NEUE
KANTONSSCHULE
AARAU

ReSketch AI

Maturarbeit

Ian Wasser, Robin Steiner

11. Oktober 2022

Betreut durch: Nicolas Ruh
Zweitbeurteilung: Dieter Koch

NKSA G19E

Abstract

ReSketch ist eine künstliche Intelligenz, die versucht Strichbilder, wie zum Beispiel Ziffern oder Buchstaben, auf eine physische Weise nachzuzeichnen. Um die Frage zu beantworten, in wiefern das möglich ist, sind definierende Kriterien des Nachzeichnes festgelegt. So soll die künstliche Intelligenz zum Beispiel nur Bewegungen ausführen können, die auch mit einem Stift möglich wären. Die künstliche Intelligenz erlernt das Nachzeichnen nach diesen Kriterien durch Deep Q-Learning, einem Reinforcement Learning Modell. Das Modell basiert auf der Arbeit hinter Doodle-SDQ (Zhou et al., 2018), erfährt aber konzeptuelle Variationen wie die Integration einer Physiksimulation. Die künstliche Intelligenz ist auf das Nachzeichnen von Ziffern trainiert. Ein Test dieser trainierten künstlichen Intelligenz auf Buchstaben und andere Arten von Strichbildern führt zur Antwort auf die Frage, ob eine künstliche Intelligenz das Nachzeichnen im Allgemeinen erlernen kann. Testchange

Vorwort

Diese Arbeit ist eine Untersuchung über künstliche Intelligenz. Die Fragestellung der Untersuchung wird mithilfe einer selbst programmierten künstlichen Intelligenz beantwortet.

Wir haben uns für das Thema Künstliche Intelligenz entschieden, weil damit praktische Arbeit mit intellektueller Forschung verbunden werden kann. Das Thema ermöglicht ausgeprägte, praktische Programmierarbeiten, was uns zuspricht. Zusätzlich ermöglicht künstliche Intelligenz einfache Forschung. Mit einfacher Forschung ist dabei nicht der Grad der Komplexität gemeint, sondern die Vielfalt der Möglichkeiten. Es gibt Aspekte und Anwendungen der künstlichen Intelligenz, die für uns zugänglich sind und noch nicht zu weit erforscht sind, um neue Ideen zu finden. Ausserdem benötigt die Forschung an künstlicher Intelligenz nur einen Computer. Experimente und Tests können durch Programmcode ausgeführt werden. Die Auswertung der Experimente findet auf dem selben Computer statt und die Genauigkeit der Ergebnisse stellt ebenfalls kein Problem dar, da der Computer die Zahlen direkt berechnet. Der Computer ist eine optimale Umgebung für eine erste Forschungsarbeit.

Diese Arbeit ist für uns eine erste vertiefte Erfahrung mit dem grossen Gebiet der künstlichen Intelligenz. Wir erhoffen uns durch diese Erfahrung einen erweiterten Horizont, neues Wissen und verbesserte Programmierkenntnisse.

Vielen Dank an unseren Betreuer, Dr. Nicolas Ruh, für die hilfreichen Vorschläge, die ausgeprägte Beratung und das Vertrauen in uns. Vielen Dank auch an Günther Wasser für das Korrekturlesen dieser Arbeit.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der Computer ist ein Werkzeug, das dem Menschen Arbeit abnehmen kann. Um komplizierte Aufgaben zu übernehmen, muss sich der Computer jedoch an menschliches Verhalten, menschliches Urteilsvermögen und menschliche Intelligenz annähern. Mit anderen Worten braucht der Computer, oder das steuernde Computerprogramm, eine künstliche Intelligenz. Ein Intelligentes Computerprogramm zu entwickeln ist komplex. Der fähigste und am weitesten verbreitete Ansatz liefert Machine Learning. Diese Arbeit selbst ist eine Untersuchung Im Bereich Machine Learning. Spezifischer ist die Arbeit im Bereich Reinforcement Learning, einem Teilgebiet von Machine Learning.

Die Fragestellung der Untersuchung lautet: Kann eine künstliche Intelligenz lernen, Strichbilder auf eine physische Weise nachzuzeichnen, sodass diese durch ein automatisches System richtig erkannt werden können?

Für ein gegebenes Strichbild soll die künstliche Intelligenz (KI) erlernen, ein möglichst gleiches Bild daneben zeichnen können. Die Frage ist, ob die KI das Nachzeichnen genug gut lernen kann, damit die Zeichnung von einem automatischen System richtig erkannt wird. Richtig erkannt heisst in diesem Fall vereinfacht, dass eine zweite KI in der Zeichnung das selbe Motiv wie in der Vorlage erkennt. Wenn das zutrifft, kann die künstliche Intelligenz erfolgreich nachzeichnen. Es existieren allerdings weitere Kriterien, die die Leistung der KI bei der Tätigkeit des Nachzeichnens beurteilen.

Nachzeichnen ist eine menschliche Tätigkeit. Menschen führen beim Zeichnen durch gewisse Handbewegungen einen Stift, wodurch das Nachzeichnen mit physischen Einschränkungen verbunden ist. Der Stift kann sich nicht teleportieren, sondern sich nur mit einer bestimmten Geschwindigkeit fortbewegen. Die KI soll das Nachzeichnen mit ähnlichen physischen Einschränkungen erlernen. Mit anderen Worten soll die KI lernen, einen Stift zu führen. Die physischen Einschränkungen sind dabei jedoch simuliert und im Vergleich zu der echten Welt vereinfacht.

Die KI soll das Nachzeichnen von Strichbildern allgemein erlernen. Strichbilder können Zahlen, Buchstaben, Formen, Symbole und allgemeine Kritzeleien sein. Natürlich kann die KI nicht mit allen Arten von Strichbildern trainiert werden, weil die Vielfalt zu gross ist. Daraus ergibt sich die Frage, wie gut die künstliche Intelligenz Arten von Strichbildern nachzeichnet, die nicht im Training enthalten waren.

Die vorangehenden Überlegungen sind in einer Sammlung an Unterfragen, die in dieser Arbeit beantwortet werden, vertreten. Die Unterfragen lauten:

- Wie kann die Architektur einer KI aussehen, die das Nachzeichnen erlernt?
- Nach welchen Kriterien lässt sich die Leistung der KI in dieser Aufgabe beurteilen?
- Wie lässt sich die Leistung der KI in dieser Aufgabe verbessern?
- Wie ändert sich die Leistung der KI für Strichbilder, die im Training nicht enthalten sind?
- Welche Einflüsse haben physische Einschränkungen auf die Leistungs der KI?
- Wie und in wiefern lässt sich die KI mit menschlichem Zeichnen vergleichen?

Kapitel 2

Theoretische Grundlagen

Die theoretischen Grundlagen führen die Konzepte ein, die über die ganze Arbeit hinweg Anwendung finden. Es handelt sich dabei um Zusammenfassungen. Die Theorie wird auf den Teil reduziert, der für ein grundsätzliches Verständnis der Arbeit nötig ist. Weitere Informationen sind in den referenzierten Quellen einsehbar. Auch die verwendeten Fachbegriffe werden in diesem Kapitel eingeführt.

2.1 Machine Learning

Machine Learning ist ein Teilbereich der künstlichen Intelligenz. "Künstliche Intelligenz (KI) bezieht sich im Allgemeinen auf jedes menschenähnliche Verhalten durch eine Maschine oder ein System" („What is Artificial Intelligence (AI)?", n. d.) Mit Maschinen und Systemen sind in den allermeisten Fällen Computer, beziehungsweise die steurenden Computerprogramme gemeint. Diese Computerprogramme bilden ein Modell von menschlichem Verhalten. Machine Learning Modelle entwickeln (oder erlernen) eine Mustererkennung durch die Analyse von Daten („What is Machine Learning?", n. d.). Mustererkennung bedeutet hier, dass der Algorithmus Zusammenhänge zwischen den analysierten Daten erkennt und auf dieser Basis Vorhersagen treffen kann. Vereinfacht gesagt, versucht ein Machine Learning Modell menschliches Urteilsvermögen zu erlernen (Spaulding, 2020).

Ein Beispielproblem für ein Machine Learning Modell ist die Erkennung von handschriftlichen Zahlen. Ein Computerprogramm soll durch den Input eines Bildes mit einer handschriftlichen Zahl eine korrekte Beurteilung treffen, um welche Zahl es sich handelt. Mit anderen Worten soll der Output des Computerprogrammes der Zahl entsprechen, die auf dem Bild der Eingabe zu sehen ist (siehe Abbildung 2.1). Jedes Computerprogramm, das dieses Problem löst, fällt in den Bereich der künstlichen Intelligenz. Machine

Learning Modelle geben einen Ansatz für die Umsetzung eines solchen Computerprogrammes.

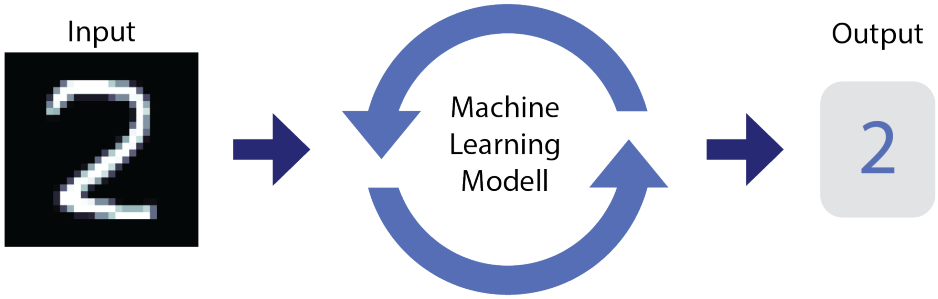

Abbildung 2.1: Erkennung von handgeschriebenen Zahlen durch ein Machine Learning Modell (eigene Abbildung)

Machine Learning Modelle, die das Beispielproblem lösen, basieren üblicherweise auf Supervised Learning. Das ist ein Teilbereich von Machine Learning, wobei das Machine Learning Modell aus Rückmeldungen der korrekten Beurteilung, der Zielvariable, als Reaktion auf ihre eigenen Beurteilungen lernt („Was ist Supervised Learning (Überwachtes Lernen)?“, n. d.). Die Zielvariable muss dabei im Voraus für jeden Datenpunkt in den analysierten Daten durch einen Menschen festgelegt sein (Hagen et al., 2020). Ausgedrückt durch den Fachbegriff müssen die Daten labeled sein („2.1 What is the difference between labelled and unlabelled data?“, n. d.). Weitere Teilbereiche von Machine Learning sind Unsupervised Learning und Reinforcement Learning (Arora, 2020). Beachte 2.2 Reinforcement Learning für eine ausgeprägtere Einführung in Reinforcement Learning.

Machine Learning Modelle sind hauptsächlich in der programmiersprache Python implementiert (Sadie Bennett, 2019). Dabei werden häufig Tensorflow und Keras verwendet. Tensorflow ist ein Machine Learning Framework („TensorFlow“, n. d.). Das bedeutet, dass Tensorflow fertige Funktionen und Algorithmen bereitstellt, die für Machine Learning Modelle nötig sind. Keras ist ein weiteres Machine Learning Framework, das selbst mit Tensorflow funktioniert.

2.1.1 Funktionsweise eines Machine Learning Modelles

Dieser Abschnitt erklärt die Funktionsweise eines Machine Learning Modelles, basierend auf dem Beispielproblem aus dem letzten Abschnitt (siehe2.1 Machine Learning).

Bei den Daten, die das Machine Learning Modell analysiert handelt es sich in diesem Fall um das MNIST Datenset („MNIST handwritten digit database,

Yann LeCun, Corinna Cortes and Chris Burges“, 2017). Dieses Datenset wurde vom NIST (National Institute of Standards and Technology) in Amerika veröffentlicht und beinhaltet 70′000 Bilder von handgeschriebenen Zahlen („The EMNIST Dataset“, 2017). Jedes Bild hat eine Auflösung von 28×28 Pixeln (siehe Abbildung 2.2).

Abbildung 2.2: Beispiele aus dem MNIST Datenset (Eigene Abbildung)

Ein Machine Learning Modell durchläuft ein Training gefolgt von einer Testphase („Training and Test Sets“, n. d.). Während dem Training erlernt das Modell die Mustererkennung, um verlässliche Aussagen zu den Daten der Eingabe zu treffen. Die Testphase misst die Genauigkeit des Modelles, also die Wahrscheinlichkeit, mit der das Modell die richige Lösung zur Eingabe liefert. Nur in den seltensten Fällen erreicht diese Genauigkeit 100%. Das Modell garantiert somit nicht die richtige Lösung. Das Machine Learning Modell erlert die Mustererkennung während dem Training durch die Analyse von Trainingsdaten aus einem Datenset. Das Modell gibt zu jedem Datenpunkt die Beurteilung, um welche Zahl es sich handelt. Das Datenset ist labeled (siehe 2.1 Machine Learning). Falls die Beurteilung des Modelles nicht mit der bekannten, korrekten Lösung übereinstimmt, passt sich das Modell automatisch an. Dadurch soll die Beurteilungen für zukünftige Datenpunkte genauer werden. Die Testphase misst die Genauigkeit des Modelles auf Testdaten. Die Testdaten bestehen aus Datenpunkten, die in den Trainingsdaten nicht enthalten sind.

Zusammengefasst kann ein Machine Learning Modell Daten Beurteilen und sich selbst Anpassen, um die Beurteilungen zu verbessern. Künstliche Neuronale Netze (siehe 2.1.3 künstliche neuronale Netze) umfassen diese Funktionalität, und finden daher in Machine Learning Modellen Anwendung.

2.1.2 Hyperparameter

Machine Learning Modelle umfassen verschiedene Hyperparameter. Diese beschreiben unter anderem wie lange das Training läuft oder wie stark sich das Modell nach einer falschen Beurteilung anpasst. Diese Hyperparameter beeinflussen das Lernverhalten des Modelles (Nyuytiymbiy, 2022), aber ihr optimaler Wert ist im Voraus nicht bekannt.

Hyperparameter können unter anderem durch den Baysian Optimization Algorithmus optimiert werden (Agnihotri & Batra, 2020)(paretos, 2021). Dieser Algorithmus versucht, den Output einer Black Box Funktion zu maximieren oder zu minimieren (Garnett, n. d., S. 15). Eine Black Box ist ein häufig komplexes System, dessen inneren Vorgänge nicht betrachtet werden („Black Box (Systemtheorie)“, 2021). Bei einer Blackbox Funktion ist folglich der Input und der Output bekannt, während die Verarbeitung des Inputs zum Output nicht betrachtet wird (siehe Abbildung 2.3).

Abbildung 2.3: Prinzip einer Black Box Funktion („Black Box“, n. d.)

Machine Learning Modelle werden häufig als Black Box Funktionen angesehen, da das Training mit hohem rechnerischen Aufwand verbunden ist, wodurch die genauen Vorgänge durch einen aussenstehenden Betrachter nicht oder nur schwer erfassbar sind (Robbins, 2017). Um ein Machine Learning Modell als eine BlackBox Funktion für den Baysian Optimization Algorithmus zu verwenden, werden die zu optimierenden Hyperparameter als Input und eine Zielvariable als Output definiert. Die Zielvarible des Outputs entspricht dabei einem Wert, der die Leistung des Modelles widerspiegelt und durch den Algorithmus maximiert werden soll. Ein Beispiel für die Zielvarible wäre die Genauigkeit des Machine Learning Modelles (siehe 2.1.1 Funktionsweise eines Machine Learning Modelles). Die inneren Vorgänge in der BlackBox Funktion entsprechen in diesem Fall einem Training des Modelles.

Der Baysian Optimization Algorithmus kann bis zu 20 Hyperparameter zuverlässig optimieren (Moriconi et al., 2020). Der Algorithmus führt die Blackbox Funktion für eine bestimmte Anzahl Iterationen mit jeweils verschiedenen Parametern durch. Die Wahl der Parameter basiert dabei auf Bayes' Theorem (Garnett, n. d., S. 7). Diejenigen Parameter, die den höchsten gefundenen Wert der Zielvariable auslösen, werden gespeichert.

2.1.3 künstliche neuronale Netze

Ein neuronales Netz ist, im biologischen Sinne, ëine beliebige Anzahl Neuronen, die miteinander Verbunden sind"(„Neuronales Netz", 2021). Ein Beispiel für ein neuronales Netz ist das menschliche Gehirn. Künstliche Neuronale Netze modellieren biologische neuronale Netze in der Form von Programmcode („Artificial Neural Network Tutorial - Javatpoint", n. d.). Diese Arbeit behandelt künstliche neuronale Netze, nicht aber biologische. Somit handelt es sich bei jedem erwähnten neuronalen Netz, um ein künstliches neuronales Netz.

