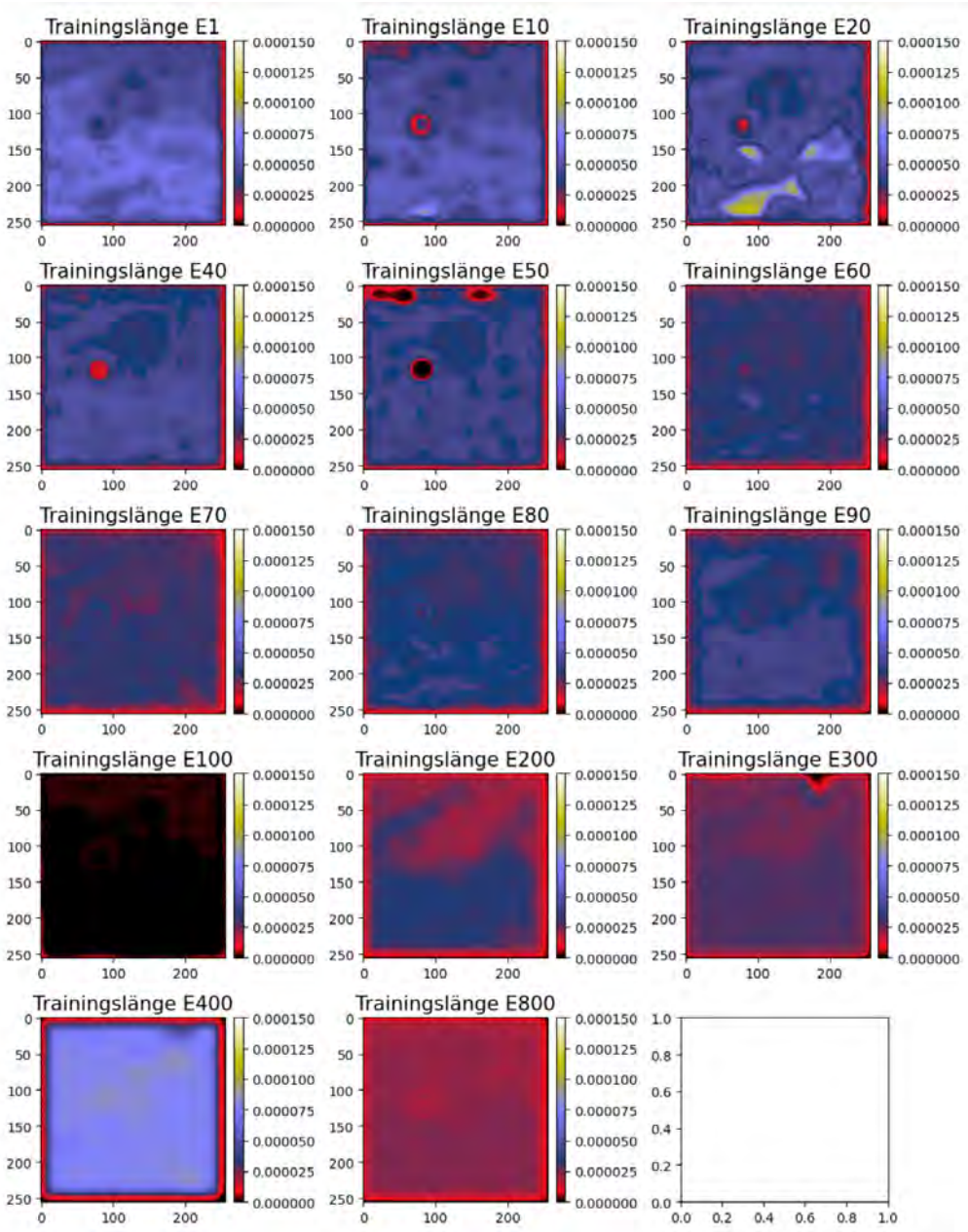

Abbildung A.5: Verschiedene Trainingslängen Test Datensatz 1 [Skala: 0 - 0,00015]

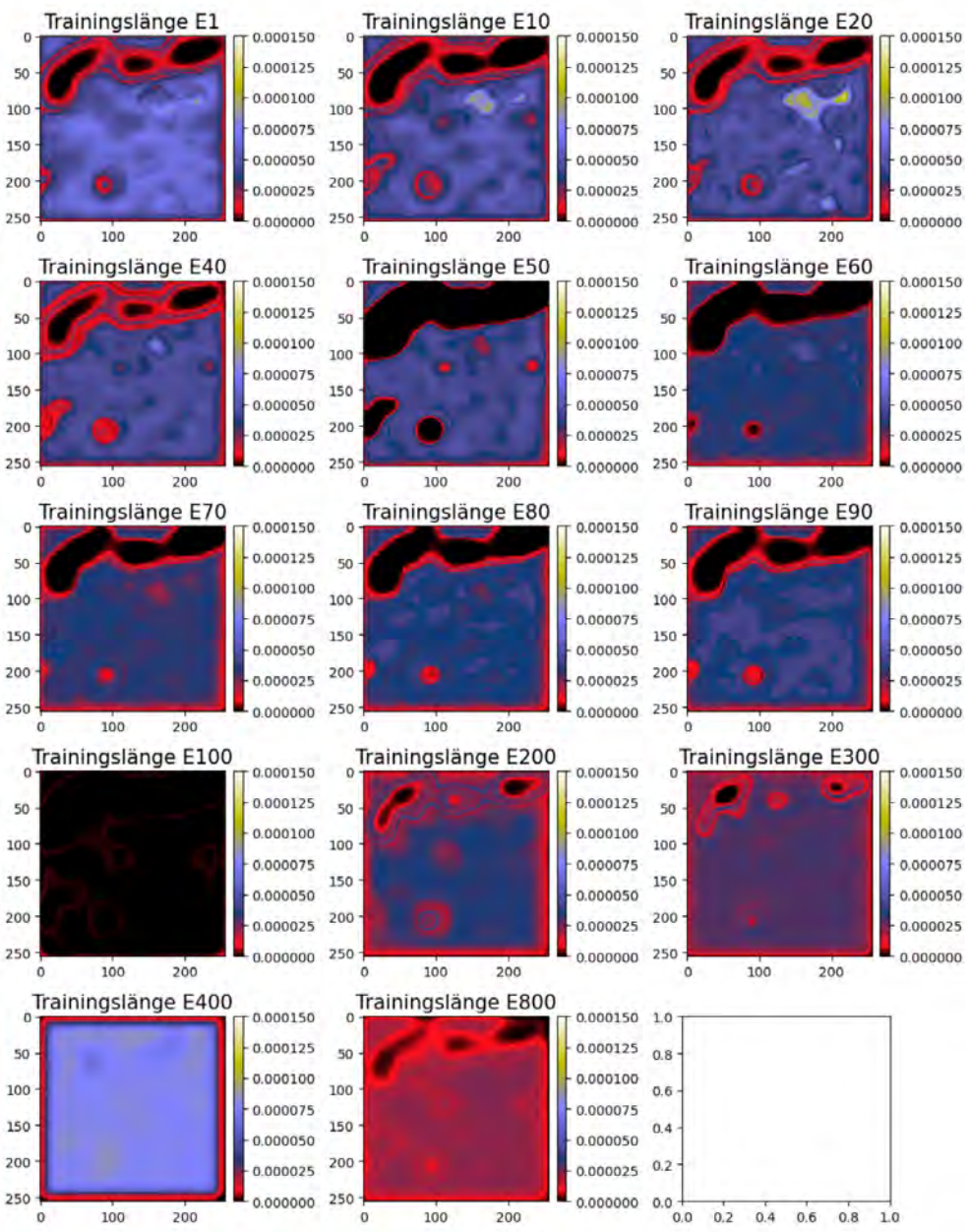

Abbildung A.6: Verschiedene Trainingslängen Traings Daten 3 [Skala: 0 - 0,00015]

A.2.2 Experiment 5 - alle Daten

Im folgenden sind die Accuracy und Loss Kurven und die Beispielbilder zu allen getesten Netzkonfiguration aus dem fünften Experiments abgebildet.

Accuracy und Loss

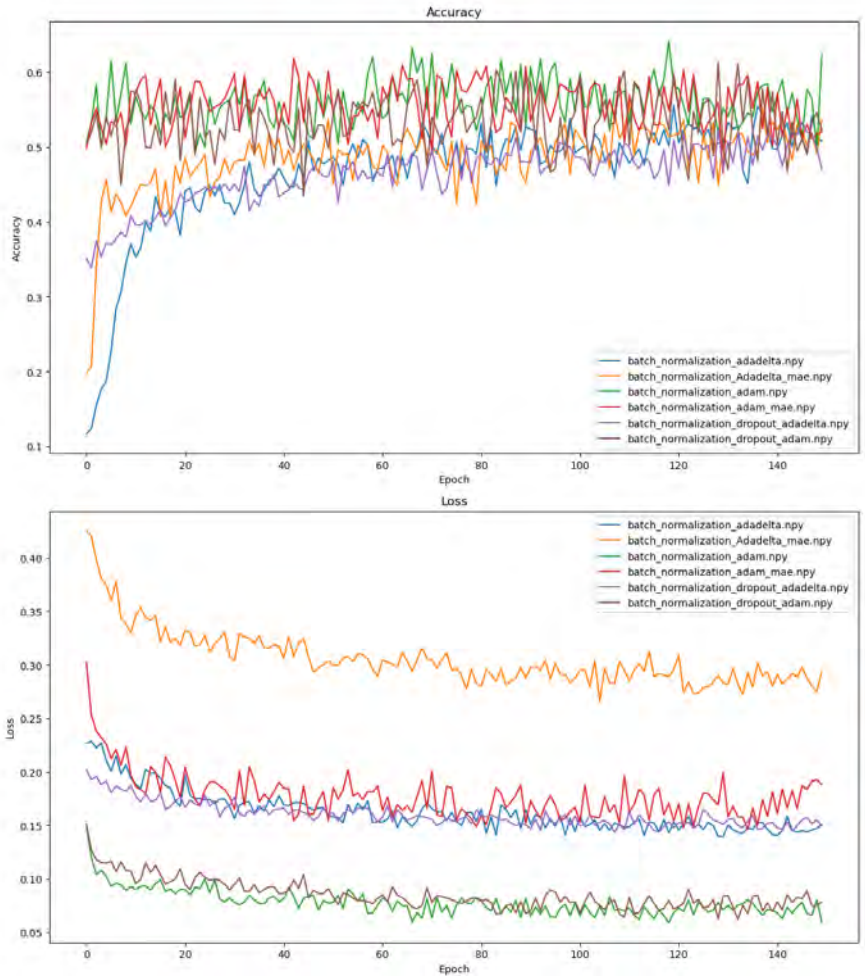

Abbildung A.7: Accuracys und Loss Kurven

Beispielbilder

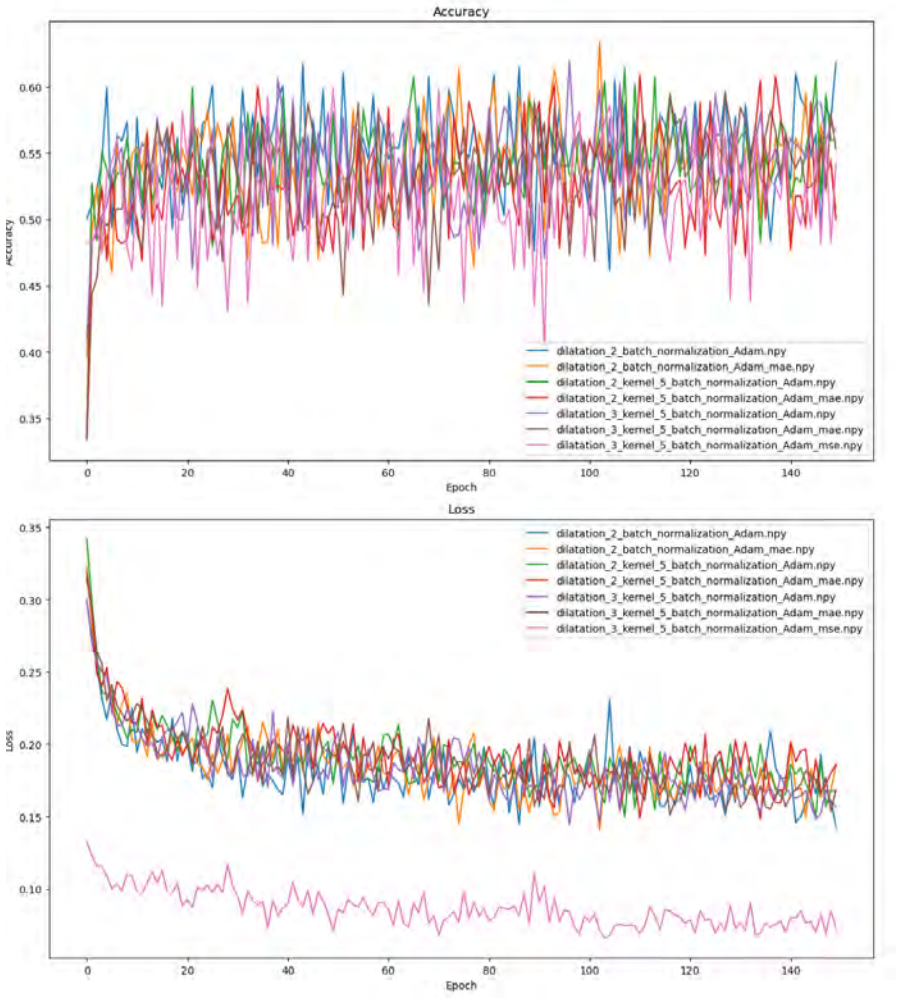

Abbildung A.8: Accuracys und Loss Kurven

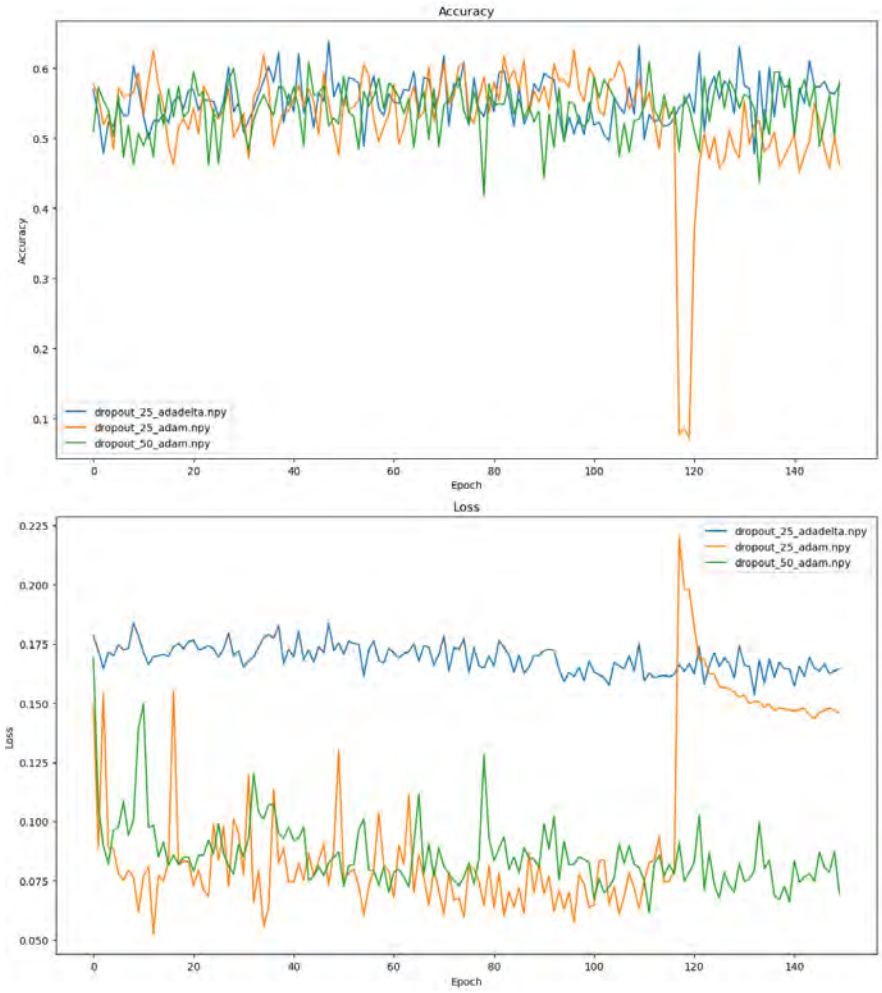

Abbildung A.9: Accuracys und Loss Kurven

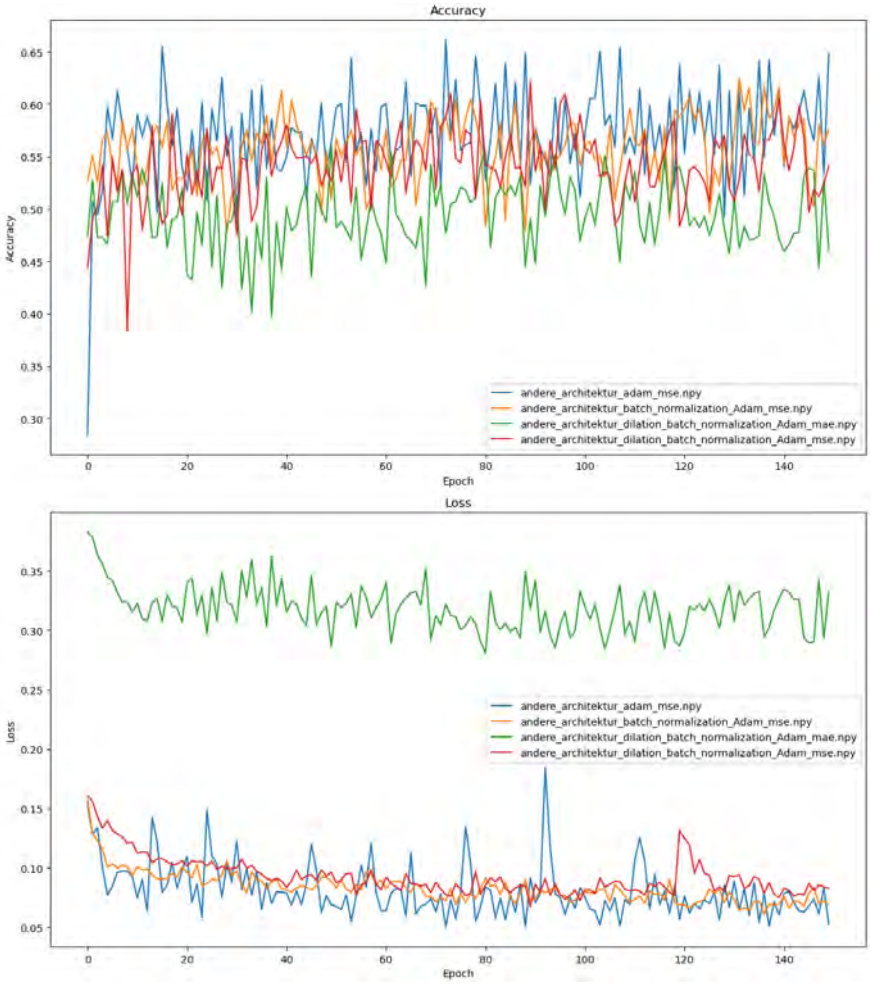

Abbildung A.10: Accuracys und Loss Kurven

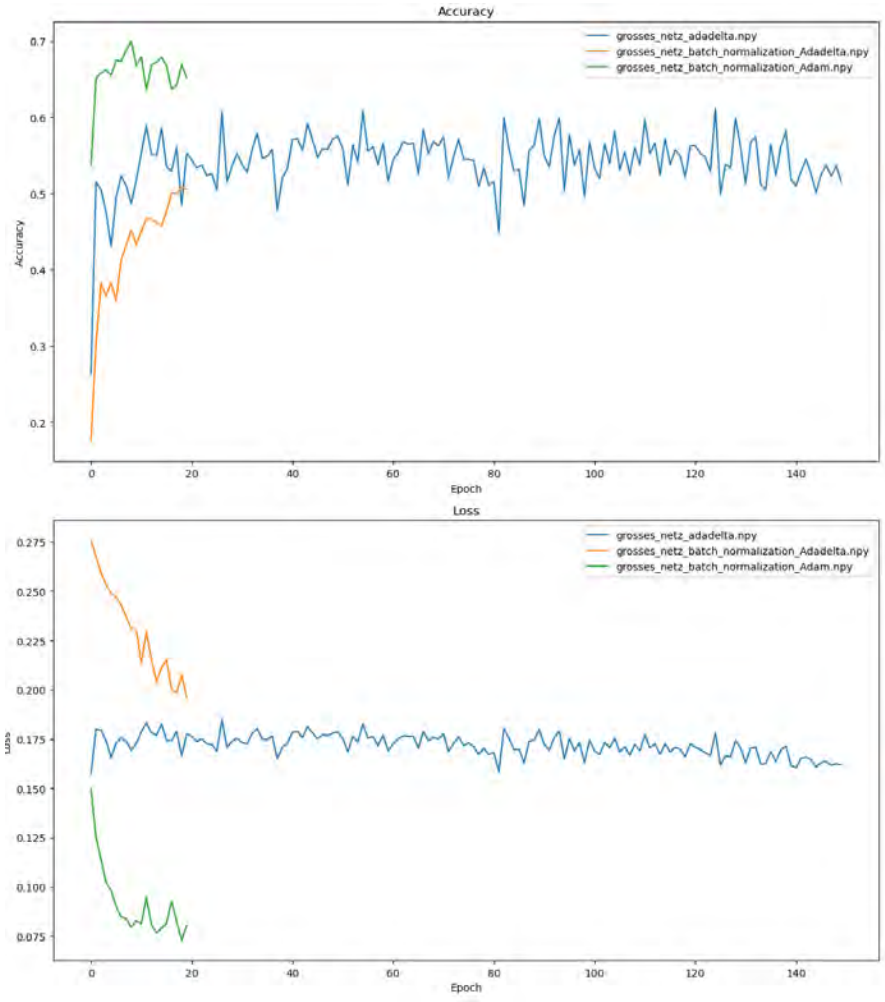

Abbildung A.11: Accuracys und Loss Kurven

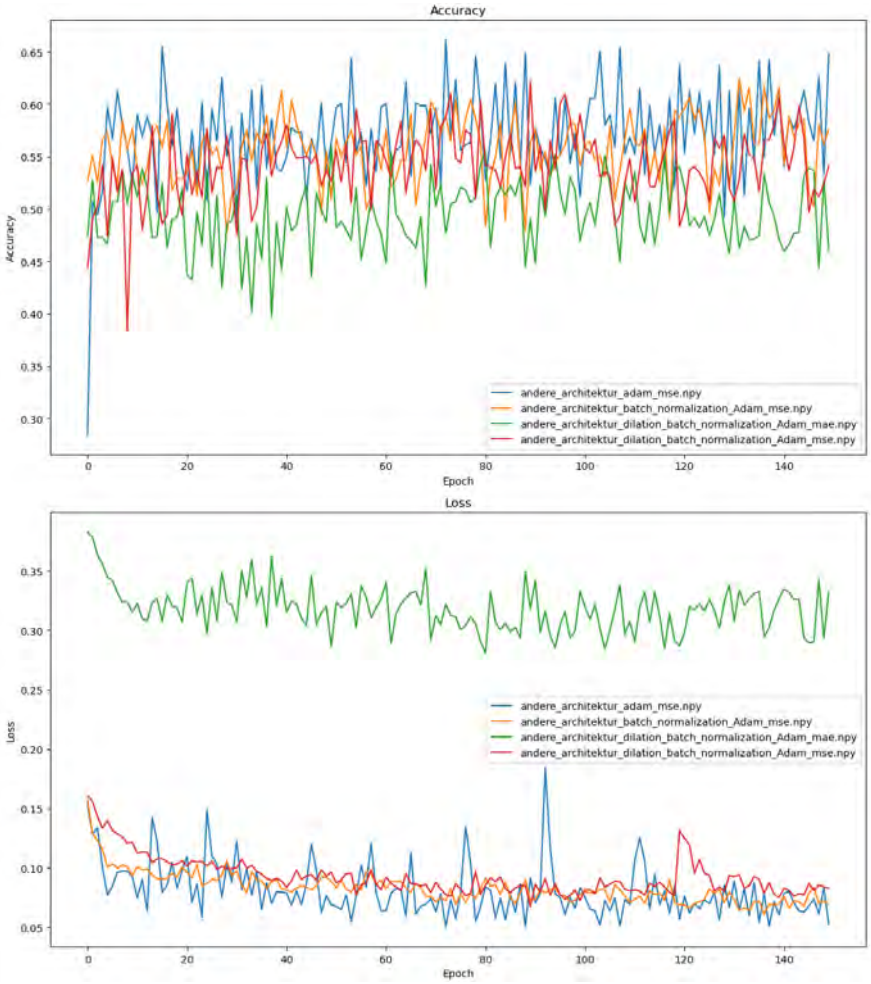

Abbildung A.12: Accuracys und Loss Kurven

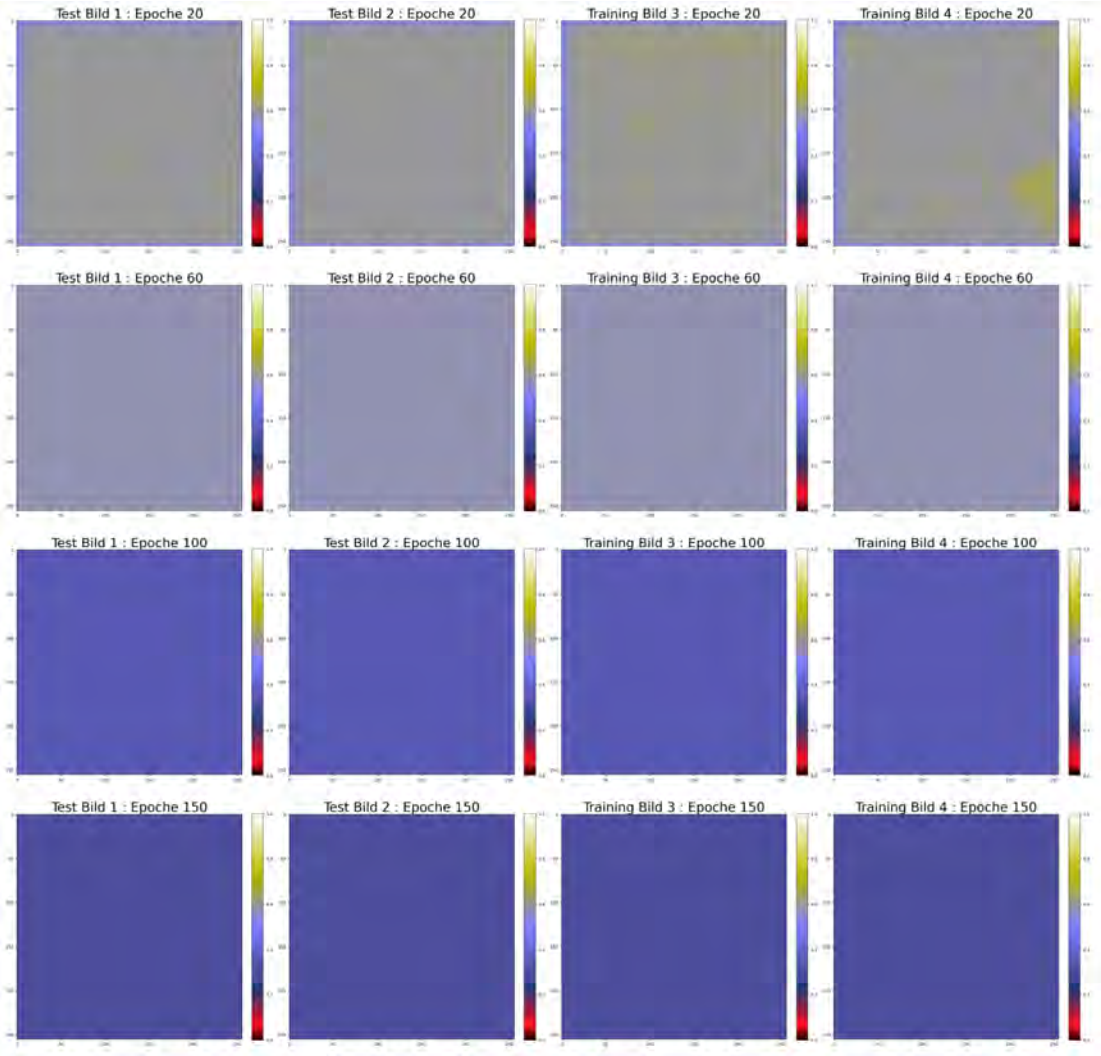

Abbildung A.13: Potentialkarten Netz: Adam und MSE [Skala: 0,3 - 0,63]