Der Grundbaustein eines neuronalen Netzes ist das Neuron. Im Modell stellt das Neuron ein Objekt dar, das eine beliebiege Anzahl Inputs, aber nur einen Output hat (siehe Abbildung 2.4) (Pramoditha, 2021). Input und Output sind hierbei rationale Zahlen. Die Ausgabe des Neurons ist im einfachsten Modell, dem Perzeptron Neuron, grundsätzlich entweder 0 oder 1. Die Ausgabe ist 1, wenn die Summe der Eingaben einen vorgegebenen Wert, den *Threshold*, überschreitet. Ansonsten ist die Ausgabe gleich 0. Jede Eingabe hat ein *Gewicht*, das einer rationalen Zahl entspricht. Vor der Addition der Inputs wird jeder Input mit seinem Gewicht multipliziert. Die Grösse des Gewichts bestimmt somit den Einfluss der zugehörigen Eingabe auf die Ausgabe des Neurons. (Nielsen, 2015)(Simplilearn, 2021)

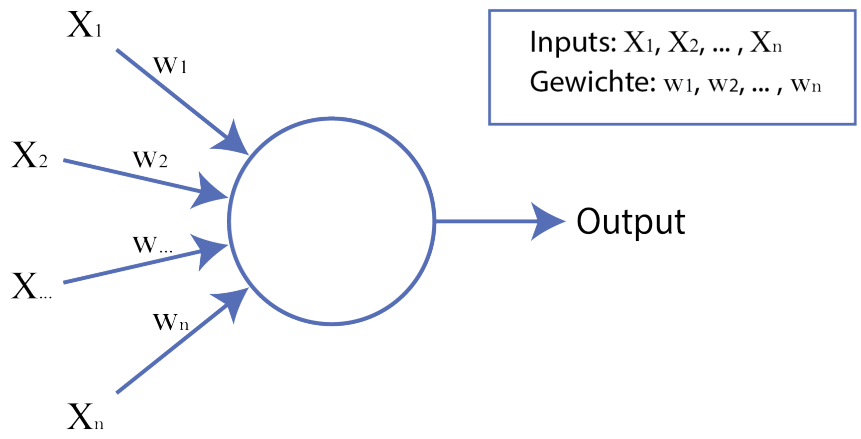

Abbildung 2.4: Perzeptron Neuron (eigene Abbildung)

Neuronale Netze in Machine Learning Modellen verwenden kompliziertere Neuronen als das Perzeptron Neuron, wie zum Beispiel das Sigmoid-Neuron. Die Neuronen unterscheiden sich in ihrer Activation Function und somit im Verhalten ihres Outputs (Pragati Baheti, 2022). So nimmt der Output

im Sigmoid Neuron beispielsweise auch Werte zwischen 0 und 1 an in einem stetigen Übergang zwischen den beiden Grenzen (siehe Abbildung 2.5) (Kumar, 2019).

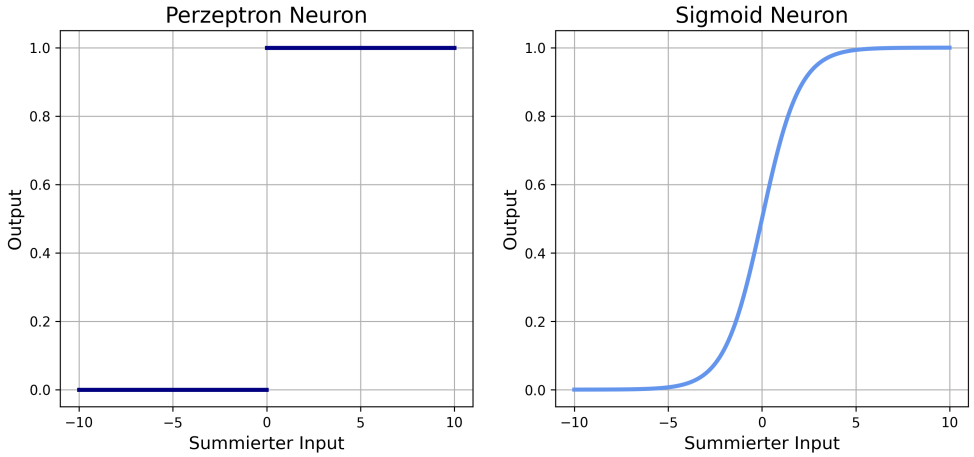

Abbildung 2.5: Vergleich des Outputs eines Perzeptron Neurons und eines Sigmoid Neurons (eigene Abbildung)

Neuronale Netze sind Verbindungen dieser Neuronen. Dabei dient der Output eines Neurons als Input in ein anderes Neuron. Der Output eines Neurons kann gleich für mehrere Neuronen ein Input sein. Die Neuronen sind in *Layers* geordnet (siehe autoreflayers). Neuronale Netze haben mindestens einen *Input Layer* und einen *Output Layer* (Nielsen, 2015)(Ognjanovski, 2020). Der Input Layer umfasst die Daten, zu dem das neuronale Netz eine Beurteilung liefern soll. Im Beispielproblem (siehe 2.1 Machine Learning) bestände die Eingabe-Ebene aus 28×28 Neuronen, wobei jedes Neuron die Graustufe (durch einen Wert von 0 bis 255) eines Pixels im Bild beschreibt. Der Input ist in diesem Fall zweidimensional. Die Dimensionen sind allerdings flexibel. Die Output Layer besteht im Beispiel aus 10 Neuronen, wobei jedes Neuron einer Beurteilung entspricht (das fünfte Neuron beschreibt zum Beispiel die Ziffer Fünf als Beurteilung). Dasjenige Neuron mit dem höchsten Output entspricht der Beurteilung des neuronalen Netzes. (siehe Abbildung 2.6).

Zwischen dem Input Layer und dem Output Layer kann es weitere *Hidden Layers* geben (Malik, 2019). Es gibt verschiedene Arten von Hidden Layers, die verschidene Funktionen haben. Zwei der meist verwendeten Layers sind Fully Connected (Dense) Layers und Convolutional Layers (Unzueta, 2022). In Fully Connected Layers dient jedes Neuron als Input für jedes Neuron in der nächsten Layer. In Convolutional Layers trifft das nicht zu (siehe Abbildung 2.7). Die Funktion von Convolutional Layers beinhaltet es, wichtige Merkmale aus dem Input hervorzuheben (Deshpande, n. d.).

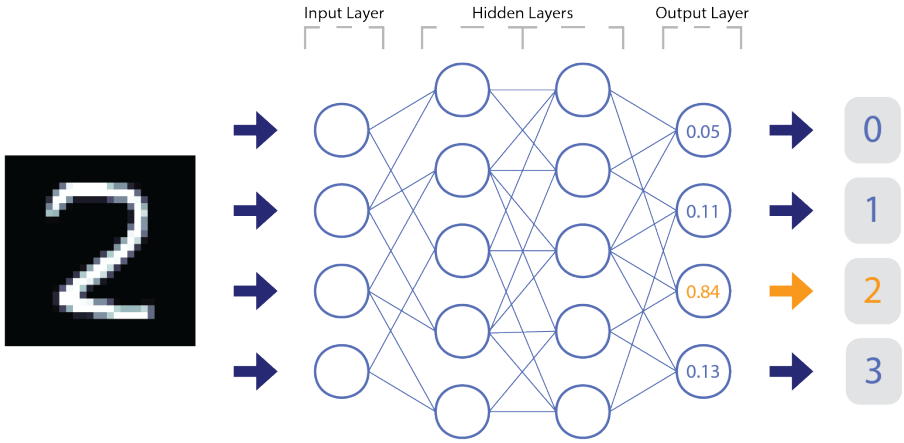

Abbildung 2.6: Neuronales Netz mit beschrifteten Layers (eigene Abbildung)

Concatenation Layers (Jayawardana, 2021) sind eine weitere Form von hidden Layers, die zwei verschiedene Layers als Input haben und diese somit verbinden. Machine Learning Modelle werden ab mehr als einer Hidden Layer als Deep Learning Modelle bezeichnet (Jan-Dirk Kranz, 2019).

Ein Machine Learning Modell passt während dem Training (siehe 2.1.1 Funktionsweise eines Machine Learning Modelles) einzelne Gewichte im neuronalen Netz an, in der Hoffnung, dass die Genauigkeit der Beurteilung mit den angepassten Gewichten grösser ist. Die genaue Anpassung erfolgt in den meisten Machine Learning Modellen durch den Backpropagation Algorithmus (Ognjanovski, 2020)(David E. Rumelhart et al., n. d.).

2.2 Reinforcement Learning

Reinforcement Learning bedeutet Lernen durch Interaktion mit einer Umgebung. (Osiński & Budek, 2018). Genauer gesagt soll ein Machine Learning Modell durch Rückmeldungen und Beobachtungen aus einer Umgebung ein bestimmtes Verhalten erlernen.

Reinforcement Learning Modelle führen somit die Umgebung ein. Anders als bei Supervised Learning und Unsupervised Learning (siehe 2.1 Machine Learning) sind die Daten, aus denen das Modell lernen soll, im Voraus nicht bekannt. Reinforcement Learning Modelle trainieren somit nicht auf der Grundlage eines Datensets. Das liegt in der Natur der Umgebung, die häufig zu viele verschiedene Zustände einnehmen kann, als dass diese in einem Datenset gesammelt werden könnten. Ein Machine Learning Modell kann trotzdem aus einer Umgebung lernen, indem es selbst mit dieser interagiert und dadurch Erfahrung sammelt. (Piyush Verma & Stelios Diamantidis, 2021)

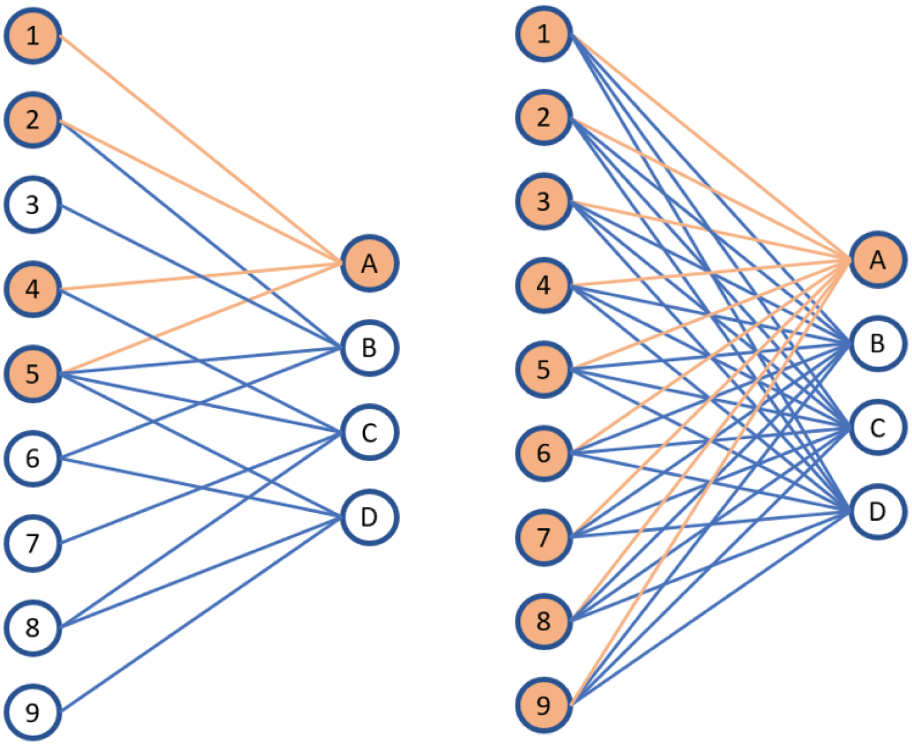

Abbildung 2.7: Vergleich zwischen Convolutional Layers (links) und Fully Connected Layers (rechts) (Unzueta, 2022)

Als Beispiel kann die echte Welt als eine Umgebung angesehen werden. Der Mensch wäre in diesem Fall das Reinforcement Learning Modell. Der Mensch lernt die Eigenschaften seiner Umgebung durch Interaktionen mit dieser kennen. Beispielsweise lernt ein Mensch die Schwerkraft durch das Hinfallen kennen. Durch diese Erfahrungen kann der Mensch ein gewisses Verhalten, zum Beispiel das Laufen, erlernen. Reinforcement Learning Modelle imitieren dieses Lernverhalten. So verwendet die Robotik häufig Reinforcement Learning, um einen Roboter laufen zu lassen. Die Umgebung, mit der das Reinforcement Learning Modell lernt, ist dabei häufig nicht echt, sondern simuliert.

2.2.1 Aufbau und Funktionsweise

Dieser Abschnitt umfasst eine genauere Erklärung eines Reinforcement Learning Modelles, in diesem Fall Deep Q-Learning, unter der Verwendung der korrekten Fachbegriffe.

Ein Reinforcement Learning Modell umfasst eine *Umgebung* und einen *Agent*. Der Agent ist dasjenige Element in der Umgebung, welches mit dieser interagiert und daraus lernt (Sutton & Barto, 2014, S. 53). Die Umgebung

verändert sich in Zeitschritten, genannt *Steps*. In jedem Step führt der Agent eine *Action* aus, die die Umgebung beeinflusst. Die Entscheidung, welche Action der Agent ausführt, basiert auf einer *Observation* (Mnih et al., 2013, S. 2) der Umgebung. Die Observation umfasst alle Daten der Umgebung, die für die Entscheidung des Agents relevant sind. Der Agent trifft seine Entscheidung auf der Basis eines neuronalen Netzes (siehe 2.1.3 künstliche neuronale Netze). Der Input in dieses neuronale Netz ist die Observation der Umgebung und der Output beschreibt die Action, die der Agent ausführt. Jedes Neuron des Outputs beschreibt eine spezifische Action des Agents. Der Agent kann somit nur eine feste Anzahl Actions ausführen. Alle Actions zusammen werden *Action-Space* (Sutton & Barto, 2014, S. 67) genannt. Jede Action im Action-Space besitzt einen *Q-Value*, der dem Output des zugehörigen Neurons entspricht. (siehe Abbildung 2.8) (Wang, 2021) Die schlussendliche Entscheidung, welche Action ausgeführt wird, basiert auf der *Epsilon-Greedy* Strategie (Sutton & Barto, 2014, S. 34). Diese Strategie sieht vor, dass die Entscheidung mit einer Wahrscheinlichkeit von ε auf eine zufällige Action fällt. Ansonsten fällt die Entscheidung auf diejenige Action mit dem höchsten Q-Value. Der Agent erkundet die Umgebung durch die zufälligen Actions, die er teilweise wählt. Der Agent wählt Actions, die er ansonsten nie wählen würde, und trifft möglicherweise zufällig auf bessere Optionen für zukünftige Steps (Rajendra Koppula, n. d.).

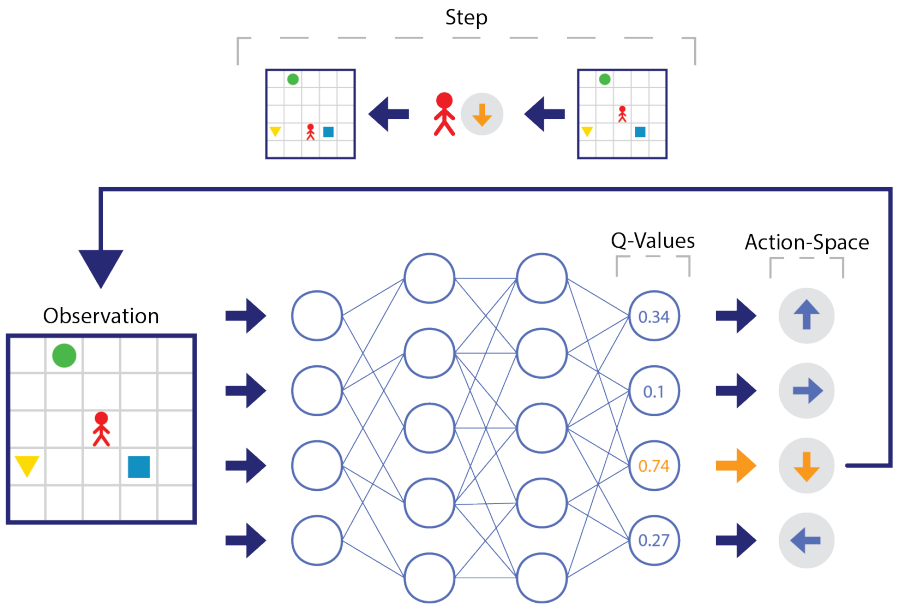

Abbildung 2.8: Funktionsweise eines Reinforcement Learning Modelles (eigene Abbildung)

Die Umgebung und somit auch der Agent werden durch die Actions des

Agenten beeinflusst. Dieser Einfluss wird durch die *Reward-Function* gemessen. Die Reward-Function gibt eine rationale Zahl, den *Reward* aus (Sutton & Barto, 2014, S. 75). Umso grösser der Reward, desto positiver ist der Effekt auf die Umgebung und umgekehrt. Ein positiver Einfluss auf die Umgebung durch eine Action ist so definiert, dass der Agent durch die Action das gewünschte Verhalten vorzeigt. Die Reward-Function definiert, welches Verhalten welchen Reward erzielt. Der Q-Value der gewählten Action wird mit dem Reward (und dem maximalen Q-Value aus den nächsten möglichen Actions) addiert. Diese Formel nennt sich Bellman-Gleichung (Mnih et al., 2013, S. 3). Der neue Q-Value hat somit einen kleineren Wert, wenn der Reward negativ ist, und einen grösseren Wert, wenn der Reward positiv ist. Die das neuronalen Netz wird daraufhin so trainiert, dass der Output für das Neuron, dessen Action ausgeführt wurde, näher am neu berechneten, besseren Q-Value ist (siehe Abbildung 2.9). Der schlussendliche Effekt ist, dass Actions, die einen positiven Reward auslösen, wahrscheinlicher gewählt werden, und umgekehrt Actions, die einen negativen Rewards auslösen, unwahrscheinlicher gewählt werden. Der Agent versucht insgesamt durch seine Actions einen möglichst hohen akkumulierten Reward zu erzielen (Sutton & Barto, 2014, S. 57). Der akkumulierte Reward entsprich der Summe der Rewards aus jedem Step in einer Episode.