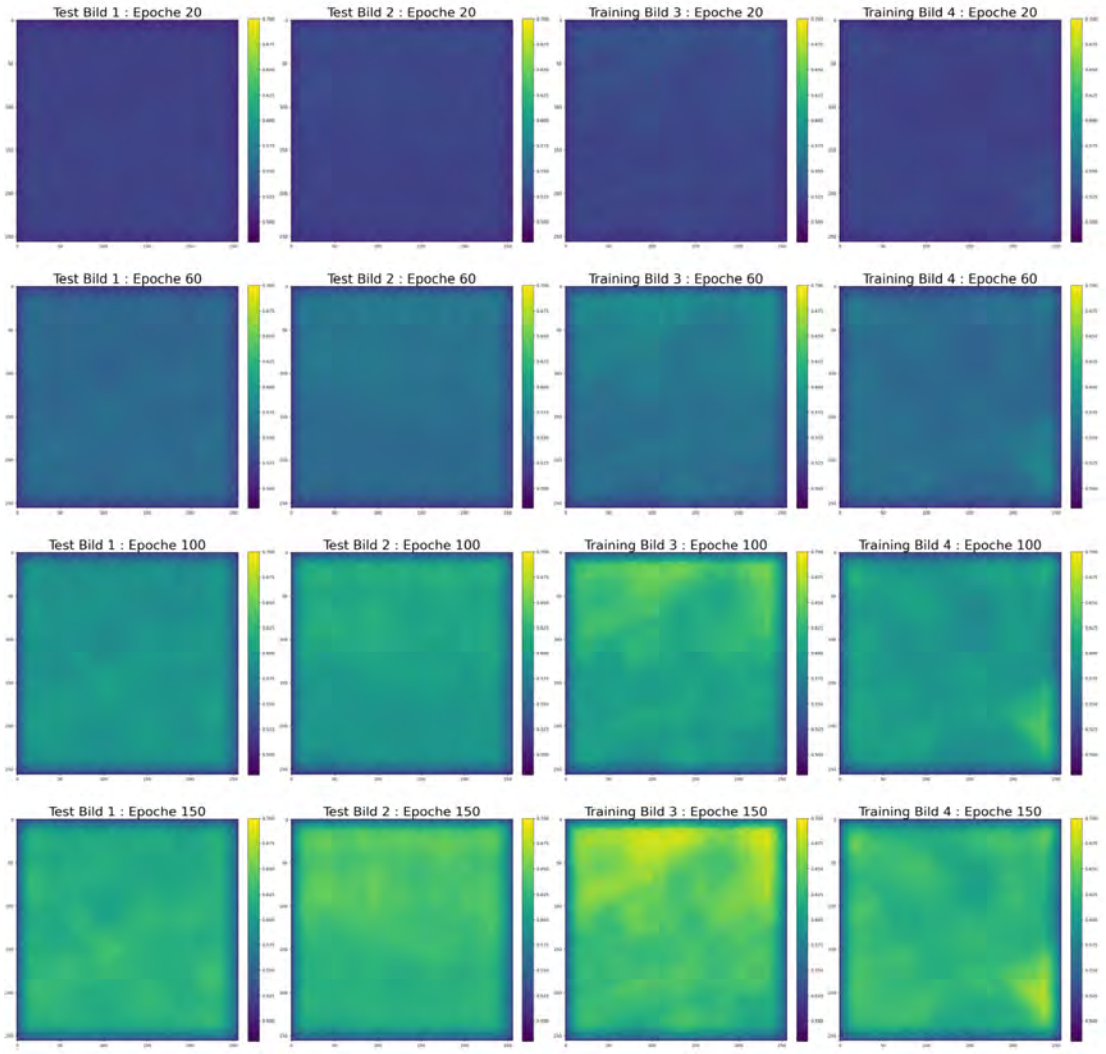

Abbildung A.14: Potentialkarten Netz: Adadelta und MSE [Skala: 0,48 - 0,7]

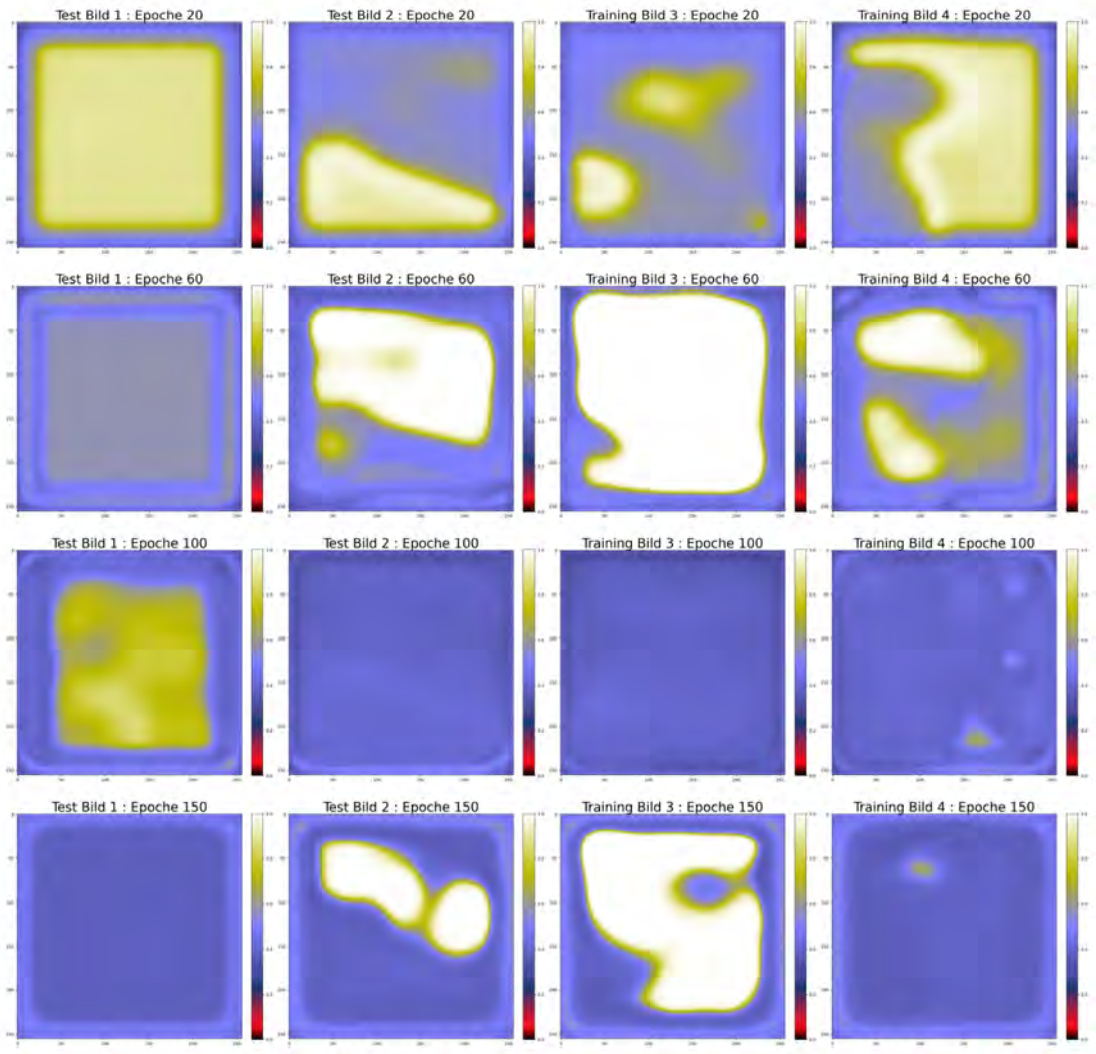

Abbildung A.15: Potentialkarten Netz: Andere Architektur, Adam und MSE [Skala: 0 - 1]

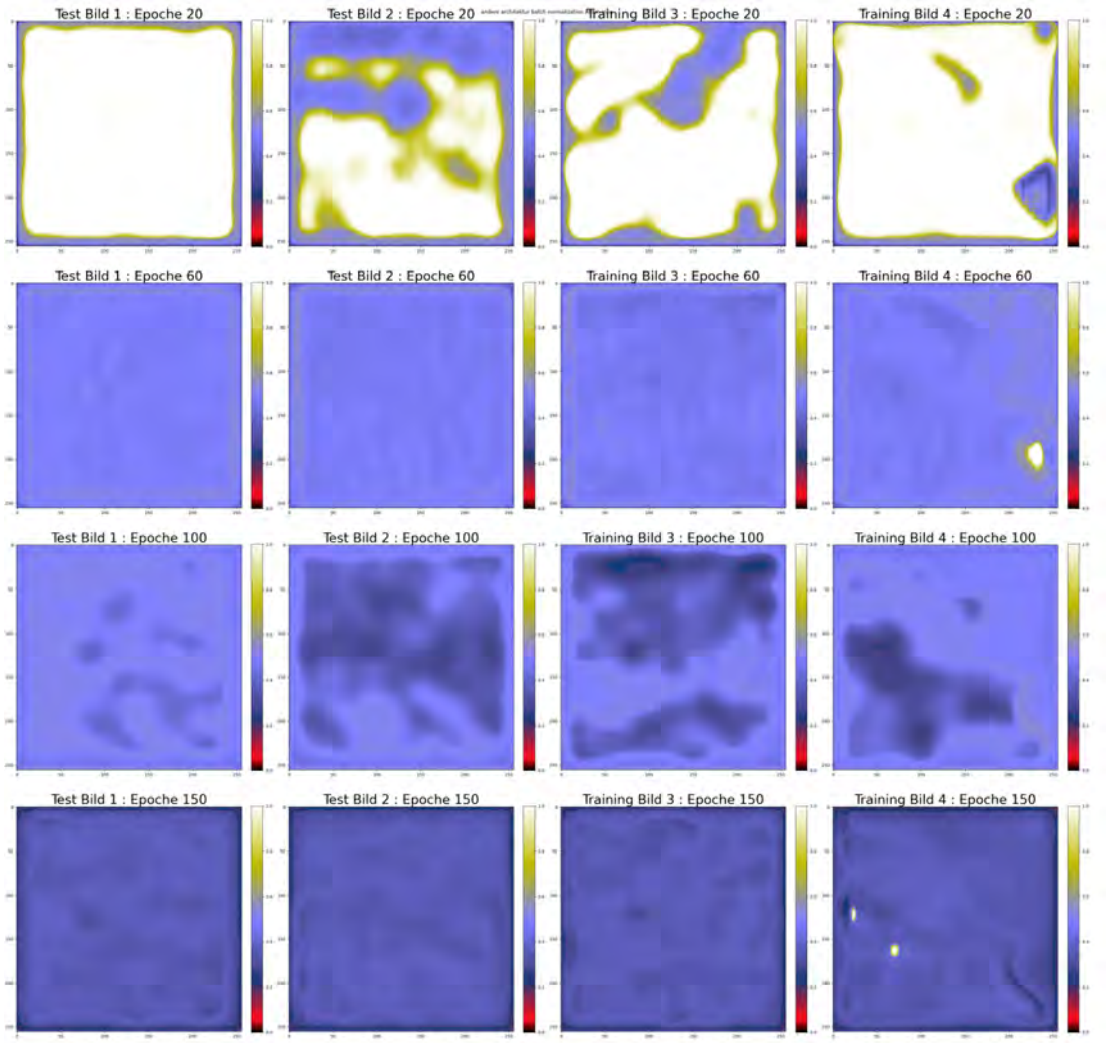

Abbildung A.16: Potentialkarten Netz: Andere Architektur, Batch Normalization, Adam und MSE [Skala: 0 - 1]

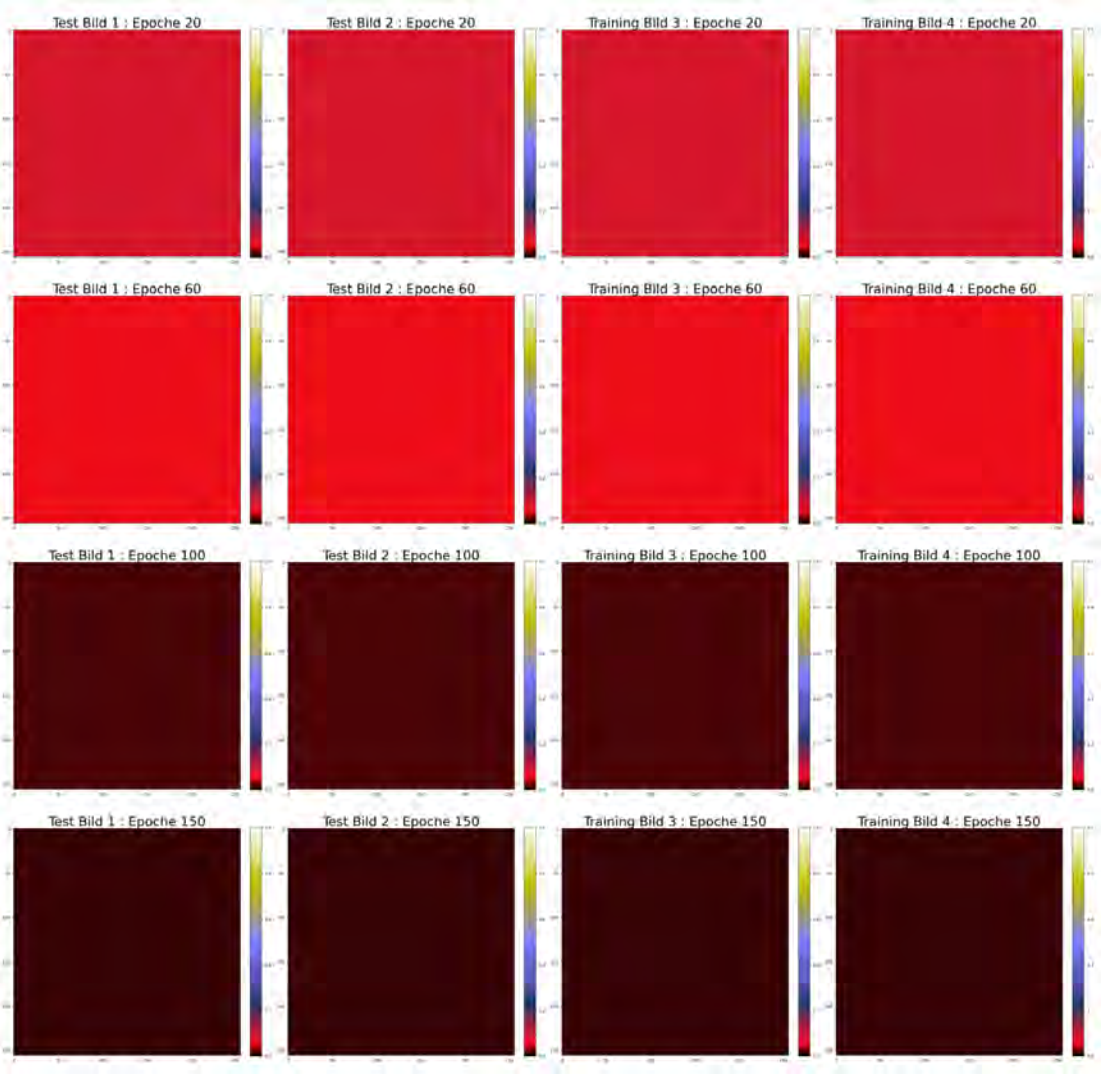

Abbildung A.17: Potentialkarten Netz: Andere Architektur, Dilation, Batch Normalization, Adam und MAE [Skala: 0 - 1]

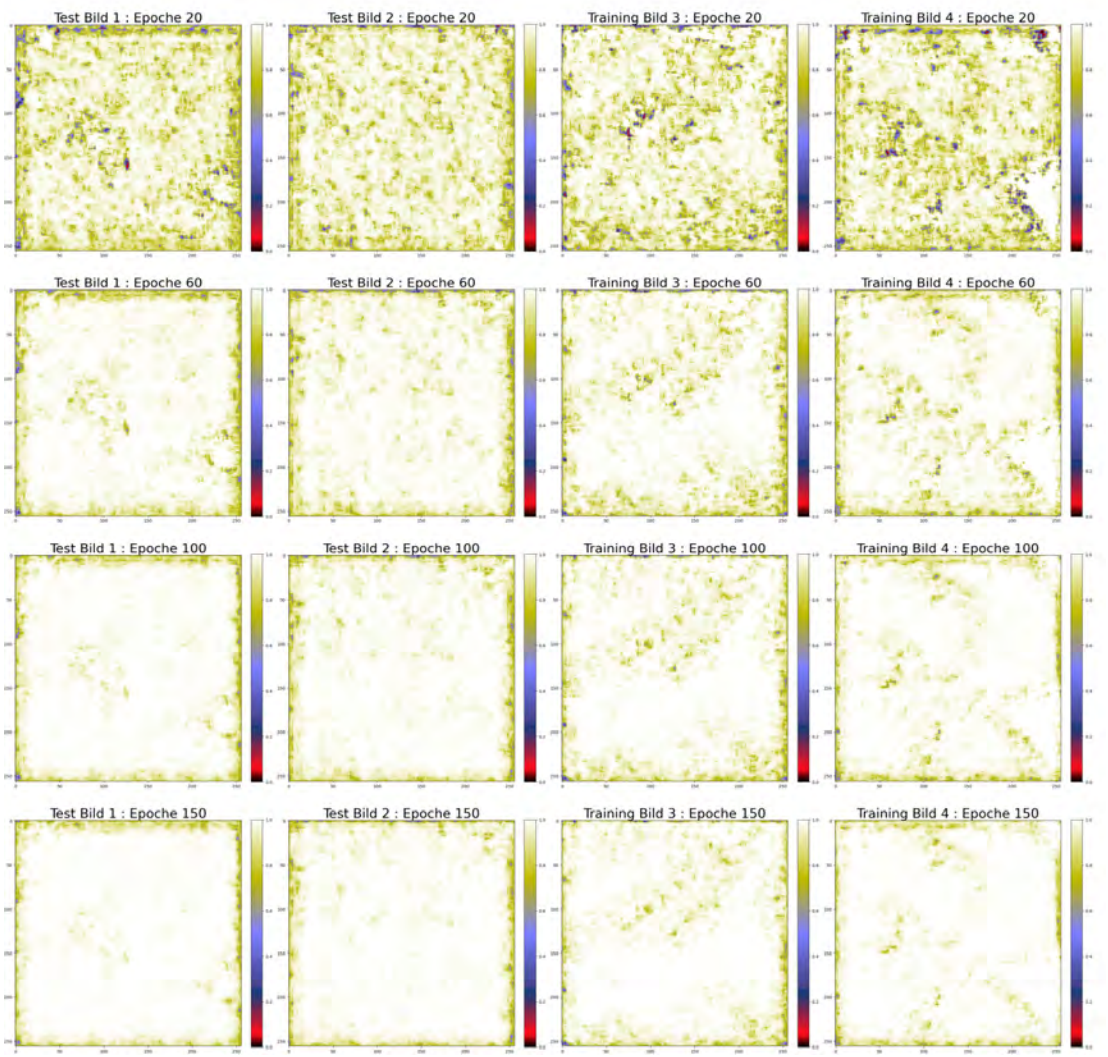

Abbildung A.18: Potentialkarten Netz: Andere Architektur, Dilation, Batch Normalization, Adam und MSE [Skala: 0 - 1]

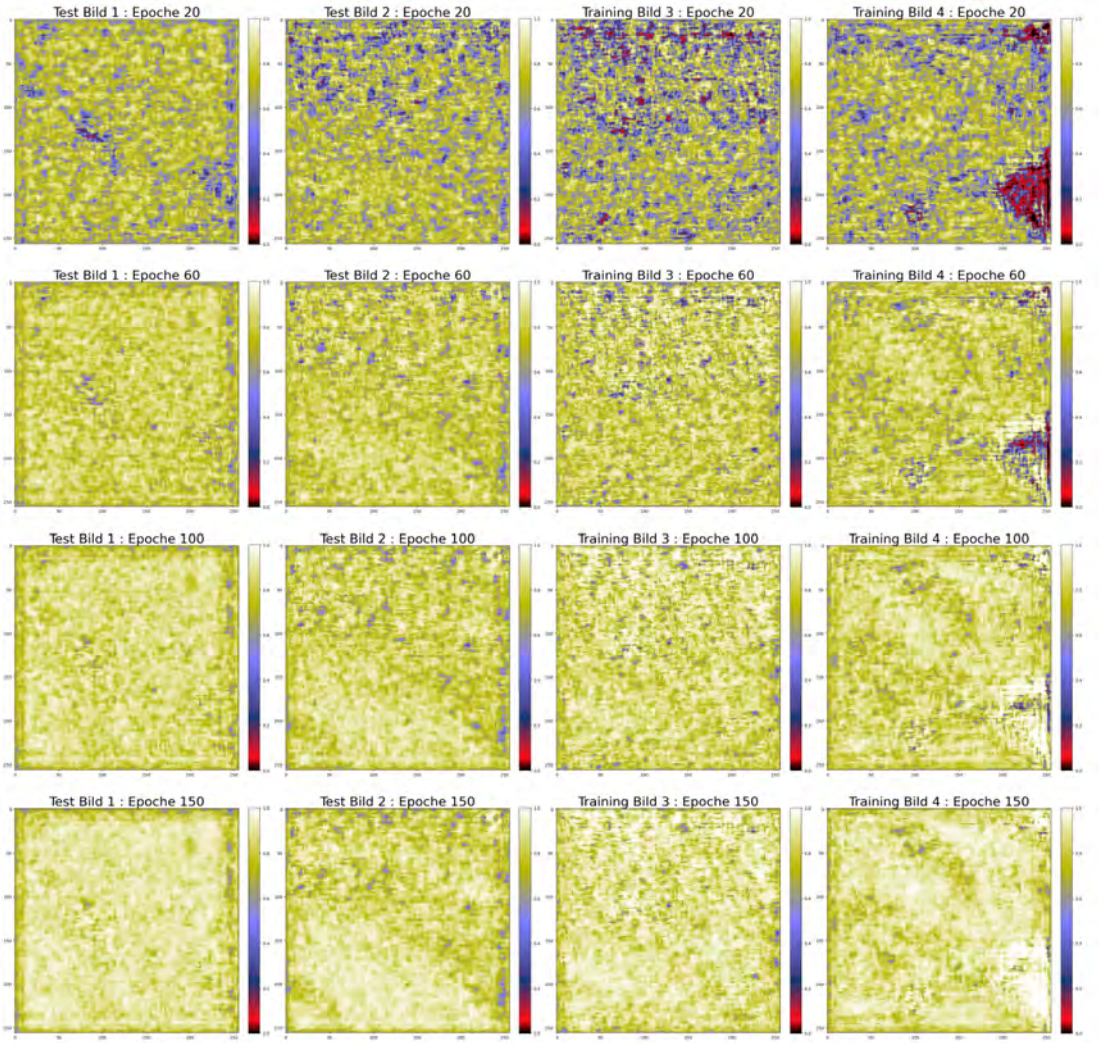

Abbildung A.19: Potentialkarten Netz: Batch Normalization, Adadelta und MSE [Skala: 0 - 1]

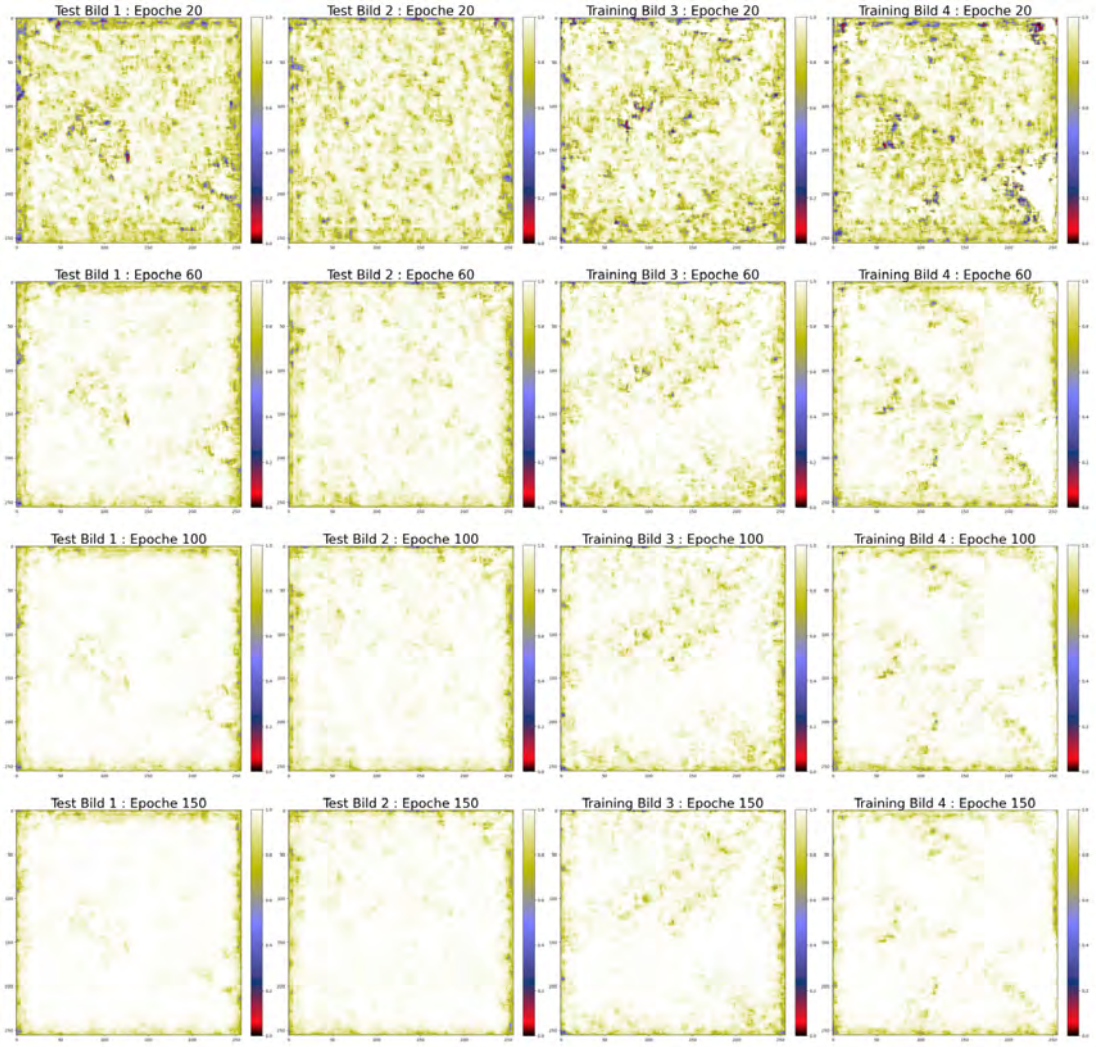

Abbildung A.20: Potentialkarten Netz: Batch Normalization, Adadelta und MAE [Skala: 0 - 1]

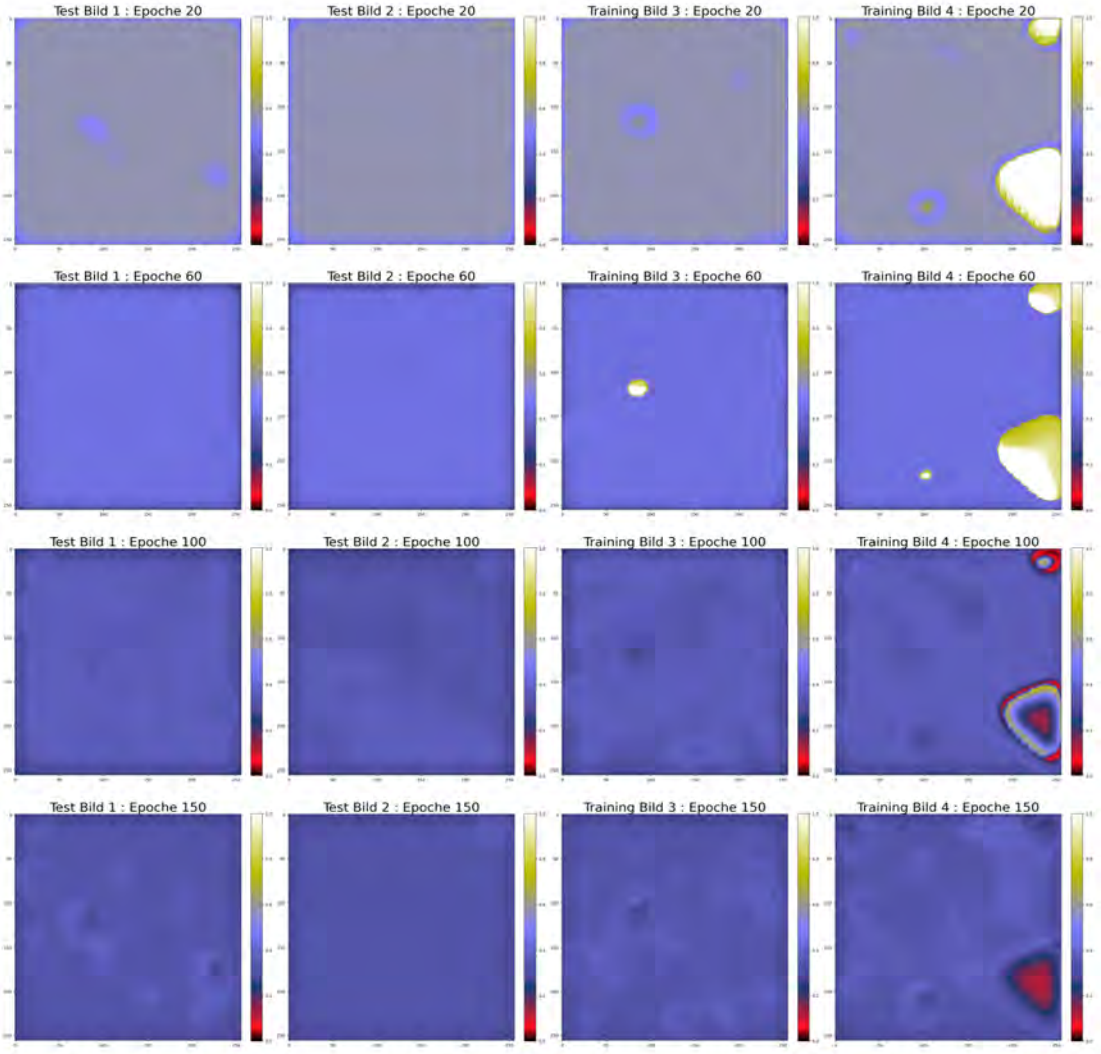

Abbildung A.21: Potentialkarten Netz: Batch Normalization, Adam und MSE [Skala: 0 - 1]