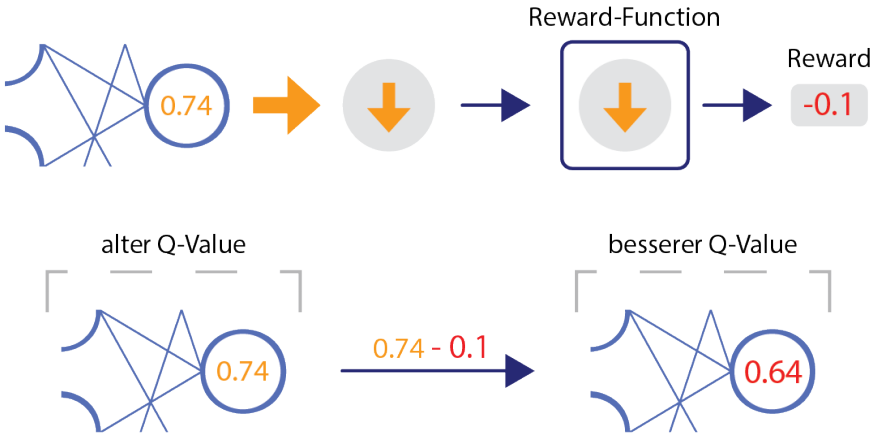

Abbildung 2.9: Funktionsweise einer Reward-Function (eigene Abbildung)

Das Training läuft in *Episodes* (Sutton & Barto, 2014, S. 14). Eine Episode umfasst eine gewisse Anzahl Steps und am Anfang jeder Episode wird die Umgebung in einen Startzustand zurückgesetzt. Die Resultate eines Steps werden in dem *Replay-Buffer* gespeichert. Dazu gehören die Observation der Umgebung, die jeweiligen Actions und der jeweilige Reward. Der Replay-

Buffer enthält Speicherplatz für eine bestimmte Anzahl Steps. Während dem Training werden zufällige Steps aus dem Replay-Buffer gewählt, auf die das neuronale Netz trainiert. Das neurnale Netz trainiert also auf Daten aus der Vergangenheit der Umgebung und des Agents. Diese Strategie nennt sich Experience Replay (Mnih et al., 2013, S. 5). Ausserdem trainiert das neuronale Netz jeweils mit einem *Batch* an Steps, also mit einer gewissen Anzahl an Steps gleichzeitig. Der Replay-Buffer und der Batch sichern zu, dass das neuronale Netz mit einer grossen Vielfalt an Steps trainiert, was das Lernverhalten stabiler macht als ein chronologisches Training auf einzelne Steps (PhD, 2021).

2.3 Verwandte Arbeiten und Themen

Das Nachzeichnen von Strichbildern ist ein Teilbereich von dem Zeichnen allgemein. Es gibt verschiedene Ansätze, um einen Computer zeichnen zu lassen. Ein häufiger Ansatz ist *Stroke-Based Rendering* (Hertzmann, 2002). Stroke-Based Rendering ist das Zeichnen von Bilder durch das Platzieren von Elementen wie Strichen. Beispiele für Arbeiten in diesem Bereich sind Strokenet (Zheng et al., 2019) und "Learning to Paint With Model-based Deep Reinforcement Learning" (Huang et al., 2019) Andere Ansätze simulieren die Führung eines Stiftes. (siehe Abbildung 2.10) Ein Beispiel dafür ist das Programm Doodle-SDQ (Zhou et al., 2018). Doodle-SDQ beschäftigt sich auch spezifischer mit dem Nachzeichnen von Strichbildern und wird deswegen im nächsten Abschnitt weiter behandelt.

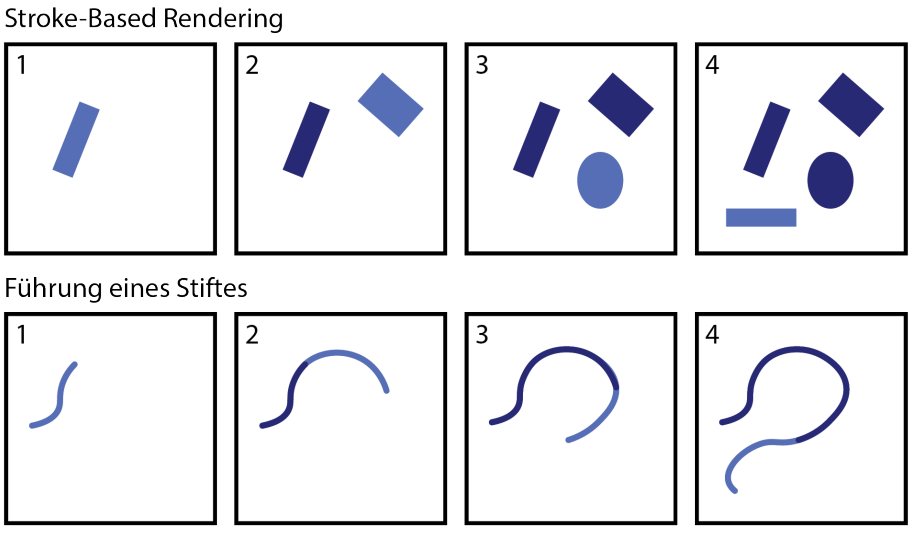

Abbildung 2.10: Vergleich zwischen Stroke-Based Rendering und dem Führen eines Stiftes (eigene Abbildung)

2.3.1 Doodle-SDQ

Doodle-SDQ ist ein Computerprogramm, das durch ein Reinforcement Learning Modell, spezifischer Deep Q-Learning (siehe 2.2.1 Aufbau und Funktionsweise), erlernt, Strichbilder aus dem Google QuickDraw Datenset („Quick, Draw! Image Recognition“, 2022) nachzuzeichnen. Nachfolgend sind die Aspekte von Doodle-SDQ beschrieben, die für diese Arbeit relevant sind.

Die QuickDraw Bilder, die das Programm nachzeichnen soll, sind zu einer einheitliche Grösse von 84×84 Pixeln verarbeiteitet (Zhou et al., 2018, S. 7). Der Agent kann sich auf einer leeren Zeichenfläche von der selben Grösse bewegen und zeichnen. Die Umgebung umfasst diese Zeichenfläche, den Agent und das abzuzeichnende Bild.

Der Agent kann sich durch eine Action in jedem Step auf einen beliebigen Pixel in einem 11×11 Feld, in dessen Zentrum er ist, bewegen. Der Agent kann ausserdem jede dieser Bewegungen im zeichnenden Zustand oder im nicht zeichnenden Zustand machen. Der Action-Space hat somit insgesamt eine Grösse von $2 \cdot 11 \cdot 11 = 242$ Actions (Zhou et al., 2018, S. 5). Im zeichnenden Zustand wird ein Strich auf der Zeichenfläche zwischen der alten und der neuen Position des Agenten gezeichnet. Der Agent begeht 100 Steps pro Episode. Eine neue Episode entspricht dabei einem neuen Bild, das abgezeichnet werden soll.

Die Obervation der Umgebung, und somit der Input in das neuronale Netz (siehe 2.2.1 Aufbau und Funktionsweise), ist in zwei Teile gegliedert: den Global Stream und den Local Stream. Der Global Stream hat eine Form von $28 \times 28 \times 4$. Der Input ist somit dreidimensional. Die Form kann als 4 aufeindandergestapelte Bilder angesehen werden, die jeweils eine Grösse von 28×28 Pixeln haben. Dabei beschreibt eine relle Zahl den Wert von jedem Pixel in einem Bild. Das erste Bild im global Stream ist die Vorlage, die abgezeichnet werden soll. Das zweite Bild ist die Zeichenfläche im aktuellen Zustand. Das dritte Bild beschreibt die Position des Agents durch seine relative Entfernung zu jedem Punkt auf der Zeichenfläche. Das vierte Bild beschreibt, ob der Agent im zeichnenden Zustand ist oder nicht. Wenn alle Pixel dieses letzten Bildes den Wert 1 haben, ist der Agent im zeichnenden Zustand. Wenn umgekehrt alle Pixel den Wert 0 haben, ist der Agent nicht im zeichnenden Zustand. Der Local Stream hat eine Form von $11 \times 11 \times 2$. Er ist somit auch dreidimensional und beschreibt zwei gestapelte Bilder. Das erste Bild umfasst die Vorlage in dem 11×11 Bereich (bezeichnet als Local image patch (Zhou et al., 2018, S. 5)), in dem sich der Agent in einem Schritt bewegen kann. Das zweite Bild beschreibt den selben Bereich von der Zeichenfläche (Zhou et al., 2018, S. 4 ff.). Der global Stream und der Local Stream werden durch eine Concatenation Layer (siehe 2.1.3 künstliche neuronale Netze) zusammengeführt.

2.4 Git und GitHub

Git und Github sind weit verbreitete Hilfsmittel für Software Entwickler. Git ist ein Programm, während GitHub ein Service ist, der dieses Programm in der Cloud zugänglich macht. GitHub hat zusätzliche Funktionen, die die Zusammenarbeit zwischen mehreren Entwicklern erleichtern. Die genaue Funktion und das Zusammenspiel dieser beiden Hilfsmittel wird nachfolgend erläutert.

2.4.1 Git

Git erkennt Veränderungen im Code eines Projektes und speichert diese Veränderungen in einer neuen Version ab. Die einzelnen Versionen des Projektes bleiben dabei zu jedem Zeitpunkt abrufbar. Dieses Konzept nennt sich Version Control (Atlassian, n. d. a). Das Programm wurde 2005 von Linus Torvald entwickelt („Git“, 2021). Versionen des Projektes werden manuell durch einen Commit gespeichert. Es wird empfohlen, nur jeweils ein bestimmtes Problem oder eine bestimmte Funktion pro Commit anzugehen (**noauthor˙5˙nodate**). Für grössere Funktionen oder Probleme kann ein Branch erstellt werden. Ein Branch ermöglicht eine abgekapselte Entwicklung eines Projektes (Guillermo Brachetta, 2022). Zum Beispiel kann das Projekt in mehreren Branches gleichzeitig und unabhängig von einander entwickelt werden (Guillermo Brachetta, 2022). Eine weit verbreitete Arbeitsweise und Branch Struktur mit Git ist GitFlow („What is Git Flow — How to use Git Flow — Learn Git“, 2022)(Atlassian, n. d. a). GitFlow schlägt grundsätzlich einen Main Branch, einen Develop Branch und verschiedene Feature Branches vor (Atlassian, n. d. a). Professionelle Anwendungen von GitFlow verwenden ausserdem so genannte Release Branches und Hotfix Branches (Cameron McKenzie, 2021). Im Main Branch sind offizielle Versionen des Projektes gespeichert, Im Develop Branch wird das Projekt als ganzes entwickelt, und in jedem Feature Branch wird eine Funktionalität in das Projektes implementiert (siehe Abbildung 2.11).

2.4.2 GitHub

GitHub wurde 2008 von Chris Wanstrath, PJ Hyett, Scott Chacon und Tom Preston-Werner entwickelt („GitHub“, 2021). 2018 wurde das Unternehmen von Microsoft gekauft. GitHub ist ein Service, der Projekte, die mit Git verwalten werden, in der Cloud speichert. Dadurch kann ein Projekt überall und von beliebig vielen Personen entwickelt werden. GitHub betreibt eine Webseite, über welche der Service verwendet werden kann („GitHub“, 2021).

GitHub besitzt verschiedene Hilfsmittel, die die Zusammenarbeit von Entwicklern weiter vereinfachen. Beispiele dafür sind Issues und Project Boards. Diese Hilfsmittel ermöglichen Organisation, Strukturierung und

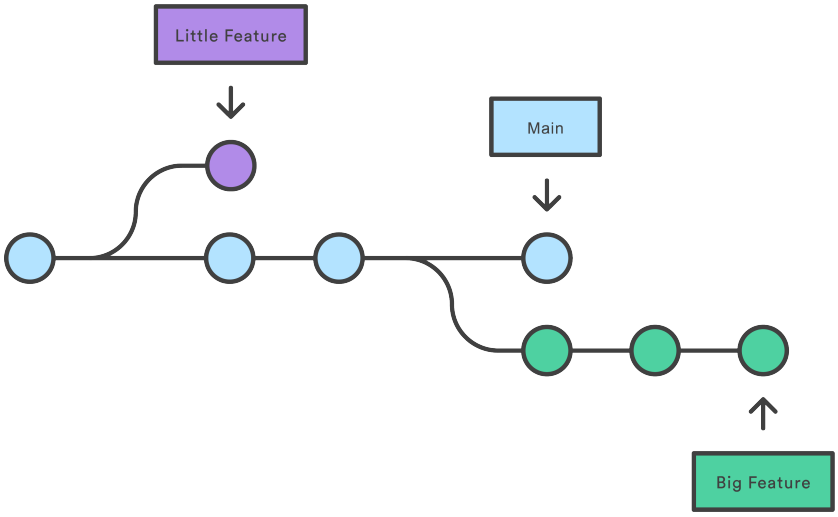

Abbildung 2.11: Branch Struktur von GitFlow (Atlassian, n. d. a)

Arbeitsteilung. Ein weiteres Hilfsmittel sind Pull Requests. Eine Pull Reqeust wird dann gestellt, wenn die Arbeit an einem Branch fertig ist. Durch Pull Requests können die Entwickler des Projektes die Funktionalität eines Branches überprüfen. Wenn ein Branch nicht die gewünschte Aufgabe erfüllt, kann die Pull Request abgelehnt werden. Erst wenn eine Pull Request angonommen wird, kann der Branch wieder in den Main Branch zurückgeführt werden (Atlassian, n. d. b).

Kapitel 3

Methode

github ist ein Die Methode dieser Untersuchung besteht darin, die in der Fragestellung beschriebene künstliche Intelligenz (KI) zu entwickeln und dessen Leistung auszuwerten. Die Diskussion dieser Resultate führt schlussendlich zu einer Antwort auf die Fragestellung. Die Entwicklung der KI besteht aus zwei Teilen. Der eine Teil umfasst die Definition der Kriterien, nach denen die Leistung der KI evaluiert wird (siehe 3.2 Evaluierung der Leistung). Der andere Teil umfasst die Entwicklung der KI (siehe 3.1 Grundprogramm), zusammen mit verschiedenen Variationen (siehe 3.3 Variationen). Die Variationen haben jeweils einen unterschiedlichen Fokus auf die definierten Kriterien. Die Auswertung (siehe 3.4 Auswertung) bezieht sich ebenfalls auf die definierten Kriterien. Die Leistung der KI wird dabei in einer Testumgebung für das Zeichnen von verschiedenen Arten von Strichbildern erfasst.

3.1 Grundprogramm

Die KI ist abhängig von den Kriterien, die dessen Leistung definieren (siehe 3.2 Evaluierung der Leistung). Mit anderen Worten trainiert die KI auf diese Kriterien. Das Ziel des Grundprogrammes ist, die allgemeine Trainingsumgebung für die KI bereitzustellen. Dieses Grundprogramm ist unabhängig von einem spezifischen Kriterium und kann stattdessen auf ein ausgewähltes Kriterium trainiert werden. Reinforcement Learning Modelle mit einer undefinierten Reward Function (siehe 2.2.1 Aufbau und Funktionsweise) stellen diese Eigenschaften bereit. Eine Reward function, basierend auf einem spezifischen Kriterium, ermöglicht das Training auf dieses Kriterium. Das Grundprogramm ist in Python unter der Verwendung des Keras Frameworks implementiert (siehe 2.1 Machine Learning).

3.1.1 Doodle-SDQ als Basis

Das Reinforcement Learning Modell des Grundprogrammes basiert auf Doodle-SDQ (siehe 2.3.1 Doodle-SDQ). Von Doodle-SDQ ist das neuronale Netz, bezogen auf die Form des Inputs, des Outputs und den Hidden Layers, grösstenteils übernommen. Die relevanten Anpassungen zwischen Doodle-SDQ und dem Grundprogramm dieser Arbeit sind nachfolgend erläutert.

Bei der Umgebung handelt es sich, wie bei Doodle-SDQ, um eine Zeichenfläche, worauf sich der Agent frei bewegen kann. Die Ziffern, die während dem Training nachgezeichnet werden sollen, stammen aus dem MNIST Datenset (siehe 2.1 Machine Learning) und haben somit eine Grösse von 28×28 Pixeln. Die Fläche, worauf sich der Agent bewegen kann, hat somit auch eine Grösse von 28×28 Pixeln. Der global Stream (siehe 2.3.1 Doodle-SDQ) des Inputs in das Neuronale Netz ändert sich bis auf die neue Grösse der Bilder nicht. Die Pixel der Bilder, wie auch die Zeichenfläche, nehmen den Wert von einem Bit an. Eine Null repräsentiert einen schwarzen (nicht gezeichneten) Pixel an dieser Stelle im Bild und eine Eins einen weissen (gezeichneten) Pixel. Die genaue Architektur des neuronalen Netzes ist im Schema Abbildung 3.1 angegeben. Jeder Block in der Abbildung repräsentiert einen Layer des neuronalen Netzes, wobei die Form des Inputs und die Form des Outputs von jedem Layer angegeben ist.

Der Local Stream, also das nahe Umfeld um den Agent schrumpft von 11×11 Pixel auf 7×7 Pixel. Somit schrumpft gleichzeitig der Action-Space (siehe 2.2.1 Aufbau und Funktionsweise) des Agenten von $2 \cdot 11 \cdot 11 = 242$ Actions auf $2 \cdot 7 \cdot 7 = 98$ Actions. Das bedeutet für den Agent, dass er sich pro Step um maximal drei Pixel von seiner Position wegbewegen kann. Diese Bewegung kann der Agent entweder zeichnend oder nicht zeichnend ausführen (siehe Abbildung 3.2).