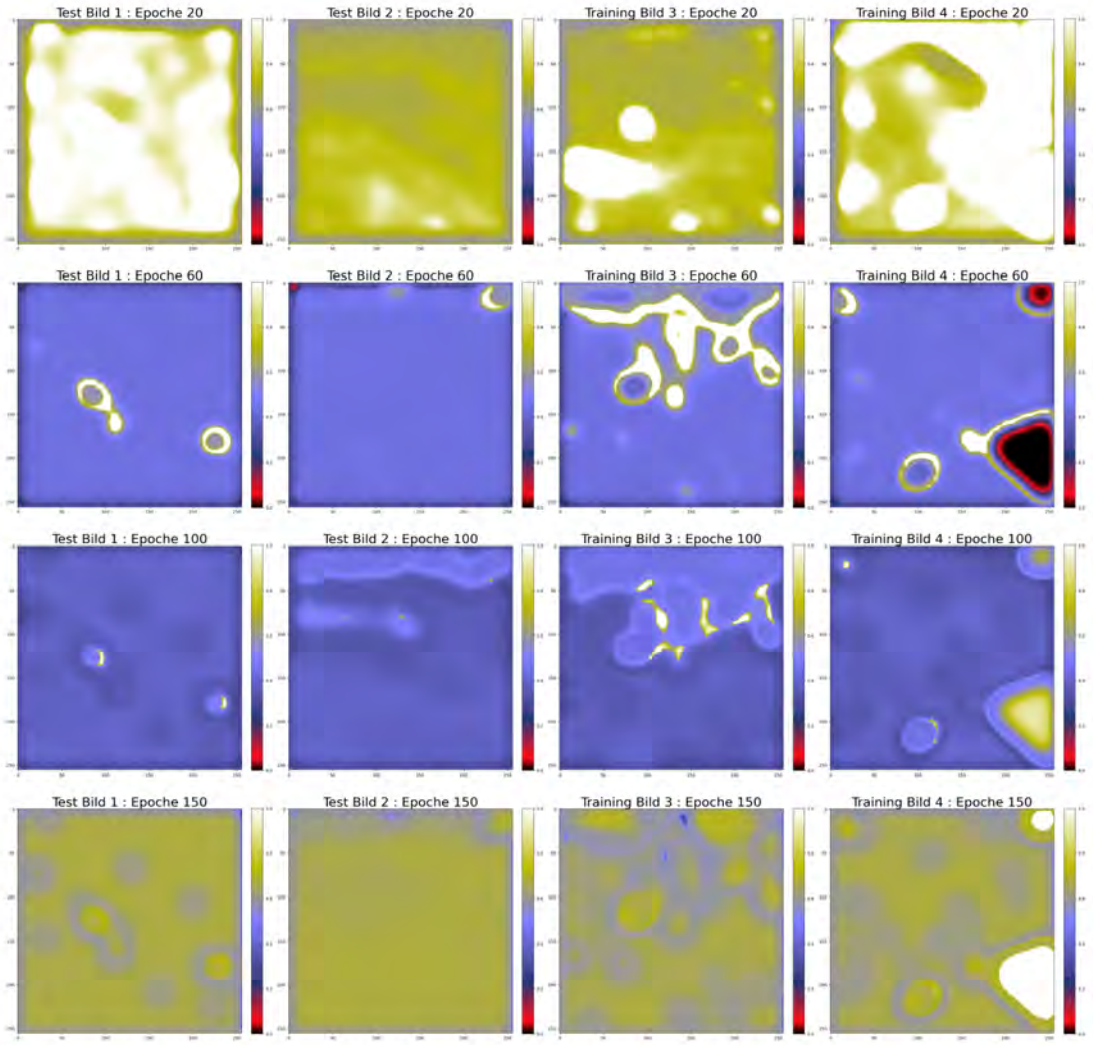

Abbildung A.22: Potentialkarten Netz: Batch Normalization, Adam und MAE [Skala: 0 - 1]

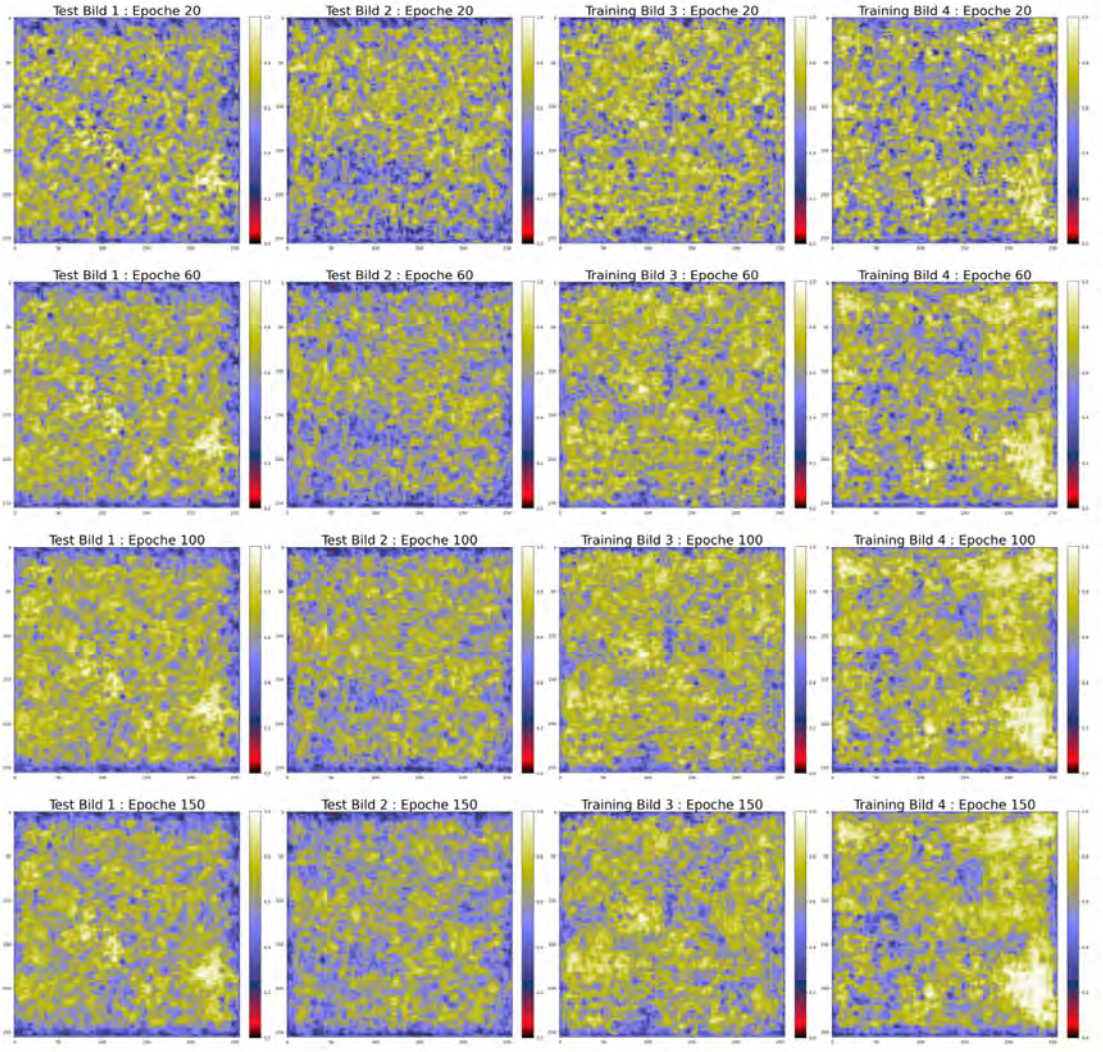

Abbildung A.23: Potentialkarten Netz: Batch Normalization, Dropout und Adadelta [Skala: 0 - 1]

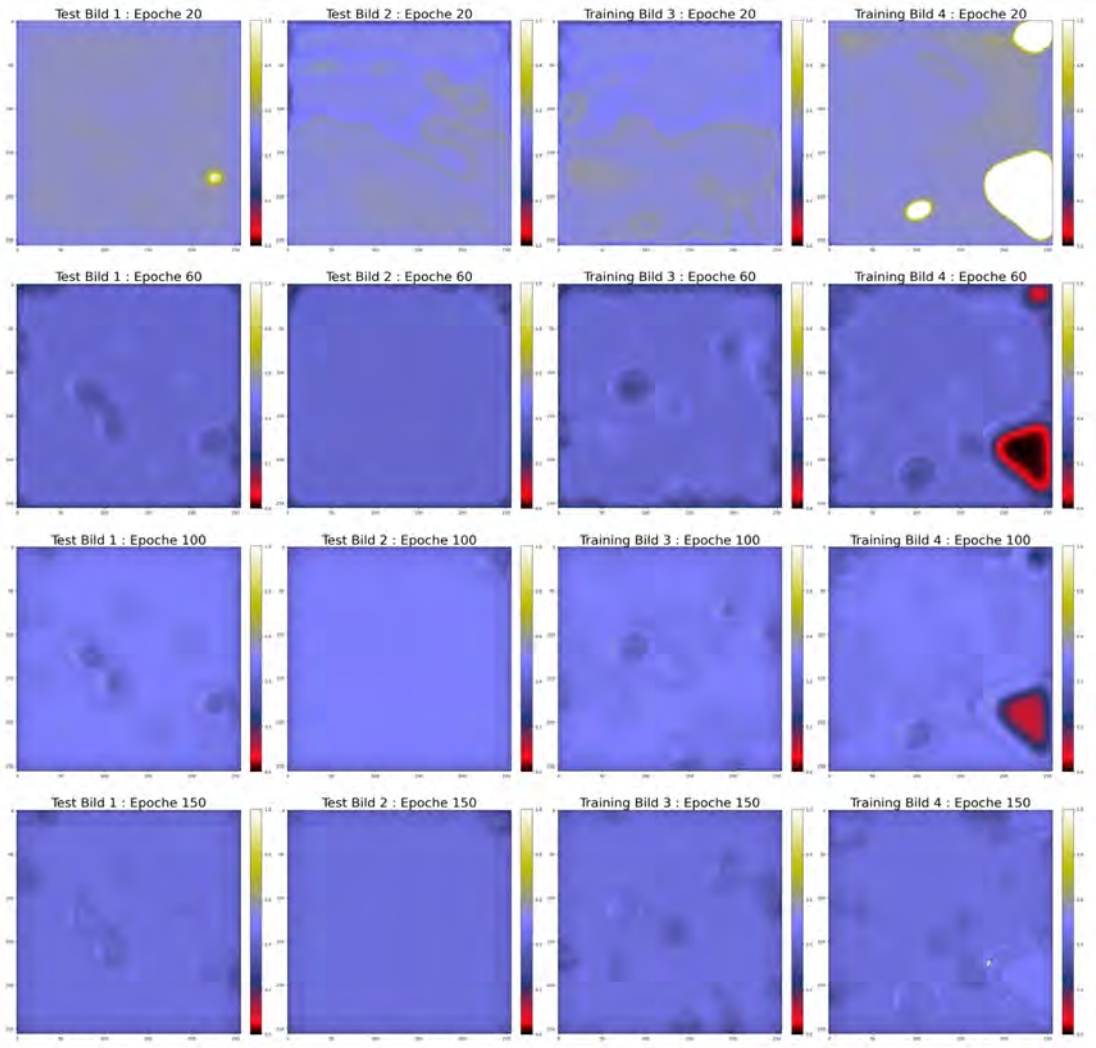

Abbildung A.24: Potentialkarten Netz: Batch Normalization, Dropout und Adam [Skala: 0 - 1]

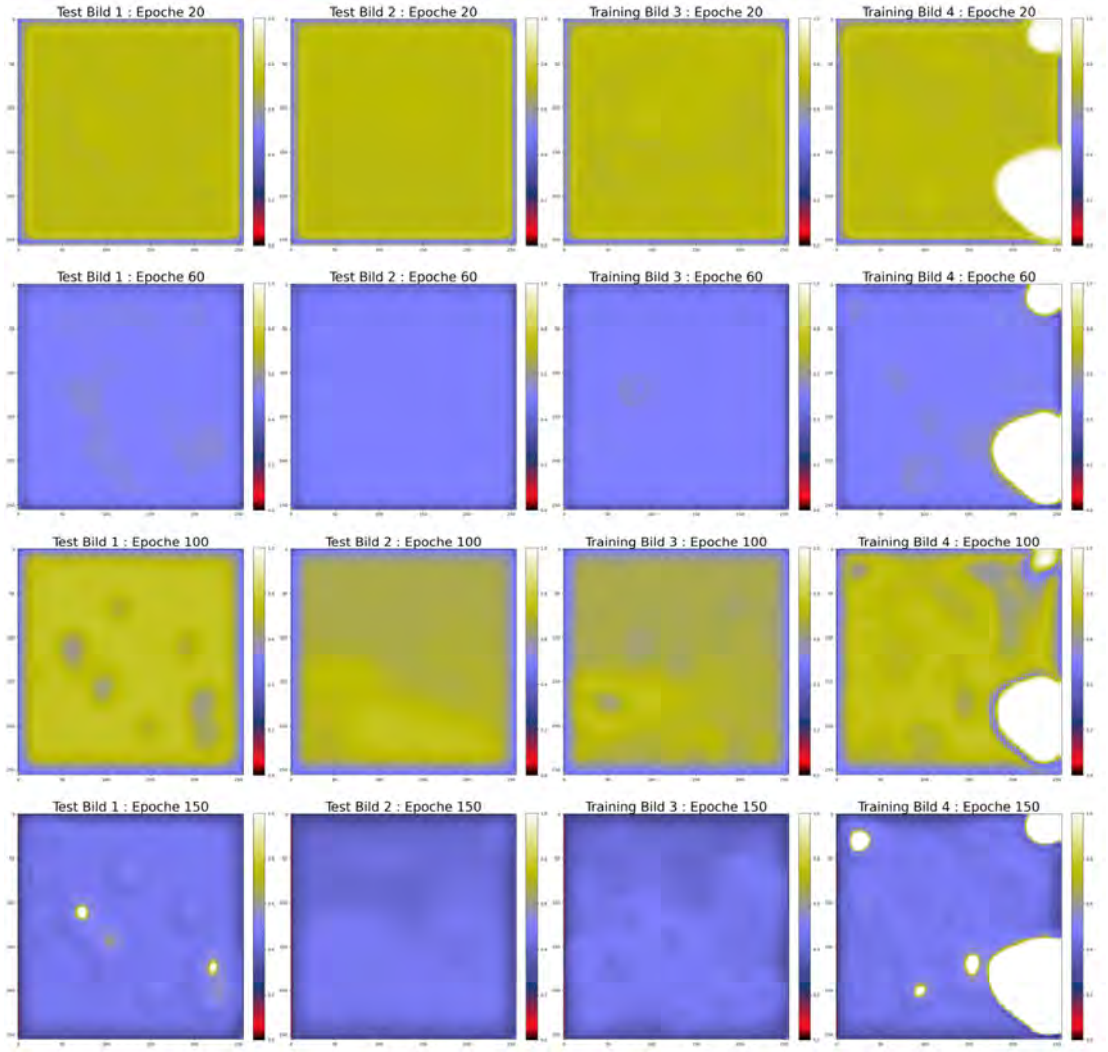

Abbildung A.25: Potentialkarten Netz: Dilation 2, Batch Normalization, Adam und MSE [Skala: 0 - 1]

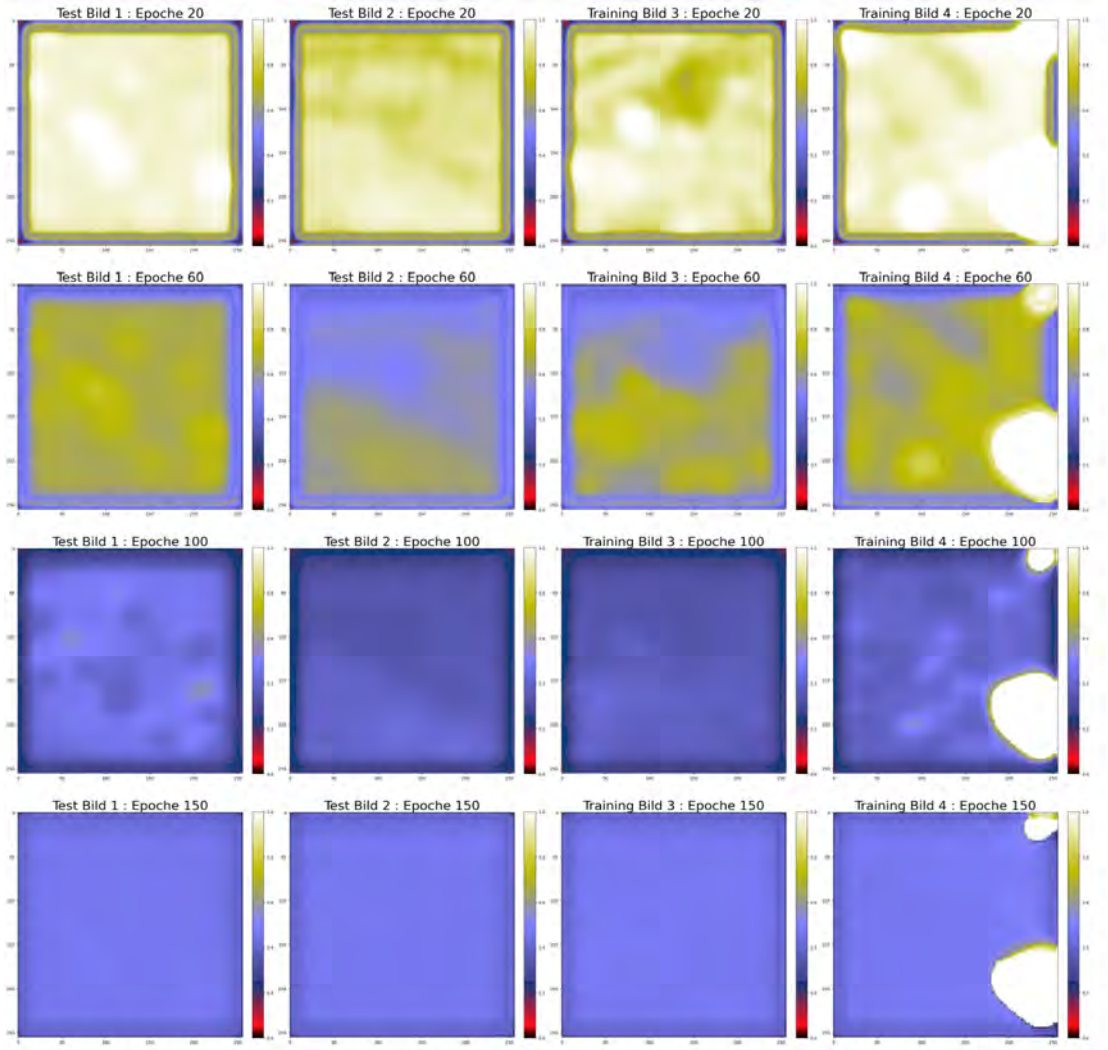

Abbildung A.26: Potentialkarten Netz: Dilation 2, Batch Normalization, Adam und MAE [Skala: 0 - 1]

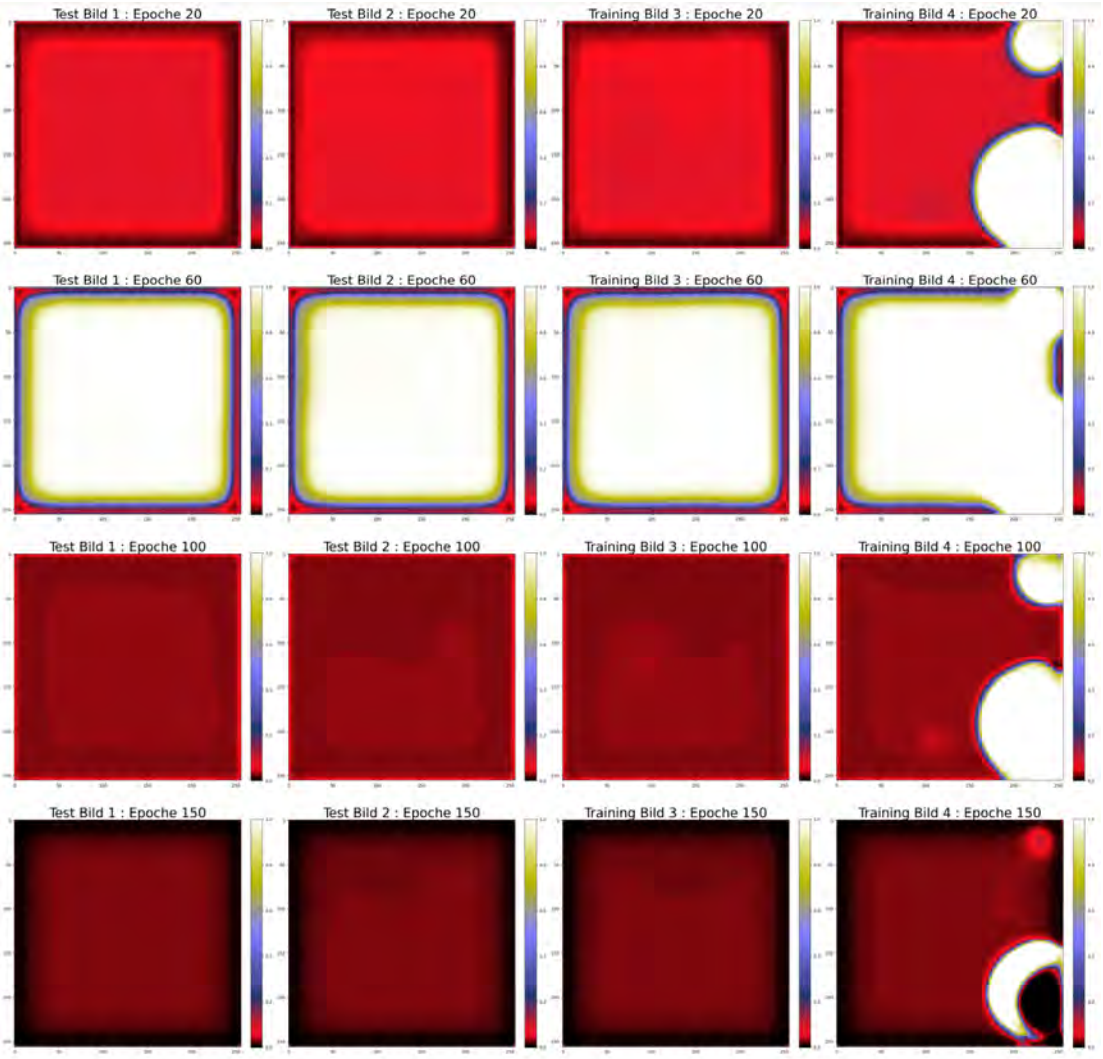

Abbildung A.27: Potentialkarten Netz: Dilation 2, Kernelgröße 5, Batch Normalization, Adam und MSE [Skala: 0 - 1]

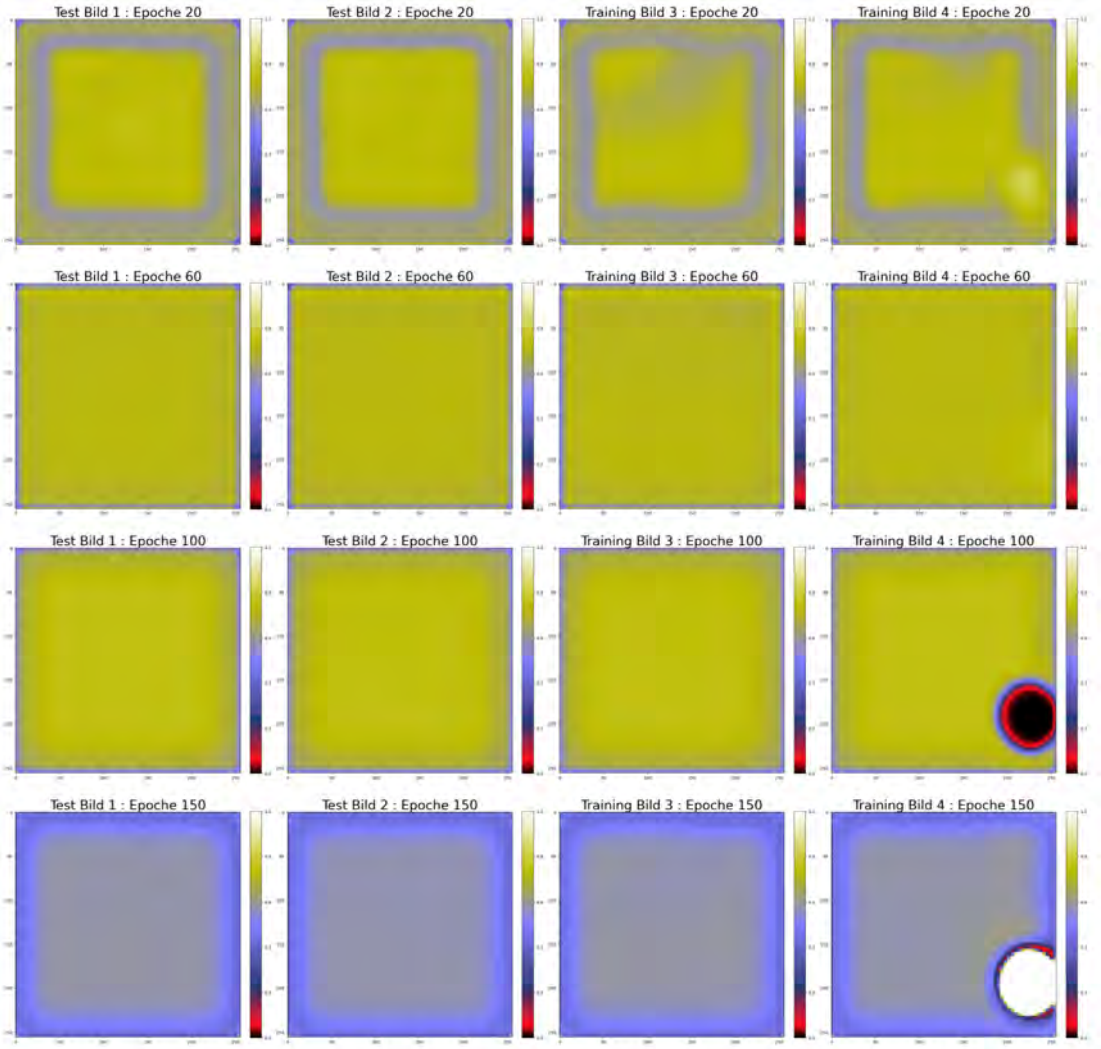

Abbildung A.28: Potentialkarten Netz: Dilation 2, Kernelgröße 5, Batch Normalization, Adam und MAE [Skala: 0 - 1]