Falls der Agent die Action zeichnend ausführt, zieht das Programm einen Strich zwischen der alten und der neuen Position. Mit anderen Worten werden alle Pixel der Zeichenfläche zwischen den beiden Positionen weiss. Der Strich hat eine festgelegte Breite von 3 Pixeln. Am Anfang jeder Episode, also bei jeder neuen Ziffer, die gezeichnet werden soll, startet der Agent in einer zufälligen Position im nicht zeichnenden Zustand. Am Anfang jeder Episode ist die Zeichenfläche leer, also vollkommen Schwarz.

Actions des Agents, die ihn über die vorgegebene Zeichenfläche hinaus positionieren würden, sind nicht zulässig. Diese Actions können vom Agent nicht gewählt werden und ihr optimaler Q-Value ist in jedem Fall 0. Das hat zur Folge, dass nach dem Training die allermeisten unzulässigen Actions einen Q-Value nahe oder gleich 0 haben. Das senkt die Wahrscheinlichkeit, dass der Agent versucht, eine unzulässige Action auszuführen.

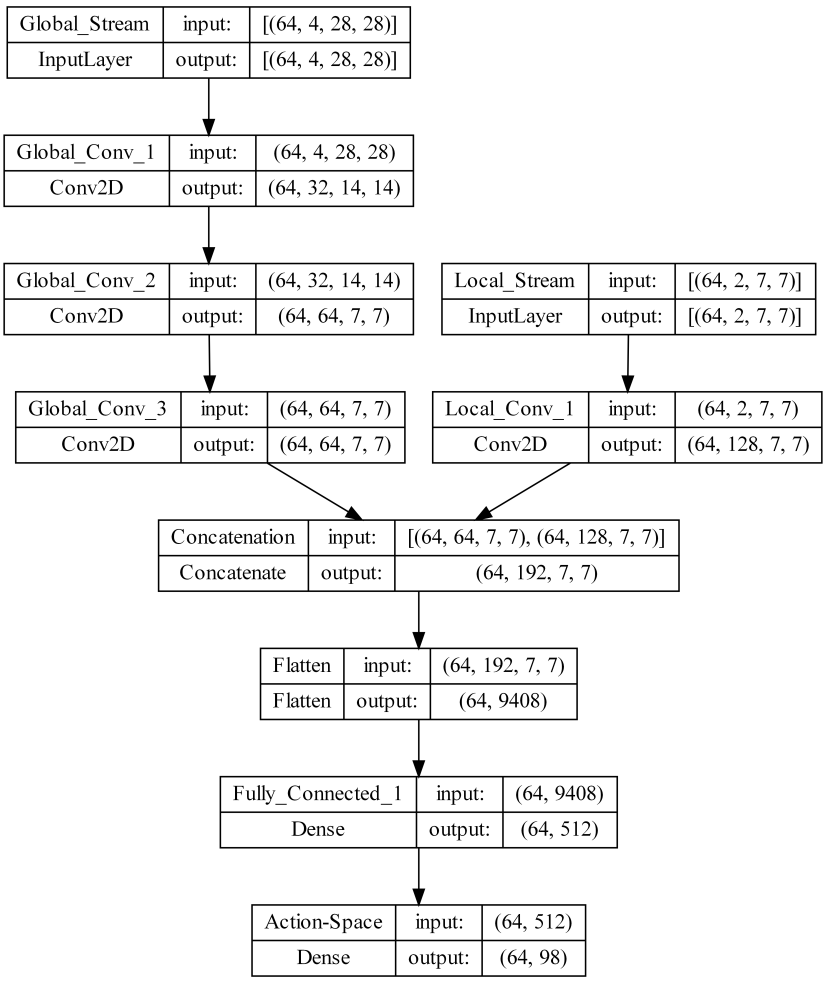

Abbildung 3.1: Architektur des neuronalen Netzes im Grundprogramm (eigene Abbildung, mit Keras erstellt)

3.1.2 Präparierung der Daten und Optimierung

Die Trainingsdaten bestehen aus 36′000 Bildern von handgeschriebenen Ziffern aus dem MNIST Datenset (siehe 2.1 Machine Learning). Die restlichen Bilder des MNIST Datensets machen die Testdaten aus. Die Bilder im Datenset sind als Bitmap dargestellt, wobei jedes Element (jeder Pixel) einen Wert zwischen 0 und 255 annimmt. Die Zahl repräsentiert eine Graustufe, wobei 0 Schwarz ist und 255 Weiss. Diese Graustufen werden entfernt. Jeder Pixel mit einem Wert über 0 übernimmt den Wert 1, wodurch die Bilder nur noch aus Einsen und Nullen bestehen. Dabei ist 0 Schwarz und 1 Weiss (siehe Abbildung 3.3). So stimmen die Bilder mit den Zeichnungen, die der Agent produzieren kann, überein.

Das Grundprogramm trainiert mit 4000 Bildern, von denen jede Ziffer

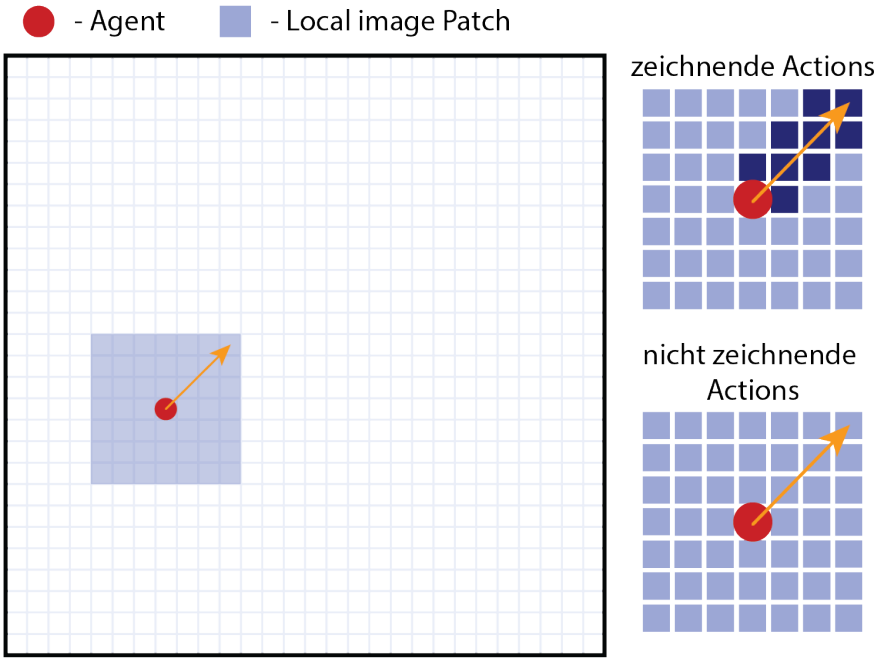

Abbildung 3.2: Action-Space im Grundprogramm

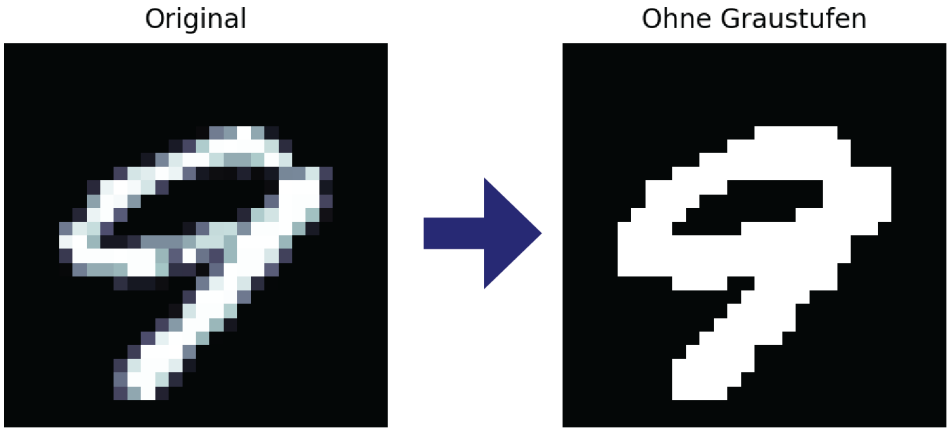

Abbildung 3.3: Entfernung der Graustufen im MNIST Datenset (eigene Abbildung)

400 Bilder ausmacht. Die restlichen Bilder in den Trainingsdaten sind für mögliche Erweiterungen aufgehoben. Der Agent zeichnet jedes der 4000 Bilder ein Mal und trainiert somit für 4000 Episodes. Der Agent macht 64 Steps pro Episode. Er kann sich also pro Zeichnung 64 Mal bewegen. Das neuronale Netz passt sich in jedem vierten Step an, mit einem Batch von 64 zufällig ausgewählten Steps aus dem Replay Buffer.

Die Hyperparameter des Grundprogrammes, wie auch die der Variationen (siehe 3.3 Variationen) sind durch den Bayesian Optimization Algorithmus optimiert (siehe 2.1.2 Hyperparameter). Die Implementierung des Algorithmus in Python stammt von (Fernando Nogueira, 2014). Der Algorithmus ändert sich für verschiedene Variationen der KI nicht und ist somit Teil des Grundprogrammes, wobei er zu der optimalen Leistung der KI beiträgt.

Mit jeder Iteration des Baysian Optimization Algorithmus trainiert das Reinforcement Learning Modell für eine vom Algorithmus selbst bestimmte Anzahl Episodes. Die Zielvariable, die durch den Baysian Optimization Algorithmus maximiert werden soll, wird am Ende jeder Iteration des Trainings in der Testumgebung berechnet (siehe 3.4.1 Testumgebung). Auf welchem Kriterium die Zielvariable basiert, ist frei wählbar.

3.2 Evaluierung der Leistung

In diesem Unterkapitel sind die Kriterien definiert, die die Leistung der künstlichen Intelligenz evaluieren. Mit anderen Worten beschreiben die Kriterien, wie gut die KI nachzeichnet. Für eine präzise und objektive Evaluierung sind alle Kriterien durch einen Zahlenwert definiert. Dieser Zahlenwert geht direkt aus Berechnungen vom Computerprogramm hervor. Die Kriterien und ihre jeweilige Berechnung sind nachfolgend beschrieben.

3.2.1 Erkennbarkeit

Das Kriterium der Erkennbarkeit beschreibt, ob in der Vorlage das gleiche Motiv wie in der Zeichnung der künstlichen Intelligenz erkannt wird. Wenn Beispielsweise in beiden Fällen eine Fünf erkannt wird, hat das Kriterium den Wert 1. Wird in der Vorlage eine Fünf erkannt, aber in der Zeichnung eine Vier, hat das Kriterium den Wert 0

Welches Motiv in der Zeichnung erkannt wird, ist durch eine zweite künstliche Intelligenz bestimmt (siehe 2.1.1 Funktionsweise eines Machine Learning Modelles). Diese zweite KI beurteilt ein Motiv nur als erkannt, wenn das zugehörige Neuron im Output des neuronalen Netzen einen Wert von über 0.75 hat. Das entspricht mit einer hohen Wahrscheinlichkeit der korrekten Beurteilung.

Um die verschiedenen Arten von Strichbildern, die die KI zeichnen soll, zu erkennen, existieren vortrainierte Machine Learning Modelle. Die in dieser Arbeit implementierten vortrainierten Modelle sind in der Tabelle Tabelle 3.1 ersichtlich. Diese Modelle sind mit den selben Daten trainiert, die in der Testumgebung (siehe 3.4.1 Testumgebung) als Vorlage zum Abzeichnen dienen.

Art	Entwickler	Trainiert mit
Zahlen	(Mazzia et al., 2022)	MNIST
Buchstaben	(Mor, 2022)	EMNIST Letters
Strichbilder von Objekten	(Lâm (Linus), 2022)	Auswahl aus QuickDraw

Tabelle 3.1: Vortrainierte Modelle

Dieses Kriterium ist in der Fragestellung (siehe 1 Einleitung) angedeutet. Die Antwort auf die Frage fällt positiv aus, wenn die KI dieses Kriterium der Erkennbarkeit konsequet erfüllt. Neben der Erkennbarkeit existieren weitere Kriterien, die andere Aspekte der Leistung der künstlichen Intelligenz betreffen.

3.2.2 Prozentuale Übereinstimmung

Dieses Kriterium ist durch die prozentuale Übereinstimmung der weissen (gezeichneten) Pixel zwischen der Vorlage und der Zeichnung der künstlichen Intelligenz definiert. Der Wert K dieses Kriteriums zu einem bestimmten Step t berechnet sich aus folgender Formel:

$$K(t) = \frac{G(t)}{G_{\max}}$$

$G_{\max}$ entspricht der Anzahl aller weissen Pixeln in der Vorlage. $G(t)$ entspricht der Anzahl der weissen Pixel, die zwischen der Vorlage und der Zeichenfläche übereinstimmen. Wenn der gleiche Pixel (am gleichen Ort) in der Vorlage und in der Zeichenfläche weiss ist, erhöht sich diese Anzahl um Eins. Wenn in der Zeichenfläche ein weisser Pixel gezeichnet ist, der in der Vorlage schwarz ist, sinkt die Anzahl um Eins. $G(t)$ und somit auch $K(t)$ können dadurch auch negative Werte annehmen. Der maximale Wert von $K(t)$ ist 1, was einer Genauigkeit von 100% entspricht (siehe Abbildung 3.4).

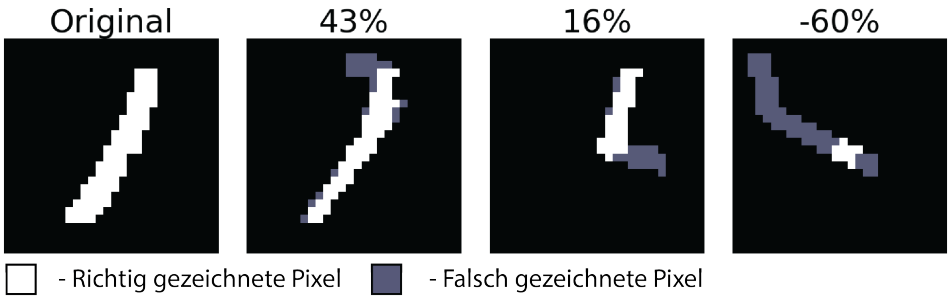

Abbildung 3.4: Drei Beispiele für den Wert des Kriteriums der Übereinstimmung (eigene Abbildung)

3.2.3 Geschwindigkeit

Dieses Kriterium beschreibt, wie schnell die Zeichnung der KI fertig ist. Der Wert dieses Kriteriums entspricht der Anzahl Steps bis zur Fertigstellung der Zeichnung. Eine kleinere Anzahl Steps entspricht einer schnelleren Fertigstellung der Zeichnung und somit einer besseren Leistung nach diesem Kriterium.

Eine Zeichnung gilt als fertig, wenn die prozentuale Übereinstimmung (siehe 3.2.2 Prozentuale Übereinstimmung) mindestens 70% beträgt und die Zahl der Definition entsprechend erkannt wird (siehe 3.2.1 Erkennbarkeit). Wenn die Zeichnung bis zum Ende der Episode die Bedingungen einer fertigen Zeichnung nicht erfüllt, hat dieses Kriterium den Wert 64. Das entspricht der maximalen Anzahl Steps, die von der KI pro Zeichnung begangen werden.

3.3 Variationen

Dieses Kapitel beschreibt verschiedene Variationen ausgehend vom Grundprogramm (siehe 3.1 Grundprogramm). Bei einigen dieser Variationen handelt es sich um konkrete Implementierungen der definierten Kriterien in die Reward Function (siehe 2.2.1 Aufbau und Funktionsweise). Die Reward Function kann dabei auch auf mehreren Kriterien basieren. Der Unterschied zwischen den Variationen liegt im Fokus auf die Kriterien. Einige Variationen sind untereinander kombinierbar. Andere Variationen führen Strukturelle Änderungen an der KI ein, die über die Reward Function hinaus gehen. Das Ziel der strukturellen Änderungen ist eine grundsätzliche Verbesserung, oder zumindest eine Anpassung des Verhaltens und der Leistung der künstlichen Intelligenz.

3.3.1 Basis Reward-Function

Die Basis Reward Function ist die einfachste Erweiterung des Grundprogrammes (siehe 3.1 Grundprogramm) zu einer funktionierenden künstlichen Intelligenz. Diese Reward Function implementiert das Kriterium der prozentualen Übereinstimmung (siehe 3.2.2 Prozentuale Übereinstimmung). Der Reward für eine Action berechnet sich aus der Differenz zwischen der prozentualen Übereinstimmung vor dem Ausführen der Action, und der prozentualen Übereinstimmung nach dem Ausführen der Action (also $K(t-1)$ und $K(t)$). Somit wird der Reward R zum Step t durch folgende Formel berechnet.

$$R(t) = K(t) - K(t-1)$$

Der Reward eines Steps entspricht somit nicht der gesamten prozentualen Übereinstimmung zu diesem Step. Stattdessen Entspricht der Reward der

Veränderung der prozentualen Übereinstimmung, ausgelöst durch die Action zu diesem Step. Der addierte Reward aller Steps enstspricht dem absoluten Wert der prozentualen Übereinstimmung.