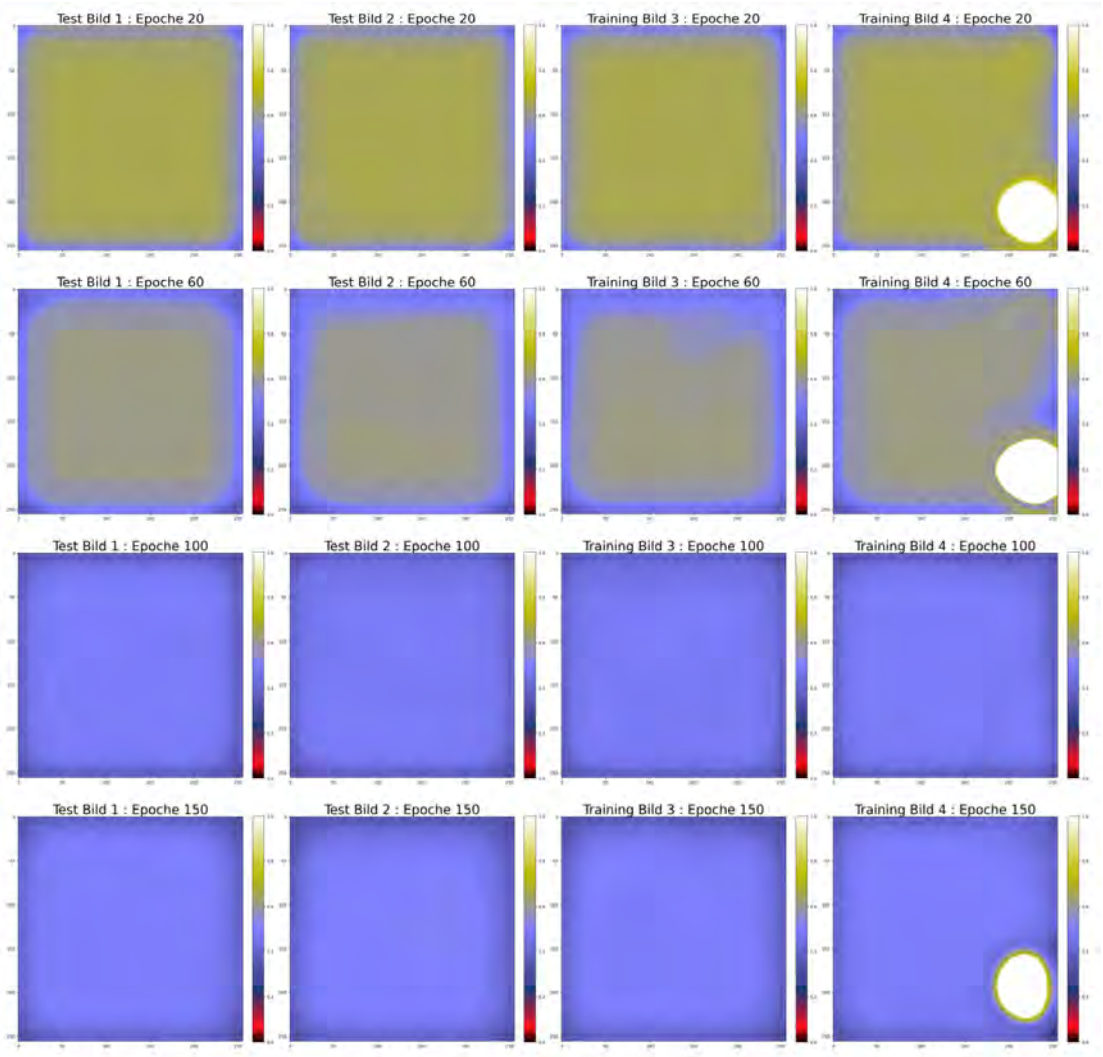

Abbildung A.29: Potentialkarten Netz: Dilation 3, Kernelgröße 5, Batch Normalization, Adam und MSE [Skala: 0 - 1]

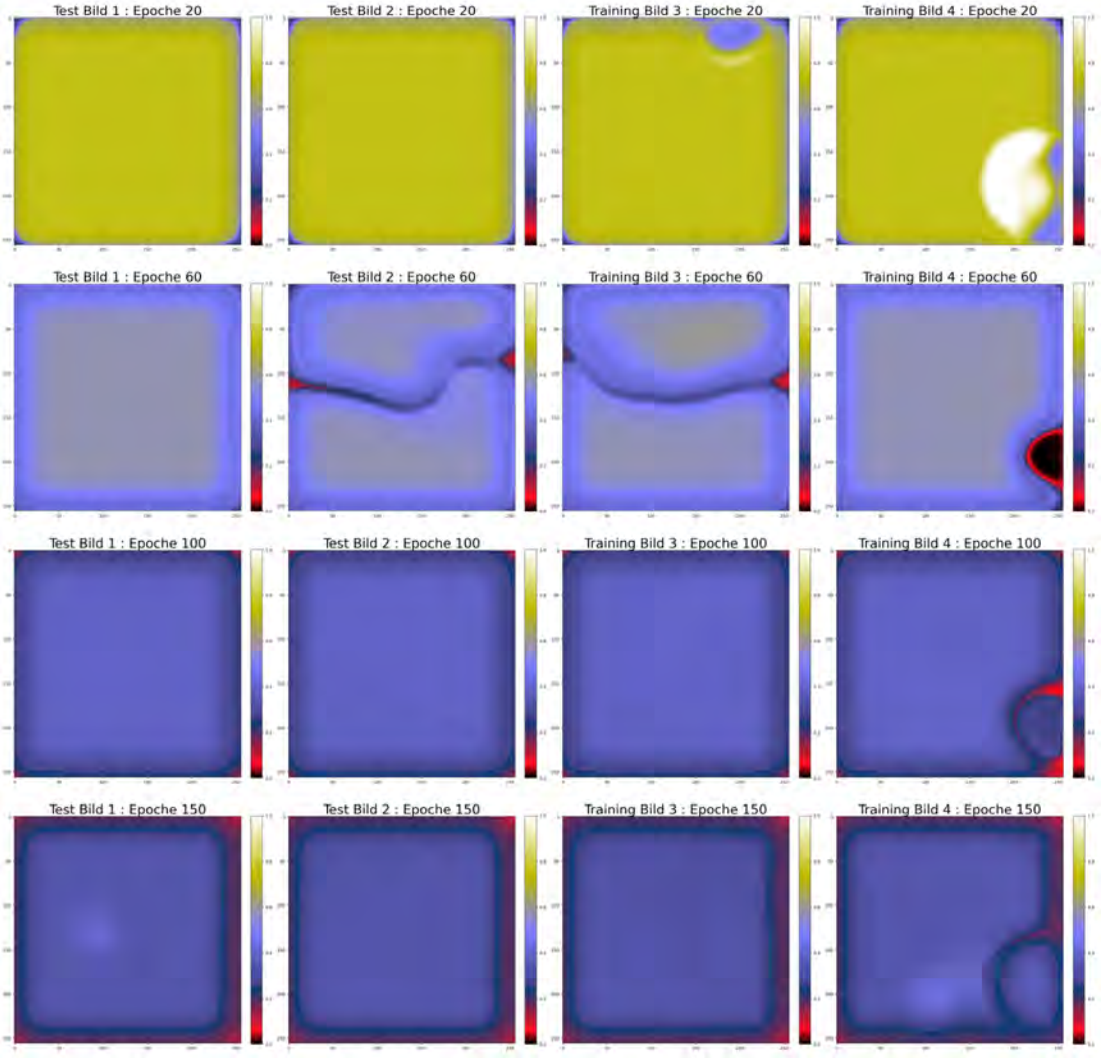

Abbildung A.30: Potentialkarten Netz: Dilation 3, Kernelgröße 5, Batch Normalization, Adam und MAE [Skala: 0 - 1]

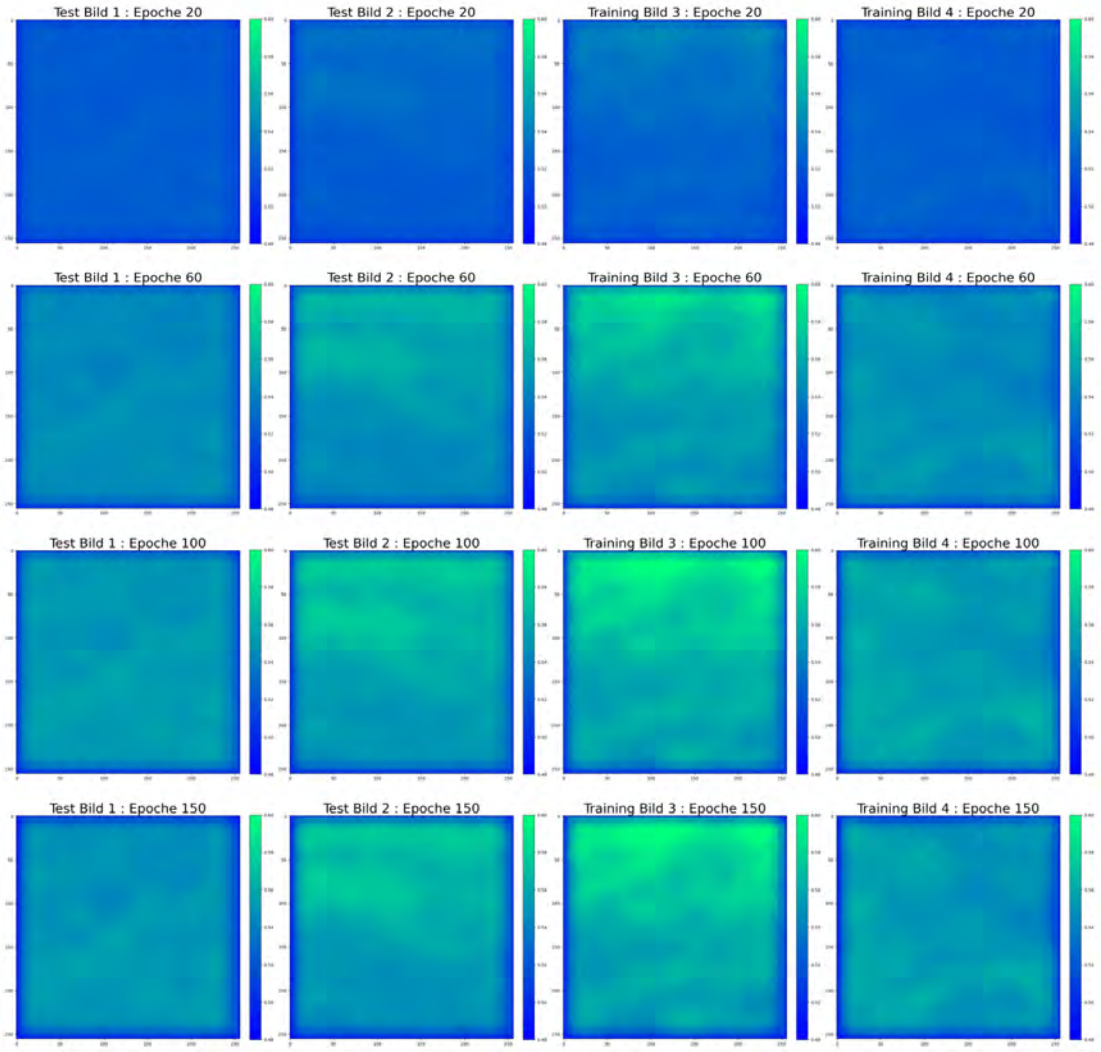

Abbildung A.31: Potentialkarten Netz: Dropout 25 %, Adadelta und MSE [Skala: 0,48 - 0,6]

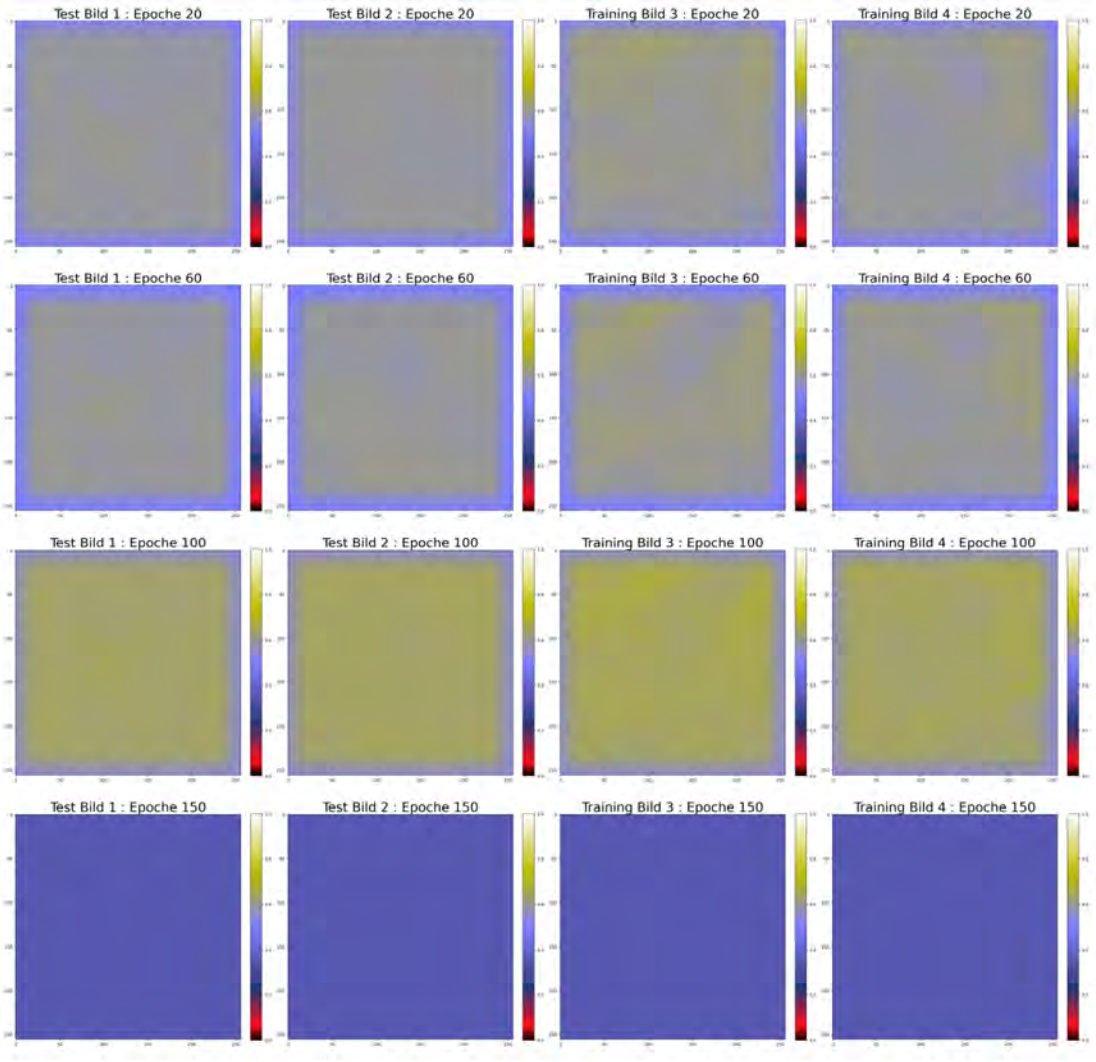

Abbildung A.32: Potentialkarten Netz: Dropout 25 %, Adam und MSE [Skala: 0 - 1]

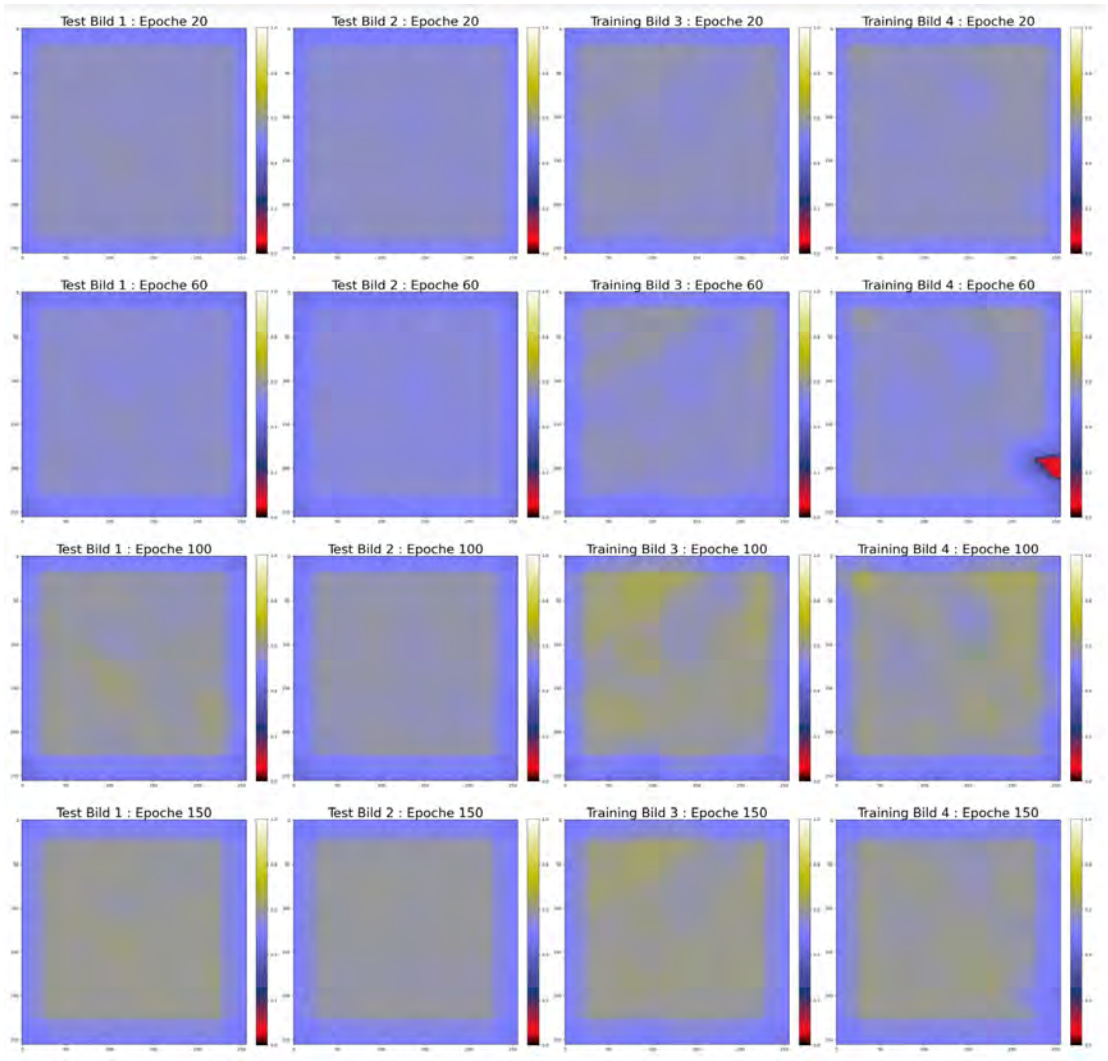

Abbildung A.33: Potentialkarten Netz: Dropout 50 %, Adam und MSE [Skala: 0 - 1]

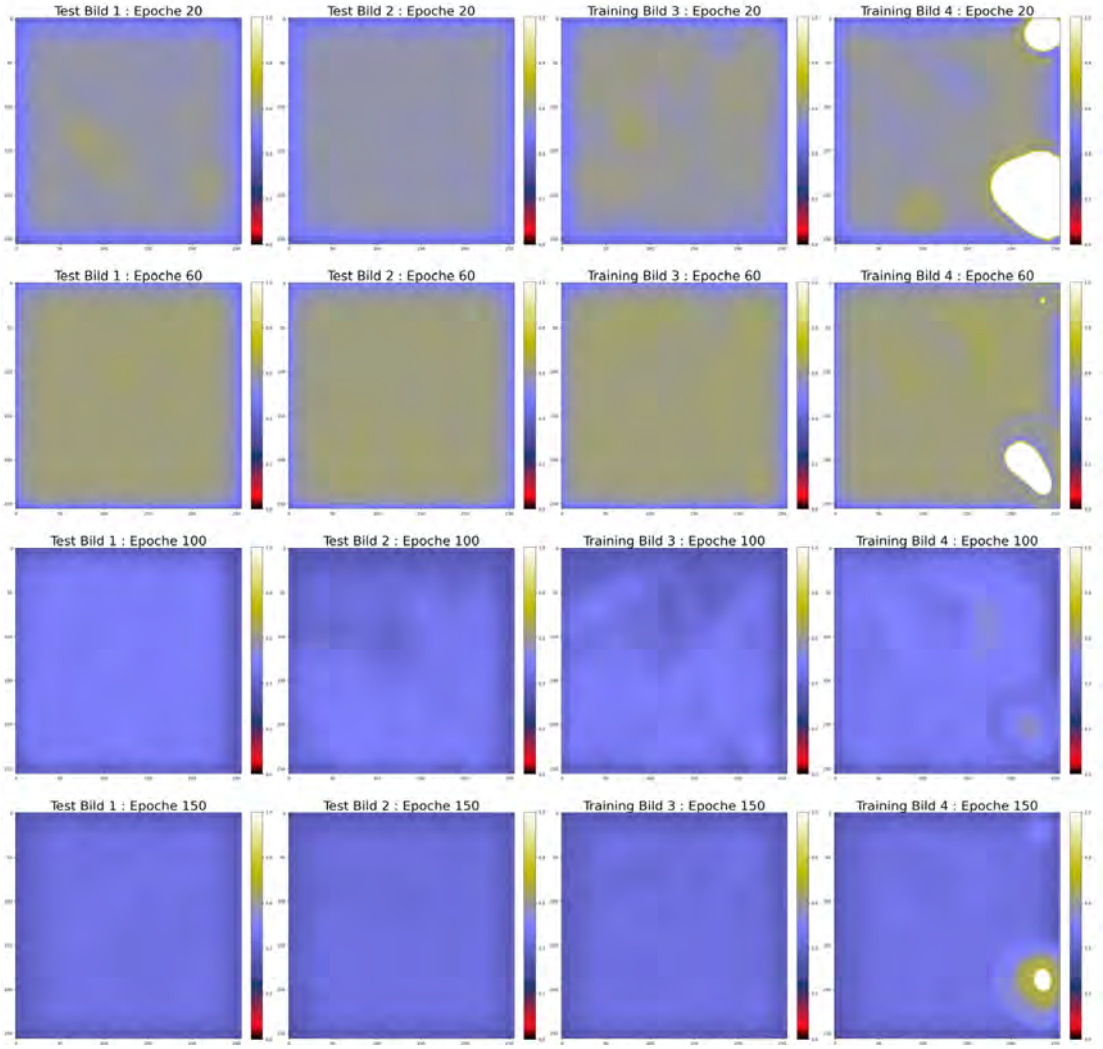

Abbildung A.34: Potentialkarten kleines Netz: Dilation 3, Kernelgröße 5, Batch Normalization, Adam und MSE [Skala: 0 - 1]

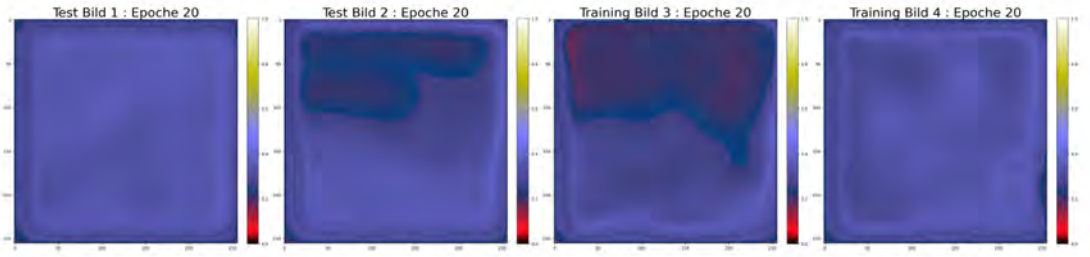

Abbildung A.35: Potentialkarten großes Netz: Batch Normalization, Adam und MSE [Skala: 0 - 1]

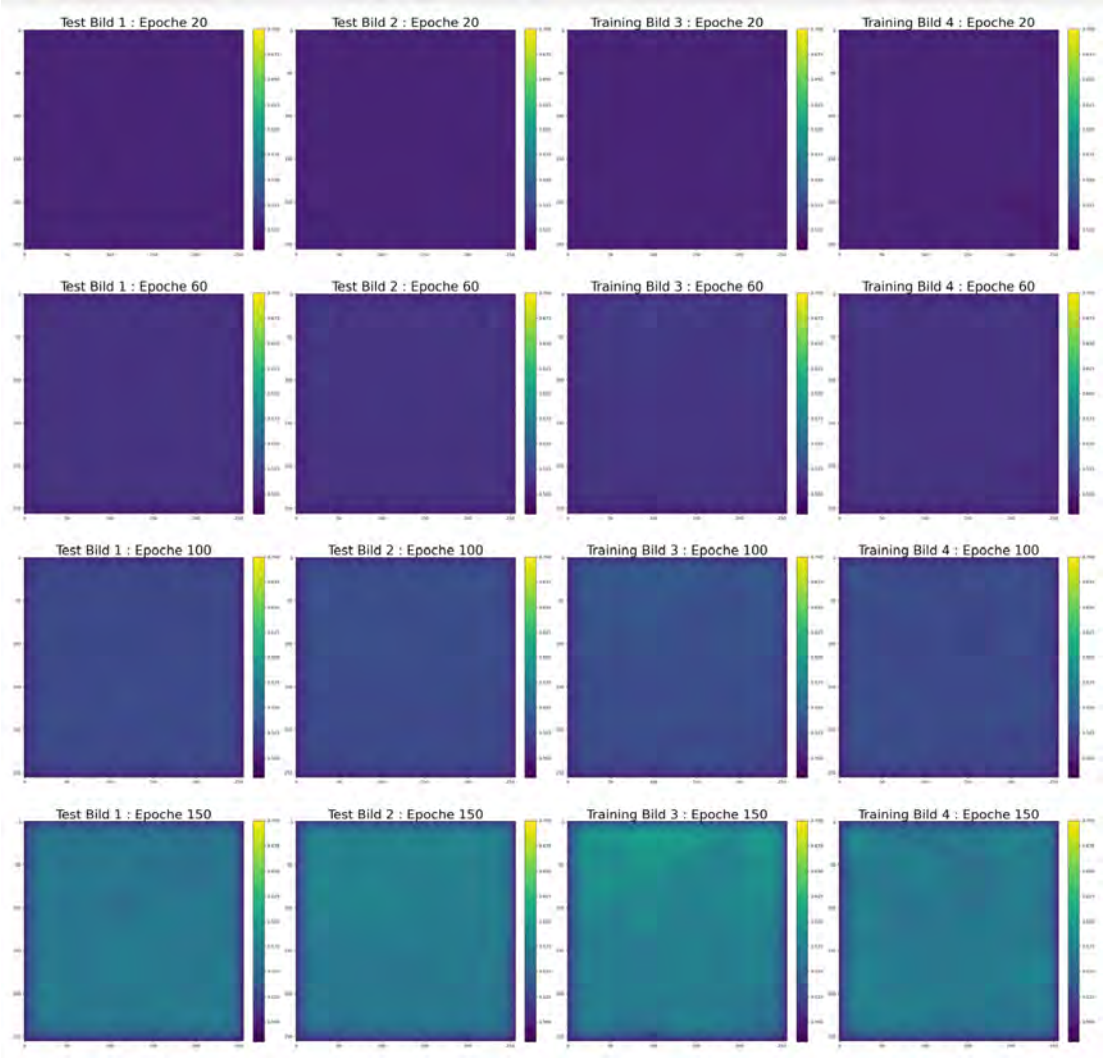

Abbildung A.36: Potentialkarten großes Netz: Adadelta und MSE [Skala: 0,48 - 0,7]

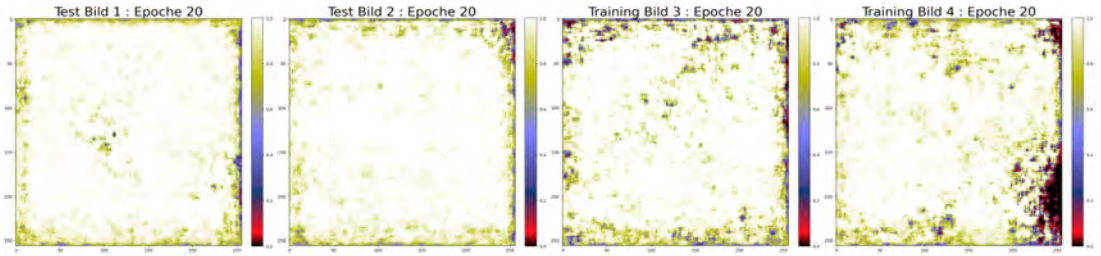

Abbildung A.37: Potentialkarten großes Netz: Batch Normalization, Adadelta und MSE [Skala: 0 - 1]

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original