3.3.2 Training auf Geschwindigkeit

Der numerische Wert für die Zeit bis zur Fertigstellung der Zeichnung (siehe 3.2.3 Geschwindigkeit) kann in die Reward-Function integriert werden. Dadurch trainiert die künstliche Intelligenz auf eine minimale Zeit bis zur Fertigstellung. Die Variation verwendet grundsätzlich die Basis Reward-Function (siehe 3.3.1 Basis Reward-Function). Die Anpassung davon sieht folgendermassen aus: Am Ende jeder Zeichnung wird der Reward jedes Steps mit einem Faktor f multipliziert. Dieser Faktor berechnet sich aus folgender Formel:

$$f = 2 - \frac{S}{S_{\max}}$$

$S_{\max}$ entspricht der Anzahl Steps, die der Agent pro Zeichnung (Episode) begeht (siehe 3.1.2 Präparierung der Daten und Optimierung). S entspricht der Anzahl Steps bis zur Fertigstellung der Zeichnung. Der Faktor nimmt einen Wert zwischen 1 und 2 an. Ein grösserer Faktor f entspricht einer schnellen Fertigstellung und deswegen einem hohen Reward. Wenn der Agent die Zeichnung bis zum Ende einer Episode nicht fertigstellt, ist $f = 1$ (siehe 3.2.3 Geschwindigkeit). In diesem Fall unterscheidet sich die Reward-Function nicht von der Basis Reward-Function. Wenn die Zeichnung früher fertiggestellt wird, zeichnet der Agent trotzdem $S_{\max}$ Steps. Das verhindert eine ungleichmässige Verteilung zwischen verschiedenen Episodes im Replay-Buffer (siehe 2.2.1 Aufbau und Funktionsweise). In diesem Fall wird S nur in dem Step gespeichert, in dem die Zeichnung zum ersten Mal die Bedingung einer Fertigstellung erfüllt.

Der Fokus des Trainings auf eine maximale Geschwindigkeit erfährt einen weiteren anstieg, durch eine Anpassung der Bedingung für eine fertige Zeichnung während dem Training. Die minimale prozentuale Übereinstimmung einer fertigen Zeichnung ist als 75% definiert. Zu beginn des Trainings wird dieser Wert auf 25% heruntergesetzt, und über das Training hinweg linear bis auf 75% erhöht. Dadurch löst die Reward Function bei einer unfertigen Zeichnung bereits positive Rewards für die Geschwindigkeit aus.

3.3.3 Training auf Erkennbarkeit

Das Kriterium der Erkennbarkeit kann, anders als die anderen Kriterien, nur teilweise in die Reward Function integriert werden. Das Kriterium strebt eine Erkennbarkeit, unabhängig von der Art der Strichbilder, an (siehe 3.2.1 Erkennbarkeit). Die künstliche Intelligenz trainiert allerdings nur auf das

Nachzeichnen von Ziffern. Aus diesem Grund trainiert diese Variation nur auf die Erkennbarkeit von Ziffern, und lässt die anderen Arten von Strichbildern aussen vor.

Die Reward-Function (siehe 2.2.1 Aufbau und Funktionsweise) dieser Variation beinhaltet eine zweite KI, die handgeschriebene Ziffern erkennt (siehe Tabelle 3.1). Diese zweite KI beurteilt in jedem Step, welche Ziffern sie in der Vorlage und in der aktuellen Zeichnung erkennt. Wenn die erkannte Zahl in der Vorlage und der Zeichnung gleich ist, erhält der Agent einen Reward von 0.1. In diesem Zustand funktioniert die Reward-Function allerdings nicht. Der Agent kann den akkumulierten Reward nicht maximieren. Zwei Ansätze gehen auf dieses Problem ein. Beide Ansätze sind Teil dieser Variation.

Der erste Ansatz schlägt vor, die zweite KI erst ab einer gewissen prozentualen Übereinstimmung (siehe 3.2.2 Prozentuale Übereinstimmung) einzusetzen. In diesem Fall löst die korrekte Erkennung erst ab einer prozentualen Übereinstimmung von 20% einen positiven Reward aus. Diese zusätzliche Bedingung ist notwendig, weil die Beurteilungen der zweiten KI teilweise für einen menschlichen Betrachter fragwürdig sind. Zum Beispiel schätzt die zweite KI eine leere Zeichenfläche mit einer hohen Wahrscheinlichkeit als eine Eins (siehe **??**) ein. Das ist ein Problem, weil dadurch der Agent einen positiven Reward für eine leere Zeichenfläche erhält. Das stört das weitere Lernverhalten, weil es die Wahrscheinlichkeit erhöht, dass die KI nicht mehr zeichnet.

Der zweite Ansatz implementiert neben der Reward-Function der Erkennbarkeit erneut die Basis Reward-Function (siehe 3.3.1 Basis Reward-Function). Die Relevanz der beiden Reward-Functions ändert sich allerdings über das Training hinweg. Die Rewards werden in jedem Step mit einem bestimmten Faktor multipliziert. Zu Beginn des Trainings ist der Faktor für den Reward der Basis Reward-Function $f_b = 1$ und der Faktor für den Reward basierend auf der Erkennbarkeit $f_e = 0$. Vom Start ausgehend sinkt f_b linear und f_e steigt linear. Ab einem gewissen Punkt bleiben beide Faktoren stehen (siehe Abbildung 3.6). Blieben die Faktoren ab diesem Punkt nicht konstant, würde die Variation, gestützt auf Beobachtungen, an Stabilität der Leistung verlieren.

Das Zusammenspiel der beiden Reward-Functions hat den Vorteil, dass die künstliche Intelligenz zu Beginn des Trainings durch die Basis Reward-Function für kleine Erfolge positive Rewards erzielt. Die Reward-Function der Erkennbarkeit ermöglicht das nicht, da sie erst für eine korrekte Erkennung einen Reward auslöst. Eine korrekte Erkennung ist für eine untrainierte KI schwer zu erreichen. Deswegen muss die KI durch die Basis Reward-Function gewissermassen vortrainiert werden, um schlussendlich von der Reward-Function der Erkennbarkeit zu profitieren

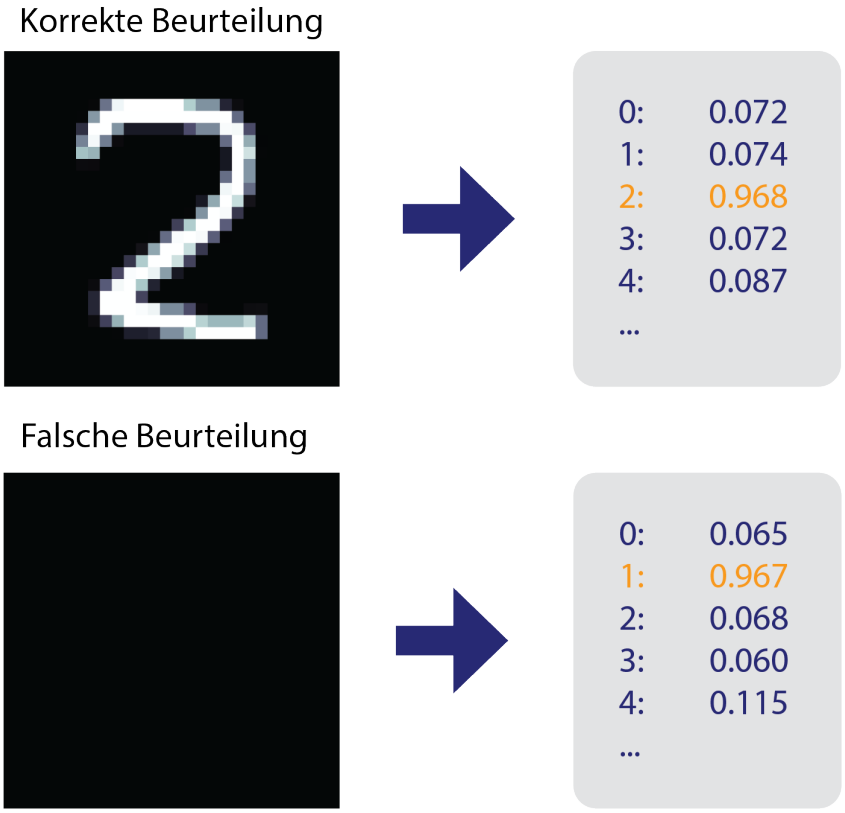

Abbildung 3.5: Beispiele einer richtigen und einer falschen Erkennung von handgeschriebenen Zahlen durch eine KI (eigene Abbildung). Die Werte sind durch einen Test der KI berechnet

3.3.4 Physikalische Umgebung

Diese Variation spezialisiert sich auf kein Kriterium. Stattdessen verändert sich die Umgebung, in der sich der Agent bewegt (siehe 2.2.1 Aufbau und Funktionsweise). Auch der Input und der Output des neuronalen Netzes sind angepasst. Durch diese Veränderungen unterscheidet sich die Variation vom Grundprogramm. Sie bleibt allerdings mit den anderen Variationen (siehe 3.3.3 Training auf Erkennbarkeit und 3.3.2 Training auf Geschwindigkeit) kompatibel, Da diese ausschliesslich die Reward-Function anpassen.

Die Variation ergänzt die Umgebung durch physikalische Simulationen. Diese physikalische Umgebung definiert die physischen Rahmenbedingungen des Zeichnens neu, mit dem Ziel, diese näher an die Realität zu bringen (siehe 5.1.1 Beantwortung der Unterfragen).

Der Agent hat neu eine Geschwindigkeit, die durch einen Vektor $\vec{v}$ dargestellt ist. Die Geschwindigkeit beschreibt, um wie viele Pixel und in welche

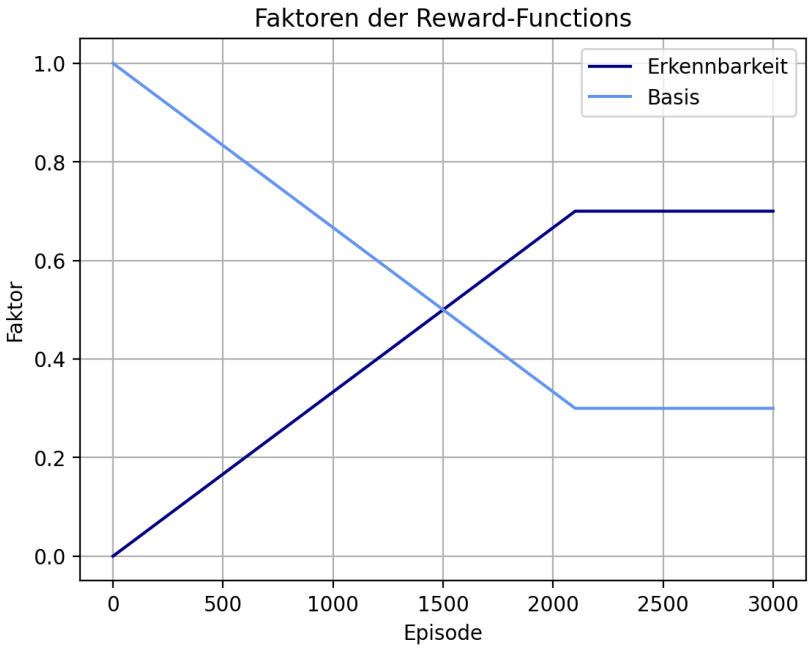

Abbildung 3.6: Veränderung der Faktoren der Basis Reward-Function und der Reward-Function der Erkennbarkeit über das Training hinweg (eigene Abbildung)

Richtung sich der Agent pro Step bewegt. Die folgende Formel beschreibt, wie sich die Position des Agenten vom Step t bis zum nächsten Step $t+1$ ändert:

$$\vec{p}(t+1) = \vec{p}(t) + \vec{v}(t)$$

$\vec{p}(t)$ beschreibt die Position des Agents als einen Ortsvektor auf der Zeichenfläche zum Step t und $\vec{v}(t)$ beschreibt die Geschwindigkeit des Agenten zum Step t. Die Position rundet in jedem Step auf ganze Zahlen. Das kommt daher, dass die Geschwindigkeit auch Dezimalzahlen annehmen kann, aber die Position nur durch ganze Zahlen dargestellt ist.

Zur Geschwindigkeit des Agent wird in jedem Step ein Beschleunigungsvektor addiert. Jede Action, die der Agent wählen kann, entspricht einem anderen Beschleunigungsvektor. Der Action-Space (siehe 2.2.1 Aufbau und Funktionsweise) besteht neu aus 42 Actions. 21 der 42 Actions entsprechen Beschleunigungsvektoren im zeichnenden Zustand. Die anderen 21 Actions entsprechen den selben Vektoren im nicht zeichnenden Zustand. Die 21 verschiedenen Beschleunigungsvektoren haben folgende Form: Ein Vektor entspricht dem Nullvektor. Dieser verändert die Geschwindigkeit des Agents nicht. 8 Vektoren sind um den Agent herum mit einer Länge von 0.9 Pixeln in gleichmässigem Abstand von einander angeordnet. Zusammen bilden diese Vektoren einen Kreis um den Agent. Die restlichen 12 Vektoren sind in einem grösseren Kreis gleichmässig

angeordnet. Die Vektoren haben dabei eine Länge von 1.8 Pixeln (siehe **??**). Mit dem gewählten Beschleunigungsvektor $\vec{a}(t)$ berechnet sich die Geschwindigkeit im nächsten Step $t+1$ aus dem aktuellen Step t durch folgende Formel:

$$\vec{v}(t+1) = \vec{v}(t) + \vec{a}(t)$$

Der Betrag der Geschwindigkeit $\vec{v}(t+1)$ des Agents wird, unabhängig von der gewählten Action, in jedem Step um 0.2 Pixel verringert. Das simuliert eine Reibungskraft, die auf den Agent einwirkt.

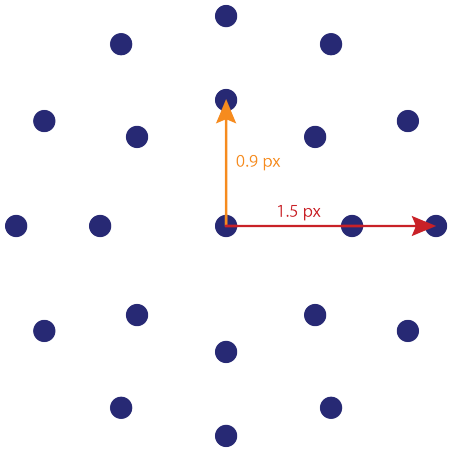

Abbildung 3.7: Action-Space in der physikalischen Umgebung (eigene Abbildung)

Die Veränderung in der Umgebung erfordert weitere Anpassungen im neuronalen Netz (siehe 2.1.3 künstliche neuronale Netze). Ohne diese Anpassungen kann die KI den akkumulierten Reward nicht maximieren. Das Problem ist, dass die aktuelle Geschwindigkeit des Agents kein Teil der Observation (siehe 2.2.1 Aufbau und Funktionsweise) ist. Der Agent berücksichtigt deswegen seine Geschwindigkeit nicht in seinen Entscheidungen. Die Lösung dieses Problems bietet eine Verschiebung des Local image patch (siehe 2.3.1 Doodle-SDQ). Im Grundprogramm entspricht der Mittelpunkt des Local image patchs genau der Position des Agents. Neu befindet sich der Mittelpunkt dort, wo sich der Agent laut seiner aktuellen Geschwindigkeit im nächsten Schritt befinden wird. Durch diese Verschiebung des Local image Patches erhält der Agent Informationen über seine Geschwindigkeit, ohne dessen numerischen Wert zu kennen. Wie im Grundprogramm gibt der Local image patch den gesamten Bereich an, in dem sich der Agent im nächsten Schritt befinden kann. Die tatsächliche neue Position des Agents wird durch die Action seiner Wahl bestimmt. Die Grösse des Local image patches schrumpft von 7×7 Pixeln auf 5×5 Pixel,

da alle möglichen Positionen des Agents nach einem Step auf einem 5×5 Feld Platz haben (siehe Abbildung 3.8).

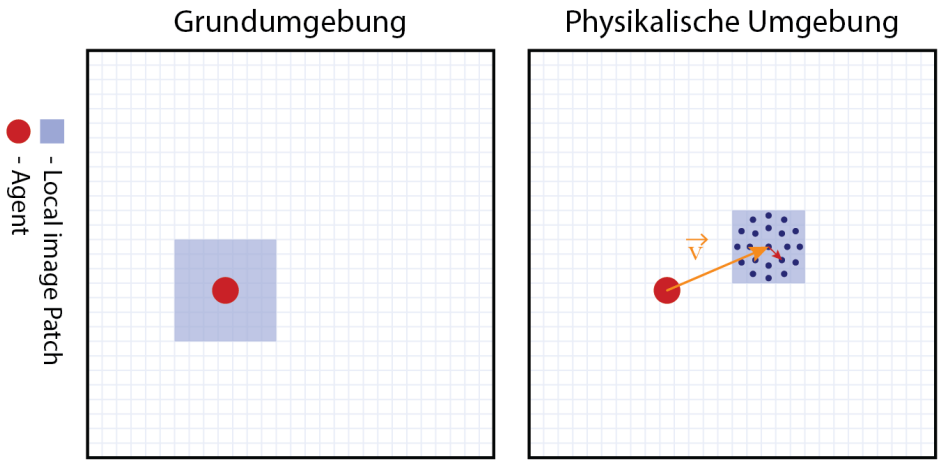

Abbildung 3.8: Angabe der Geschwindigkeit durch eine Verschiebung des Local image Patches (eigene Abbildung)

Ein weiteres Problem ist, dass der Agent sich durch seine Geschwindigkeit aus den vorgegebenen Grenzen der Zeichenfläche begeben kann. Im Grundprogramm (siehe 3.1.1 Doodle-SDQ als Basis) kann der Agent Actions, die ihn in eine unzulässige Position bewegen würden, nicht auswählen. Wenn allerdings in der phyiskalischen Umgebung die Geschwindigkeit des Agents zu hoch ist, kann dieser keine Actions mehr wählen, die ihn innerhalb der Grenzen der Zeichenflächen hielten. Wenn der Agent durch zu hohe Geschwindigkeit über die Grenze hinausgeht, wird seine Geschwindigkeit auf den Nullvektor zurückgesetzt und die Reward Function löst einen negativen Reward von -0.05 aus. Der negative Reward soll die Häufigkeit dieser Vorfälle vermindern

3.4 Auswertung

Die Auswertung der Daten über die Leistung der künstlichen Intelligenz liefert das Resultat der Methode. Die Auswertung berechnet den Zahlenwert der definierten Kriterien (siehe 3.2 Evaluierung der Leistung) für verschiedene Variationen der KI.

Die Variationen werden auf ihre Leistung für drei verschiedene Datensets überprüft. Die drei Datensets beinhalten verschiedene Arten von handgemachten Strichbildern (siehe Tabelle 3.1). Das erste Datenset, MNIST, beeinhaltet Ziffern, die in den Trainingsdaten nicht vorkommen. Das zweite

Datenset, EMNIST Letters, beeinhaltet die 26 Kleinbuchstaben des Alphabets. Das dritte Datenset, QuickDraw, beeinhaltet Zeichnungen von insgesamt 345 verschiedenen Motiven. Die KI wird allerdings nur auf das Nachzeichnen von zehn Motiven überprüft. Die zehn Motive sind: 'Amboss', 'Apfel', 'Besen', 'Eimer', 'Bulldozer', 'Uhr', 'Wolke', 'Computer', 'Auge' und 'Blume' (siehe Abbildung 3.9). Die Bilder in den drei Datensets sind gleich verarbeitet wie die Trainingsdaten (siehe 3.1.2 Präparierung der Daten und Optimierung).

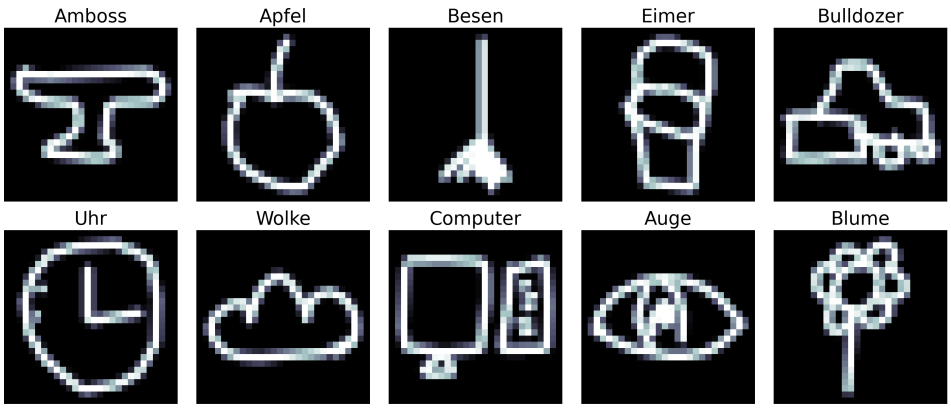

Abbildung 3.9: Beispiele der verwendeten Motiven aus dem QuickDraw Datenset

Die Variationen (siehe 3.3 Variationen) der KI umfassen zwei Umgebungen und drei Reward-Functions 2.2.1 Aufbau und Funktionsweise. Für jede Variation ist zur Vereinfachung eine Abkürzung definiert.

- 3.1 Grundprogramm Umgebung: Grund
- 3.3.4 Physikalische Umgebung: Physik
- 3.3.3 Training auf Erkennbarkeit: Basis
- 3.3.2 Training auf Geschwindigkeit: Speed
- 3.3.3 Training auf Erkennbarkeit (von MNIST Ziffern): MNIST

Die folgenden Kombinationen an Variationen der künstlichen Intelligenz werden ausgewertet. Diese Kombinationen stellen einzelne Versionen der KI dar

- Grund-Basis
- Grund-MNIST
- Grund-Speed
- Grund-MNIST-Speed
- Physik-Basis
- Physik-MNIST
- Physik-Speed
- Physik-MNIST-Speed

3.4.1 Testumgebung

Die Leistungen der verschiedenen Variationen der künstlichen Intelligenz werden in einer Testumgebung (siehe 2.1.1 Funktionsweise eines Machine Learning Modelles) ausgewertet. Zwischen der Trainingsumgebung und der Testumgebung sind drei relevante Unterschiede erkennbar. Erstens trainiert die KI in der Testumgebung nicht. Die Testumgebung übernimmt eine trainierte Version der KI und verändert diese während dem Test nicht. Zweitens wählt der Agent in keinem Fall mehr eine zufällige Action. Stattdessen wählt er immer die Action mit dem höchsten Q-Value (gleichbedeutend mit $\varepsilon = 0$) (siehe 2.2.1 Aufbau und Funktionsweise). Der Dritte Unterschied liegt in den Strichbildern, die für die künstliche Intelligenz als Vorlage dienen. Im Test zeichnet das Computerprogramm 2000 Bilder aus einem der drei zur Verfügung stehenden Datensets (siehe Tabelle 3.1).

Am Ende jeder Episode (das heisst jeder Zeichnung), wird der Zahlenwert für die verschiedenen Kriterien (siehe 3.2 Evaluierung der Leistung) nach ihrer Definition ausgewertet und gespeichert. Die KI zeichnet auch in der Testumgebung für 64 Steps. Wenn eine Zeichnung der Definition entsprechend früher fertig ist (siehe 3.2.3 Geschwindigkeit), wird die Anzahl Steps zu diesem Zeitpunkt als Wert des Kriteriums der Geschwindigkeit gespeichert und die KI zeichnet weiter. Der Durchschnitt aller gespeicherten Werte eines Kriteriums entspricht der Leistung der getesteten Variation in diesem Kriterium. Das Kriterium der Erkennbarkeit (siehe 3.2.1 Erkennbarkeit) verwendet zur Auswertung entsprechend dem verwendeten Datenset im Test, dasjenige vortrainierte Modell, das auf dem selben Datenset trainiert ist. Da das Kriterium der Erkennbarkeit jeweils den Wert 0 oder 1 hat, ergibt der Durchschnitt aus allen Werten dieses Kriteriums eine prozentuale Angabe (in Dezimalform) darüber, in wie vielen Fällen das richtige Motiv erkannt wird.

Kapitel 4

Resultate

Die Resultate bestehen aus drei Tabellen. Jede Tabelle beschreibt die Leistung der acht Versionen (siehe 3.3 Variationen), bezogen auf die drei definierten Kriterien (siehe 3.2 Evaluierung der Leistung). Die Daten in den Tabellen stammen aus den Tests der KI (siehe 3.4 Auswertung) Der Unterschied in den Tabellen liegt im Datenset, mit denen die Versionen der KI jeweils getestet sind. Eine Sammlung von gezeichneten Strichbildern ergänzt die Resultate. Die Strichbilder sind dabei jeweils in Paaren angeordnet. Das linke Bild im Paar zeigt die Vorlage aus dem Datenset und das rechte Bild zeigt die nachgezeichnete Variante von der KI. Die Zeichnungen, die in der Sammlung vertreten sind, sind zufällig ausgewählt aus dem Test der jeweiligen Version der KI. Die Bilder haben einen Farbverlauf, der den zeitliche Verlauf des Zeichnens dargestellt. Die Helligkeit eines Striches ist proportional zu dem Step, in dem dieser gezeichnet wurde. Das bedeutet, dass dunklere Striche früher gezeichnet werden als hellere Striche. Bewegungen des Agents, in denen dieser nicht zeichnet, sind in den Bildern nicht erkennbar.

4.1 Tabellen

Tabelle 4.1: Testen auf MNIST Datenset — 2000 Tests

	Übereinstimmung %	Erkennbarkeit %	Geschwindigkeit
Grund-Basis	86.5	86.6	24.5
Grund-MNIST	66.8	64.3	51.2
Grund-Speed	86.2	86.6	25.5
Grund-MNIST-Speed	61.4	55.1	56.8
Physik-Basis	56.4	46.4	62.5
Physik-MNIST	38.4	35.7	63.9
Physik-Speed	63.0	58.2	61.2
Physik-MNIST-Speed	29.2	27.3	63.7

Tabelle 4.2: Testen auf EMNIST Datenset — 2000 Tests

	Übereinstimmung %	Erkennbarkeit %	Geschwindigkeit
Grund-Basis	86.8	74.5	38.2
Grund-MNIST	65.2	45.0	57.4
Grund-Speed	87.8	77.0	37.7
Grund-MNIST-Speed	62.2	40.0	60.9
Physik-Basis	57.6	32.4	63.5
Physik-MNIST	43.3	23.6	63.9
Physik-Speed	56.3	35.0	63.6
Physik-MNIST-Speed	30.2	13.9	64.0

Tabelle 4.3: Testen auf QuickDraw-Datenset — 2000 Tests

	Übereinstimmung %	Erkennbarkeit %	Geschwindigkeit
Grund-Basis	79.1	80.5	39.1
Grund-MNIST	57.3	62.5	59.9
Grund-Speed	79.5	82.2	40.0
Grund-MNIST-Speed	54.9	58.9	62.5
Physik-Basis	48.1	55.7	63.8
Physik-MNIST	30.5	38.9	64.0
Physik-Speed	50.0	58.3	63.6
Physik-MNIST-Speed	22.4	31.1	64.0

4.2 Bildersammlung

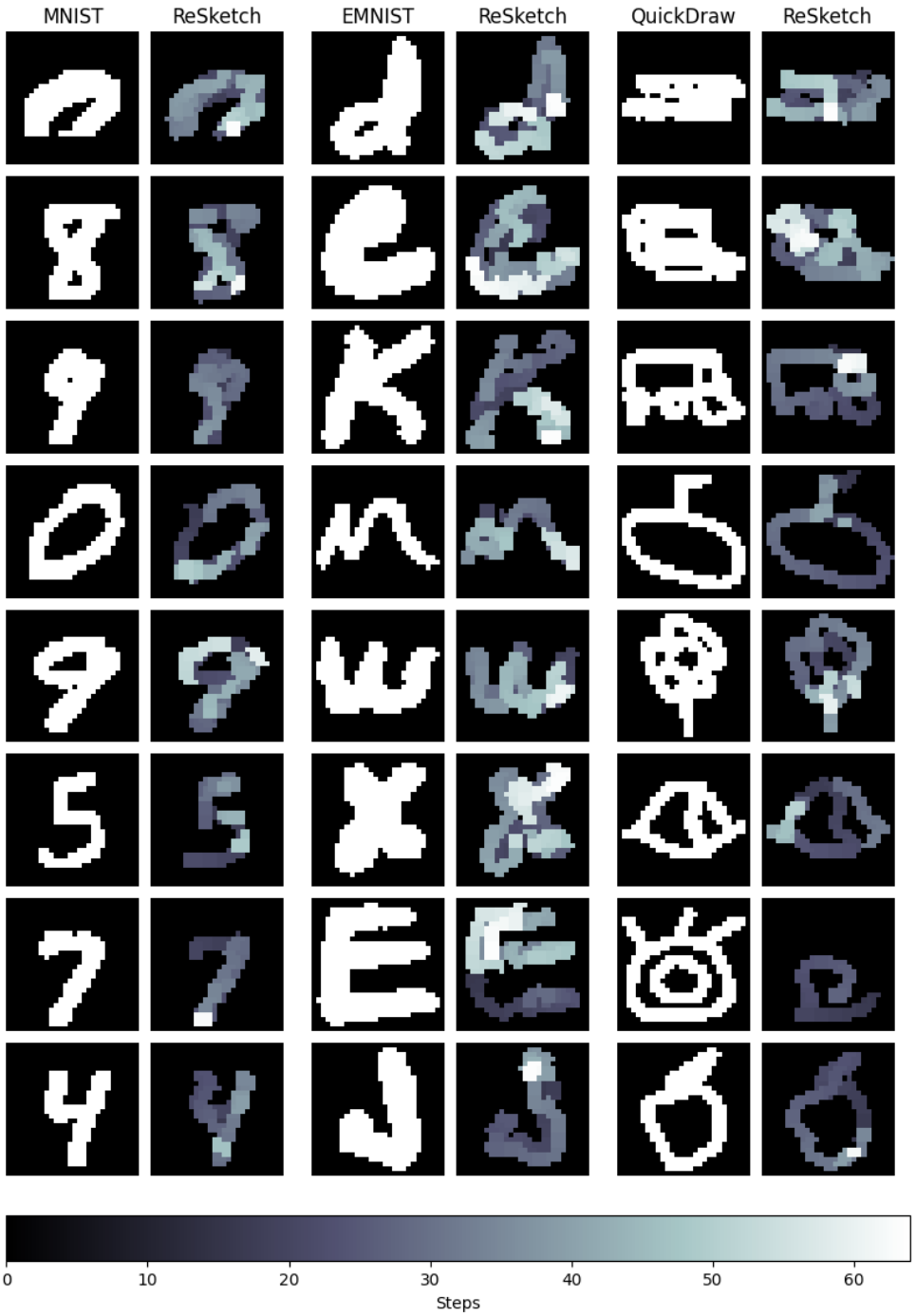

Abbildung 4.1: Grund-Basis

Abbildung 4.2: Grund-MNIST

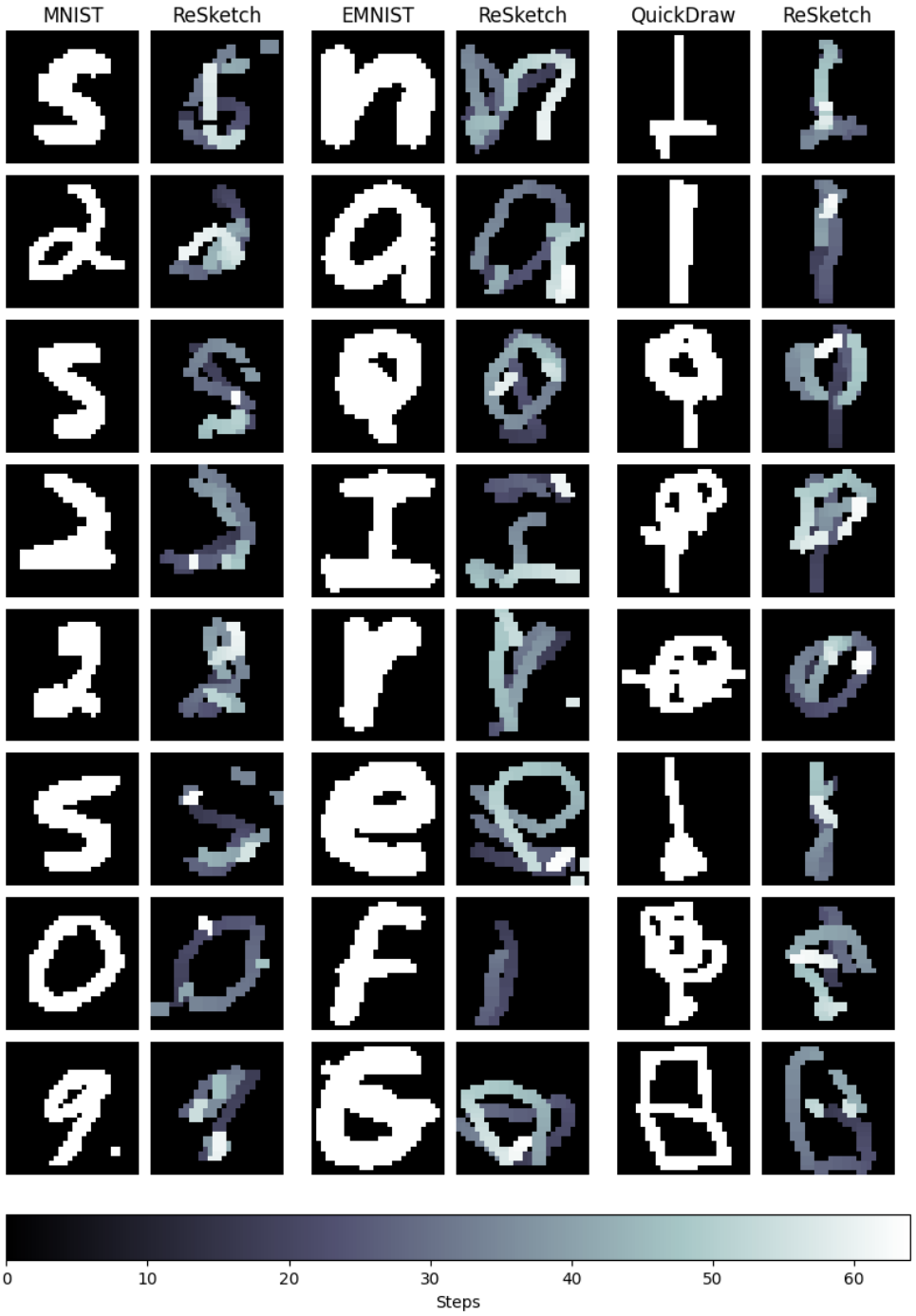

Abbildung 4.3: Physik-Basis

Abbildung 4.4: Physik-Speed

Kapitel 5

Diskussion

Die Diskussion analyisiert die Resultate der Methode (siehe 3 Methode), um daraus eine Antwort auf die Fragestellung zu bilden. Zu diesem Zweck werden einige allgemeine Feststellungen getroffen und die Unterfragen beantwortet (siehe 5.1 Fragestellung und Unterfragen). Im zweiten Teil der Diskussion folgt ein Fazit, ein Ausblick (siehe 5.2 Fazit und Ausblick), und eine Selbstreflexion (siehe 5.3 Selbstreflexion). Dabei verschiebt sich der Fokus von der Fragestellung weg und auf eine allgemeinere Betrachtung der Arbeit.

5.1 Fragestellung und Unterfragen

Die Fragestellung und die Unterfragen decken nicht alle Erkenntnisse aus den Resultaten ab. Einige allgemeine Feststellungen geben Einblick, wie die Resultate (siehe 4 Resultate) zu verstehen sind.

Die Grund-Basis Version und die Grund-Speed Version (siehe 3.4 Auswertung) erreichen in allen Kriterien für alle Datensets die beste Leistung. Die Resultate zwischen den Versionen sind dabei fast ununterscheidbar. Vor allem im Kriterium der Geschwindigkeit (siehe 3.2.3 Geschwindigkeit) zeigt die Speed Variation keine Verbesserung. Unter den Versionen, die auf der physikalischen Umgebung basieren, erreichen Ebenfalls die Physik-Basis und die Phyisk-Speed Versionen die beste Leistung. Die Grund-MNIST Version und die Physik-MNIST Version sind in allen Kriterien schlechter als die Basis und Speed Versionen. Auch im Kriterium der Erkennbarkeit (siehe 3.2.1 Erkennbarkeit) bringt die Variation keinen Vorteil. Die Physik-MNIST-Speed Version erbringt die schlechteste Leistung. Eine Erklärung dafür ist, dass diese Version eine Kombination von allen Variationen (siehe 3.3 Variationen) ist und somit von der besten Version, der Grund-Basis Version am stärksten abweicht.

Die Bildersammlung (siehe 4.2 Bildersammlung) zeigt, dass die KI in vielen Fällen präzise der Linie folgt und selten willkürliche Sprünge begeht. Das ist interessant, weil der KI nie explizit mitgeteilt wird, wie genau sie zeichnen sollte.

5.1.1 Beantwortung der Unterfragen

Insgesamt sechs Unterfragen werden beantwortet (siehe 1 Einleitung). Diese Unterfragen weiten die Fragestellung aus und tragen zu der schlussendlichen Antwort auf die Fragestellung bei. Die Antworten beruhen auf den Resultaten, aber auch auf Erkenntnissen aus der Methode selbst (siehe 3 Methode).

Wie kann die Architektur einer KI aussehen, die das Nachzeichnen erlernt?

Unter der Annahme, dass die KI dieser Arbeit das Nachzeichnen erlernt, (siehe 5.1.2 Beantwortung der Fragestellung), kann die Architektur genau so aussehen, wie sie in dieser Arbeit beschrieben ist (siehe 3.1.1 Doodle-SDQ als Basis).

Wie lässt sich die Leistung der KI in dieser Aufgabe beurteilen?

Die Leistung der KI lässt sich durch die definierten Kriterien (siehe 3.2 Evaluierung der Leistung) beurteilen. Das Kriterium Die Übereinstimmung ist ein objektiver und Absoluter Wert, und somit das Aussagekräftigste Kriterium. Ausserdem ist der maximale Wert des Kriteriums, unabhängig vom gezeichneten Bild, gleich 1. Dadurch ist das Kriterium geeignet für Vergleiche zwischen Versionen der KI.

die Kriterien der Erkennbarkeit und der Geschwindigkeit sind an subjektive Annahmen gebunden. Zum Beispiel wird für das Kriterium der Geschwindigkeit ein subjektiver Punkt der Fertigstellung definiert (siehe 3.2.3 Geschwindigkeit). Dadurch sinkt ihre Aussagekraft. Allerdings verändern sich die Annahmen nicht und die Kriterien sind in jedem Fall durch einen Zahlenwert repräsentiert. Somit eignen sich auch diese Kriterien für Vergleiche zwischen Versionen der KI. Aus der Annahme heraus, dass für Menschen beim Nachzeichnen Erkennbarkeit wichtiger als absolute Genauigkeit ist, ergibt sich das Kriterium der Erkennbarkeit als besonders wichtiges. Aus diesem Grund ist das Kriterium in der Fragestellung (siehe 5.1.2 Beantwortung der Fragestellung) vermerkt.

Wie lässt sich die Leistung der KI verbessern?

Bezogen auf die definierten Kriterien erreicht die Grundversion Werte, die durch die implementierten Variationen nicht oder nur marginal verbessert werden. Die Variationen der KI sind somit insgesamt ein gescheiterter Versuch der Verbesserung der Leistung. Die Grundversion erfuhr allerdings

in dessen Entwicklung signifikante Verbesserungen. Die grössten Verbesserungen stammen aus der Optimierung der Hyperparamter durch den Baysian Optimization Algorithmus (siehe 3.1.2 Präparierung der Daten und Optimierung). Zum Beispiel hat die Grösse des Replay Buffers einen erheblichen Effekt auf die Leistung.

Welche Einflüsse haben Physiksimulationen auf die Leistung der KI?

Bezogen auf die definierten Kriterien verschlechtert sich die Leistung der KI. Alle Versionen, die auf der Grundumgebung basieren, erzielen höhere Werte als die gleichen Versionen basierend auf der phyiskalischen Umgebung (siehe 3.3.4 Physikalische Umgebung). Die physikalische Umgebung hat zum Ziel, die Bewegungen der KI realistischer zu gestalten. In diesem Bereich kann der Einfluss nicht objektiv bestimmt werden. Aus Beobachtungen der Bilder, welche in der physikalischen Umgebung gezeichnet sind (siehe 4.2 Bildersammlung), gehen ebenfalls keine Erkenntnisse in diesem Bereich hervor. Die Bilder unterscheiden sich kaum von denjenigen aus der Grundumgebung.

Wie ändert sich die Leistung der KI bei Strichbildern, die sich von den Trainingsdaten unterscheidenn

In allen acht Versionen bleibt die Leistung der KI zwischen den drei Datensets (siehe **??**) vergleichbar. Die Tabellen Tabelle 4.1, Tabelle 4.2 und Tabelle 4.3 zeigen die Leistung der Grund-Basis Version und der Physik-Basis Version in den drei definierten Kriterien, getestet auf die drei Datensets. Der Wert der Übereinstimmung zwischen dem MNIST Datenset und dem EMNIST Datenset ist beinahe identisch. Für beide Versionen ist der Wert der Übereinstimmung für das QuickDraw Datenset niedriger. Insgesamt ist die KI in diesem Kriterium jedoch kaum Beeinflusst durch die Wahl des Datensets. Die Analyse der anderen zwei Kriterien führt zu einer ähnlichen Schlussfolgerung. Interessant ist, dass vor allem die Grund-Basis Version eine viel höhere Geschwindigkeit im Zeichnen von MNIST Zahlen hat, als im Zeichnen von EMNIST Buchstaben. Obwohl die Formen zu grossem Teil ähnlich sind, scheint die KI durch das spezifische Training auf MNIST Ziffern eine höhere Geschwindigkeit zu entwickeln.

Inwiefern lässt sich das Zeichnen der KI mit menschlichem Zeichnen vergleichen?

Die Antwort auf diese Frage leitet sich nicht aus den objektiven Resultaten ab, sondern basiert auf subjektiven Beobachtungen. Die Bewegungen in der Physik-Version der künstlichen Intelligenz basieren grundsätzlich auf den selben Gesetzen wie die Bewegungen in der echten Welt. Allerdings sind die Bewegungen stark vereinfacht im Vergleich zu menschlichen Bewegungen

Ausserdem ist für die künstliche Intelligenz der Druck des Stiftes nicht veränderbar. Zumindest Konzeptuell nähert die künstliche Intelligenz menschliches Zeichnen, bezogen auf die physischen Einschränkungen, an. Einige menschliche Gewohnheiten sind bei der künstlichen Intelligenz allerdings nicht beobachtbar. Zum Beispiel beginnt die künstliche Intelligenz beim Zeichnen von Ziffern an zufälligen Orten, während Menschen in der Regel für jede Ziffer an der selben Stelle ansetzen einer Ziffer immer an der selben Stelle

5.1.2 Beantwortung der Fragestellung

Die Fragestellung lautet: In wiefern kann eine künstliche Intelligenz lernen, Strichbilder auf eine physische Weise nachzuzeichnen, sodass diese durch ein automatisches System erkannt werden? (siehe 1 Einleitung) Diese Frage hat mehrere Aspekte, die teilweise bereits durch die Unterfragen (siehe 5.1.1 Beantwortung der Unterfragen) erfasst werden. Für die schlussendliche Antwort folgt eine genauere Ausführung der Aspekte.

Die KI zeichnet durch Physiksimulationen und durch allgemeine Einschränkungen der Bewegungsfreiheit auf eine annähernd physische Weise. Das Zeichnen ist nur annähernd physisch, da alle Bewegungen simuliert und in keiner phyischen Umgebung umgesetzt sind. Ausserdem sind die Simulationen nicht vollkommen realitätsgetreu (siehe 5.1.1 5.1.1 Welche Einflüsse haben Physiksimulationen auf die Leistung der KI?)

Die künstliche intelligenz erlernt das Nachzeichnen bezogen auf die Kriterien, nach denen es definiert ist, erfolgreich. Dafür sprechen die Werte der besten Versionen für das Nachzeichnen von Ziffern, die teilweise an den Höchstwert grenzen (siehe 5.1 Fragestellung und Unterfragen). Die hohen Werte im Kriterium der Erkennbarkeit bestätigen ausserdem, dass die Zeichnungen der KI in den meisten Fällen von einem automatischen System erkannt werden.

Laut der Fragestellung soll die KI das Nachzeichnen von Strichbildern erlernen. Damit ist implizit das Nachzeichnen von allen möglichen Arten von Strichbildern gemeint. Die Leistung der KI kann nicht auf alle möglichen Strichbilder überprüft werden, aber der Test mit drei verschiedenen Datensets ergibt vielversprechende Resultate (siehe 4.1 Tabellen). Die KI erlernt erfolgreich das Nachzeichnen von Ziffern, Kleinbuchstaben und zehn zufälligen Motiven aus dem QuickDraw Datenset. Durch die Vielfalt im QuickDraw Datenset kann die Annahme getroffen werden, dass die KI zumindest einen grossen Teil an Strichbildern nachzeichnen kann.

Die zusammenfassende Antwort auf die Frage lautet somit: Eine künstliche Intelligenz kann das Nachzeichnen von Strichbildern auf annähernd physiche Weise in dem Sinne lernen, dass die fertige Zeichnung von einem

automatischen System grösstenteils erkannt wird, die Übereinstimmung zwischen der Vorlage und der Zeichnung gross ist und die Zeichnung nicht viel Zeit in Anspruch nimmt.

Diese Antwort bezieht sich auf die genaue Frage, wie sie in der Einleitung steht. Der nächste Abschnitt beurteilt die Frage durch die Erkenntnisse aus dieser Arbeit und geht auf mögliche Erweiterungen ein.

5.2 Fazit und Ausblick

Die Resultate erlauben eine positive Antwort auf die Fragestellung. Diese Antwort setzt allerdings einige Annahmen vorraus, die weiter diskutiert werden können. Die grösste Annahme bezieht sich auf die Definition des Nachzeichnens. Diese Arbeit definiert Nachzeichnen durch drei Kriterien und durch physische Rahmenbedingungen. Die Kriterien sind für eine künstliche Intelligenz sinnvoll gewählt (siehe 5.1.1 Wie lässt sich die Leistung der KI in dieser Aufgabe beurteilen?), allerdings wären auch ander Kriterien möglich. Nachzeichnen ist eine menschliche Tätigkeit. Dieser menschliche Aspekt ist in den definierten Kriterien nicht enthalten.

Die physischen Rahmenbedingungen unterscheiden sich von denjenigen, die ein Mensch erfährt. Das kommt daher, dass die phyischen Rahmenbedingungen für die KI lediglich simuliert sind. Das verunmöglicht eine umfassende Antwort auf die Frage, ob die künstliche Intelligenz auf eine physische Weise zeichnet. Dieses Problem könnte mit einem Roboter gelöst werden, der die künstliche Intelligenz in eine reale, phyische Umgebung überführt. Der Roboter könnte somit verschiedenste Strichbilder auf einem echten Stück Papier, und somit zwangsläufig auf physische Weise nachzeichnen. Aktuell sind die Bewegungen der künstlichen Intelligenz in gewissen Belangen eingeschränkt. So ist beispielsweise die Druckstärke nicht variierbar. Ausserdem zeichnet die künstliche Intelligenz vorwiegend kleine Strichbilder. Experimente mit grösseren Konstrukten, wie ganze Wörter, wären eine mögliche Erweiterung.

Alles in allem sind eine Vielzahl an denkbaren Fragen und Ideen möglich, die auf ReSketch, der künstlichen Intelligenz hinter dieser Arbeit, basieren.

5.3 Selbstreflexion

Die Selbstreflexion gibt genauere Einblicke in die Vorangehensweise hinter dieser Arbeit. Diese Dokumentation ist grundsätzlich eine Zusammenfassung der wichtigsten Ereignisse. Viele Aspekte, wie auch die Arbeitsweise bleiben verschwiegen. Die Selbstreflexion geht näher auf drei wichitige Aspekte ein, die in der zusammengefassten Dokumentation nicht genug betont sind.

Die Dokumentation ist mit LaTeX und spezifischer der ETH Thesis Formatvorlage („CADMO, Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich“, n. d.) formatiert. Ein Grossteil der Abbildungen stammt von den Autoren und ist in Adobe Illustrater oder der Python Library Matplotlib erstellt

5.3.1 Optimierung der KI

Insgesamt sind acht Versionen der KI präsentiert. Im Verlaufe des Projektes gab es viele weitere Versuche, die Leistung der künstlichen Intelligenz zu verbessern. Diese Versuche führten allerdings häufig dazu, dass die KI den akkumulierten Reward (siehe 2.2.1 Aufbau und Funktionsweise) nicht mehr maximieren konnte. In der Dokumentation sind deswegen nur diejenigen Versuche vermerkt, die tatsächlich funktionieren. Das Problem hinter den versuchten Variation liegt darin, dass die Ursache hinter ihrem Scheitern oder ihrem Erfolg häufig nicht erkennbar ist. Das macht die Optimierung der künstlichen Intelligenz allgemein schwierig.

Die Strategie hinter der Optimierung besteht in den meisten Fällen aus wiederholtem Ausprobieren mit Anpassungen zwischen jedem Versuch. Hilfsmittel, wie der Baysian Optimization Algorithmus (siehe 2.1.2 Hyperparameter), vereinfachen diese Aufgabe massgeblich. Tatsächlich ermöglichte der Baysian Optimization Algorithmus eine Verbesserung der KI von ungefähr $40 - 50\%$ mehr im Kriterium der Übereinstimmung (siehe 3.2.2 Prozentuale Übereinstimmung). Diese Strategie der Optimierung ist für einen Computer sehr ressourcenintensiv. In den längsten Optimierungsarbeiten liefen die beiden Computer, auf denen die Arbeit verrichtet wurde, zusammen länger als 48 Stunden.

5.3.2 Analyse der künstlichen Intelligenz

Eine Analyse der künstlichen Intelligenz ist notwendig, um die Fragestellung und die Unterfragen zu beantworten. Aber auch während der Entwicklung der KI ist eine stetige Analyse nötig, um dise zu verstehen und zu verbessern.

Die Analyse besteht hauptsächlich darin, die Leistung der künstlichen Intelligenz zu beurteilen. Das geschieht mittels den Kriterien, die für diesen Zweck definiert sind (siehe 3.2 Evaluierung der Leistung). Die Kriterien sind dabei so definiert, dass sie für jede mögliche Variation identisch bleiben. Der durchnittliche akkumulierte Reward ist beispielsweise absichtlich kein Kriterium. Der akkumulierte Reward ist abhängig von der Reward-Function (siehe 2.2.1 Aufbau und Funktionsweise) und unterscheidet sich somit zwischen Variationen. Dieser ist somit nicht für einen Vergleich zwischen Variationen geeignet.

Einzelne Variationen vergleichbar zu halten, ist ein allgemeines Ziel der Analyse. Deswegen basieren alle Variationen auf der gleichen Architektur der KI. Die Funktionalität des Grundprogrammes (siehe 3.1 Grundprogramm) ist gründlich getestet. Zum Beispiel sind die Testdaten darauf überprüft, dass sie keine Datenpunkte aus den Trainingsdaten beinhalten. Diese Tests eliminieren mögliche Fehlerquellen falls die künstliche Intelligenz unerwartetes Verhalten aufzeigt.

Eine weitere Form von Analyse stammt aus der Sammlung von Daten über das Lernverhalten der KI. So wird aus jedem Training ein Graph erstellt, der die durchschnittliche Leistung der KI in jeder Episode erfasst (siehe Abbildung 5.1). Die Leistung ist dabei durch den akkumulierten Reward in jeder Episode repräsentiert. Wie erwähnt können Versionen der KI nicht anhand ihres akkumulierten Rewards verglichen werden. Der akkumulierte Reward zeigt allerdings für einzelne Versionen am präzisesten, in wiefern diese ihren Reward maximieren können.

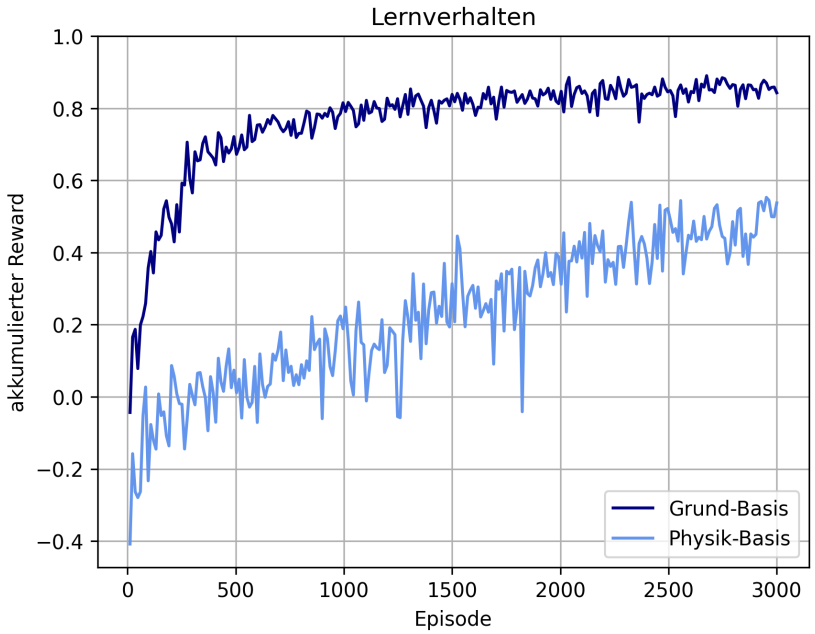

Abbildung 5.1: Akkumulierter Reward zu jeder Episode (Lernverhalten) der Grund-Basis Version und der Physik-Basis Version (eigene Abbildung)

5.3.3 Verwendung von Git und GitHub

Die Verwendung von Git und Github (siehe 2.4 Git und GitHub) erleichtert die Arbeit an einem Projekt von dieser Grösse massgelich. Die Programme ermöglichen einfache Zusammenarbeit am Programmcode und an der

Dokumentation. GitHub dient dabei zusätzlich als Hilfsmitel zur Organisation durch die integrierte Funktion der Project Boards. Diese Funktion hätte allerdings zu grösserem Ausmass Verwendung finden können.

Die Funktion der Branches und Commits von Git werden durch die Arbeit hindurch konsequent verwendet. Dabei wird die Giflow Arbeitsweise, abgesehen von den Release Branches und den Hotfix Branches, angewendet (siehe 2.4.1 Git) Neben den Feature Branches werden ausserdem Dokumentation Branches eingeführt, in denen jeweils ein Kapitel der Dokumentation verfasst wird. Für die Zusammenführungen der wichtigsten Branches wird das Prinzip der Pull Request (siehe 2.4.2 GitHub) angewendet. Die Pull Request muss für jeden Branch von beiden Autoren akzeptiert werden.

Ein weiterer Vorteil von Git und Github ist die Zugänglichkeit des Projektes. Das gesamte Projekt ist unter folgendem Link einsehbar: https://github.com/LarsZauberer/Nachzeichner-KI. Im Projektordner sind vortrainierte Variationen der künstlichen Intelligenz enthalten. Das Projekt auf GitHub erfährt möglicherweise Erweiterungen, die in dieser Arbeit nicht mehr erfasst sind.

Kapitel 6

Zusammenfassung

Diese Untersuchung beantwortet die Frage, inwiefern eine künstliche Intelligenz Strichbilder auf eine physische Weise nachzeichnen kann, sodass diese durch ein automatisches System erkannt werden. Mit Strichbildern sind in diesem Fall Ziffern aus dem MNIST Datenset, Buchstaben aus dem EMNIST Datenset und weitere Motive aus dem QuickDraw Datenset gemeint.

Zur Beantwortung der Fragestellung wird der Begriff des Nachzeichnes definiert. Zu der Defintion gehören die Rahmenbedingungen, nach denen eine Tätigkeit als Nachzeichnen gilt, und die Kriterien, die die Leistung im Nachzeichnen beurteilen. Zu den Rahmenbedingungen gehören unter anderem die phyischen Einschränkungen und die ausführbaren Aktionen der KI. Um die Leistung der KI im Nachzeichnen zu bewerten, sind drei Kriterien definiert: Die Übereinstimmung der Pixel, die Erkennbarkeit der Zeichnung und die Geschwindigkeit des Zeichnens. Die Erkennbarkeit der Zeichnung wird durch eine zweite künstliche Intelligenz ermittelt.

Das Ziel ist es, eine künstliche Intelligenz zu entwickeln, welche die gesetzten Rahmenbedingungen erfüllt und eine möglichst gute Leistung nach den definierten Kriterien erzielt. Bei der Grundsätzlichen Architektur des KI handelt es sich um ein Deep Q-Learning Modell, das auf der Arbeit hinter 'Doodle-SDQ' (Zhou et al., 2018) basiert.

Für die Rahmenbedingungen gibt es zwei Ansätze: eine Grundversion und eine physikalische Version. In der Grundversion kann sich die KI schrittweise um eine begrenzte Anzahl Pixel auf einer Zeichenfläche fortbewegen. Ausserdem startet die KI auf einer zufälligen Position auf der Zeichenfläche. Die physikalische Version ist von simulierter Physik begleitet. So kann die KI durch Beschleunigungen ihre aktuelle Geschwindigkeit anpassen und sich so fortbewegen, während diese durch simulierte Reibung kontinuierlich abgebremst wird.

Für die KI existieren weitere Variationen, die dessen Leistung nach einem bestimmten Kriterium verbessern sollen. So existiert ein spezifisches Training auf eine verbesserte Erkennbarkeit und Geschwindigkeit der KI. Durch Kombinationen der Variationen und der Rahemenbedingungen existieren schlussendlich acht Versionen der künstlichen Intelligenz.

Die acht Versionen der künstlichen Intelligenz sind alle auf das Nachzeichnen von Ziffern trainiert. Ein Experiment bestimmt, ob diese Versionen das Nachzeichnen allgemein erlernen. Die Leistung der Versionen wird auf das Nachzeichnen von Strichbildern aus dem Quickdraw und dem EMNIST Letters Datenset gemessen. Wenn die Leistung für diese Strichbilder vergleichbar bleibt mit der Leistung für die Trainingsdaten, ermöglicht das eine positive Antwort für die Fragestellung.

Einge Versionen der künstlichen Intelligenz zeigen hierbei vielversprechende Ergebnisse. Die Grundversion, ohne weitere Variationen, zeichnet in 91% der Fälle eine erkennbare Ziffer, in 70% der Fälle einen erkennbaren Buchstaben, und in 72% der Fälle ein erkennbares Motiv aus dem QuickDraw Datenset.

Literatur

2.1 what is the difference between labelled and unlabelled data? · grokking machine learning. (n. d.). Verfügbar 16. September 2022 unter https://livebook.manning.com/book/grokking-machine-learning/2-1-what-is-the-difference-between-labelled-and-unlabelled-data-/v-4/

Agnihotri, A., & Batra, N. (2020). Exploring bayesian optimization. *Distill*, *5*(5), e26. https://doi.org/10.23915/distill.00026

Arora, S. (2020, 29. Januar). *Supervised vs unsupervised vs reinforcement* [AITUDE]. Verfügbar 25. Juni 2022 unter https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/

Artificial neural network tutorial - javatpoint [Www.javatpoint.com]. (n. d.). Verfügbar 16. September 2022 unter https://www.javatpoint.com/artificial-neural-network

Atlassian. (n. d. a). *Git-flow-Workflow — Atlassian Git Tutorial* [Atlassian]. Verfügbar 7. Oktober 2022 unter https://www.atlassian.com/de/git/tutorials/comparing-workflows/gitflow-workflow

Atlassian. (n. d. b). *Pull Requests — Atlassian Git Tutorial* [Atlassian]. Verfügbar 1. Oktober 2022 unter https://www.atlassian.com/de/git/tutorials/making-a-pull-request

Black Box (Systemtheorie) [Page Version ID: 215860839]. (2021, 24. September). In *Wikipedia*. Verfügbar 2. Oktober 2022 unter https://de.wikipedia.org/w/index.php?title=Black_Box_(Systemtheorie)&oldid=215860839

CADMO, Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich. (n. d.). Verfügbar 7. Oktober 2022 unter https://cadmo.ethz.ch/education/thesis/template.html

Cameron McKenzie. (2021, 24. Februar). *Gitflow release branch process from start to finish example*. Verfügbar 7. Oktober 2022 unter https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Gitflow-release-branch-process-start-finish

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. (n. d.). *Learning representations by back-propagating errors*. Verfügbar 16. September 2022 unter https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf

Deshpande, A. (n. d.). *A Beginner's Guide To Understanding Convolutional Neural Networks Part 2*. Verfügbar 28. April 2022 unter https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

The EMNIST dataset [Last Modified: 2019-03-28T09:45-04:00]. (2017). *NIST*. Verfügbar 16. September 2022 unter https://www.nist.gov/itl/products-and-services/emnist-dataset

Fernando Nogueira. (2014). *Bayesian Optimization: Open source constrained global optimization tool for Python* [original-date: 2014-06-06T08:18:56Z]. Verfügbar 25. Juli 2022 unter https://github.com/fmfn/BayesianOptimization

File:Blackbox.svg - Wikipedia. (n. d.). Verfügbar 8. Oktober 2022 unter https://commons.wikimedia.org/wiki/File:Blackbox.svg

Garnett, R. (n. d.). *Bayesian optimization book* [Bayesian optimization book]. Verfügbar 2. Oktober 2022 unter https://bayesoptbook.com//

Git [Page Version ID: 1062380476]. (2021, 28. Dezember). In *Wikipedia*. Verfügbar 30. Dezember 2021 unter https://en.wikipedia.org/w/index.php?title=Git&oldid=1062380476

GitHub [Page Version ID: 218104003]. (2021, 11. Dezember). In *Wikipedia*. Verfügbar 1. Januar 2022 unter https://de.wikipedia.org/w/index.php?title=GitHub&oldid=218104003

Guillermo Brachetta. (2022, 3. März). *What are git branches? & how they work* [Code institute global]. Verfügbar 1. Oktober 2022 unter https://codeinstitute.net/global/blog/git-branches/

Hagen, T., Tobias Lauer, V. S., & Stephan Trahasch, K. D. (2020, 8. August). *Einführung Menschen Lernen Maschinelles Lernen - ML2*. Verfügbar 31. März 2022 unter https://imla.gitlab.io/ml-buch/ml2-buch/3-1-lr-einfuehrung.html

Hertzmann, A. (2002). Stoke-Based Rendering. Verfügbar 31. März 2022 unter https://www.cs.ucdavis.edu/~ma/SIGGRAPH02/course23/notes/S02c23_3.pdf

Huang, Z., Zhou, S., & Heng, W. (2019). Learning to paint with model-based deep reinforcement learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 8708–8717. https://doi.org/10.1109/ICCV.2019.00880

Jan-Dirk Kranz. (2019, 3. April). *Deep Learning vs Machine Learning - Was ist der Unterschied?* [IT-Talents.de]. Verfügbar 18. Juni 2022 unter https://it-talents.de/it-wissen/programmieren/deep-learning-vs-machine-learning-was-ist-der-unterschied/

Jayawardana, K. (2021, 3. Januar). *Concatenating multiple activation functions and multiple poling layers for deep neural networks* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/concatenating-multiple-activation-functions-and-multiple-poling-layers-for-deep-neural-networks-d48a4b273d30

Kumar, N. (2019, 18. Dezember). *Sigmoid neuron — deep neural networks* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/sigmoid-neuron-deep-neural-networks-a4cd35b629d7

Lâm (Linus), H. T. (2022, 12. August). *Keras DenseNet 121 for Quick, Draw! Doodle Recognition Challenge* [original-date: 2018-12-09T03:16:32Z]. Verfügbar 28. August 2022 unter https://github.com/lamhoangtung/densenet121-quickdraw-doodle-recognition-challenge

Malik, F. (2019, 20. Mai). *What are hidden layers?* [FinTechExplained]. Verfügbar 16. September 2022 unter https://medium.com/fintechexplained/what-are-hidden-layers-4f54f7328263

Mazzia, Vittorio, Salvetti, Francesco, Chiaberge & Marcello. (2022, 29. Juli). ˜ *Efficient-CapsNet* ˜ [original-date: 2020-12-29T18:13:43Z]. Verfügbar 3. August 2022 unter https://github.com/EscVM/Efficient-CapsNet

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013, 19. Dezember). Playing Atari with Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1312.5602

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. (2017, 4. April). Verfügbar 16. September 2022 unter http://yann.lecun.com/exdb/mnist/

Mor, S. (2022, 1. Juni). *EMNIST* [original-date: 2018-07-17T12:21:46Z]. Verfügbar 7. Oktober 2022 unter https://github.com/shubhammor0403/EMNIST

Moriconi, R., Deisenroth, M. P., & Kumar, K. S. S. (2020, 25. September). High-dimensional Bayesian optimization using low-dimensional feature spaces. https://doi.org/10.48550/arXiv.1902.10675

Neuronales Netz [Page Version ID: 210963657]. (2021, 15. April). In *Wikipedia*. Verfügbar 16. September 2022 unter https://de.wikipedia.org/w/index.php?title=Neuronales_Netz&oldid=210963657

Nielsen, M. A. (2015). Neural networks and deep learning [Publisher: Determination Press]. Verfügbar 22. April 2022 unter http://neuralnetworksanddeeplearning.com

Nyuytiymbiy, K. (2022, 28. März). *Parameters and hyperparameters in machine learning and deep learning* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

Ognjanovski, G. (2020, 7. Juni). *Everything you need to know about neural networks and backpropagation — machine learning made easy…* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

Osiński, B., & Budek, K. (2018, 5. Juli). *What is reinforcement learning? the complete guide* [Deepsense.ai] [Section: Deep learning]. Verfügbar 16. September 2022 unter https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

paretos. (2021, 25. Januar). *Bayesian Optimization (Bayes Opt): Easy explanation of popular hyperparameter tuning method*. Verfügbar 22. September 2022 unter https://www.youtube.com/watch?v=M-NTkxfd7-8

PhD, W. v. H. (2021, 30. August). *How to model experience replay, batch learning and target networks* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/how-to-model-experience-replay-batch-learning-and-target-networks-c1350db93172

Piyush Verma & Stelios Diamantidis. (2021, 27. April). *What is Reinforcement Learning? – Overview of How it Works — Synopsys*. Verfügbar 16. September 2022 unter https://www.synopsys.com/ai/what-is-reinforcement-learning.html

Pragati Baheti. (2022, 19. Juli). *Activation functions in neural networks [12 types & use cases]*. Verfügbar 16. September 2022 unter https://www.v7labs.com/blog/neural-networks-activation-functions,%20https://www.v7labs.com/blog/neural-networks-activation-functions

Pramoditha, R. (2021, 29. Dezember). *The concept of artificial neurons (perceptrons) in neural networks* [Medium]. Verfügbar 1. Oktober 2022 unter https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc

Quick, Draw! Image Recognition [original-date: 2019-04-30T10:10:02Z]. (2022, 28. August). Verfügbar 28. August 2022 unter https://github.com/Lexie88rus/quick-draw-image-recognition

Rajendra Koppula. (n. d.). *Exploration vs. exploitation in reinforcement learning*. Verfügbar 2. Oktober 2022 unter https://www.manifold.ai/exploration-vs-exploitation-in-reinforcement-learning

Robbins, B. (2017, 14. Juli). *MACHINE LEARNING: How black is this beautiful black box* [Medium]. Verfügbar 2. Oktober 2022 unter https://towardsdatascience.com/machine-learning-how-black-is-this-black-box-f11e4031fdf

Sadie Bennett. (2019, 10. Juni). *Why machine learning is primarily written in Python* [IBM Developer]. Verfügbar 2. Oktober 2022 unter https://developer.ibm.com/blogs/why-machine-learning-is-primarily-written-in-python/

Simplilearn. (2021, 26. Mai). *What is perceptron: A beginners guide for perceptron [updated]* [Simplilearn.com]. Verfügbar 16. September 2022 unter https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

Spaulding, N. W. (2020, 9. Juli). Is Human Judgment Necessary? Artificial Intelligence, Algorithmic Governance, and the Law. In M. D. Dubber, F. Pasquale & S. Das (Hrsg.), *The Oxford Handbook of Ethics of AI* (S. 0). Oxford University Press. https://doi.org/10.1093/oxfordhb/9780190067397.013.25

Sutton, R. S., & Barto, A. G. (2014). Reinforcement learning: An introduction. *MIT Press*, 352.

TensorFlow. (n. d.). Verfügbar 2. Oktober 2022 unter https://www.tensorflow.org/

Training and test sets: Splitting data — machine learning [Google developers]. (n. d.). Verfügbar 16. September 2022 unter https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data

Unzueta, D. (2022, 15. März). *Convolutional layers vs fully connected layers* [Medium]. Verfügbar 16. September 2022 unter https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b

Wang, M. (2021, 3. Oktober). *Deep q-learning tutorial: minDQN* [Medium]. Verfügbar 15. April 2022 unter https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

Was ist Supervised Learning (Überwachtes Lernen)? [datasolut GmbH]. (n. d.). Verfügbar 16. September 2022 unter https://datasolut.com/wiki/supervised-learning/

What is Artificial Intelligence (AI)? — Glossary. (n. d.). Verfügbar 21. Juni 2022 unter https://www.hpe.com/ch/de/what-is/artificial-intelligence.html

What is git flow — how to use git flow — learn git [Section: Git]. (2022, 17. Juni). Verfügbar 7. Oktober 2022 unter https://www.gitkraken.com/learn/git/git-flow

What is Machine Learning? — Glossary. (n. d.). Verfügbar 21. Juni 2022 unter https://www.hpe.com/ch/de/what-is/machine-learning.html

Zheng, N., Jiang, Y., & Huang, D. (2019). StrokeNet: A Neural Painting Environment. *International Conference on Learning Representations*. https://openreview.net/forum?id=HJxwDiActX

Zhou, T., Fang, C., Wang, Z., Yang, J., Kim, B., Chen, Z., Brandt, J., & Terzopoulos, D. (2018). Learning to Doodle with Deep Q Networks and Demonstrated Strokes. *arXiv:1810.05977 [cs]*. Verfügbar 31. März 2022 unter http://arxiv.org/abs/1810.05977

Abbildungsverzeichnis

Tabellenverzeichnis