KI zur autonomen Erkennung von Wölfen

Teilnehmende (mit Alter): Paul Zörb (14)

Erarbeitungsort: Gymnasium Athenaeum Stade

Projektbetreuende: Herr Dr. Carmesin, Frau von Bargen

Fachgebiet: Informatik

Wettbewerbssparte: Schüler experimentieren

Bundesland: Niedersachsen

Wettbewerbsjahr: 2024

(Abb. 1)

Kurfassung

Mein Ziel ist es ein Kamerasystem zu bauen, welches mit Hilfe einer KI autonom Wölfe erkennen und einen Landwirt oder Jäger alarmieren kann. Auf die Idee kam ich, weil gerade in der Region Stade oft Rinder oder Schafe, die wichtig für den Küstenschutz sind, gerissen werden und es noch keine richtige Lösung für dieses Problem gibt. Mithilfe der KI kann das Verhalten der Wölfe optimal erfasst werden. Weiterhin kann die KI um eine Biofeedback-Loop ergänzt werden, sodass dem Wolf ein Verhalten beigebracht wird, das mit der Nutztierhaltung kompatibel ist. Ich werde ein neuronales Netz mit Tensorflow und Python schreiben und genügend Daten einspeisen, damit die KI Wölfe erkennen und von anderen Tieren unterscheiden kann. Anschließend werde ich diesen Code mit dem Raspberry Pi und der Kamera verknüpfen, damit ein autonomes Warnsystem durch eine Fotofalle entstehen kann.

Inhaltsverzeichnis

1. Einleitung

Heutzutage liest und sieht man immer wieder in den Nachrichten Meldungen von Wölfen, die Tiere wie Schafe reißen. Ein Beispiel dafür ist der Stader-Tageblatt Artikel (2023) „Lauter Ruf nach Wolfsabschüssen“ vom 25.11.2023, der von Grit Klempow geschrieben wurde. In diesem berichtet der Wolfsberater Michael Ohlhof über den aktuellen Stand zum Wolf in der Region Stade. Die Ausbruchssicherheit der Weidetiere und die Wolfsabwehr gleichzeitig durch Zäune zu gewährleisten sei eine Herausforderung, wenn nicht irreal. Nach Ohlhof brauche man verifizierte Beweise, wie Fotos und eine entsprechende Dokumentation. Dem Text ließ sich auch entnehmen, dass die Bevölkerung wolle, dass die Wölfe abgeschossen werden dürfen. Eine neue Regel besage außerdem, dass künftig innerhalb von 21 Tagen nach einem Wolfsriss im Umkreis von 1000 Metern ohne DNA-Nachweis geschossen werden dürfe.
Die Wölfe willkürlich abzuschießen ist meiner Meinung nach keine Option. Ganz abgesehen davon, dass es ethisch nicht richtig wäre, lebt der Wolf historisch in Deutschland und der Mensch dringt in gewisser Weise in seinen Lebensraum ein.

1.1 Idee

Ich war schon immer ein Tierfreund und mich hat die Diskussion über die Wölfe als Problem und der Lösung durch Abschuss betroffen gemacht.

Daher möchte ich eine KI programmieren, welche automatisch Wölfe erkennt.

Anschließend möchte ich diese KI mit einer Kamera verbinden. Damit wäre das von Michael Ohlhof genannte Problem, dass man Beweise und eine Dokumentation brauche gelöst. Außerdem könnte innerhalb kürzester Zeit der Jäger oder Hirte alarmiert werden, damit dieser den Wolf vertreiben bzw. abschießen kann. Mit dieser Kamera könnten sämtliche Informationen über die Wolfsrudel problemlos dokumentiert werden. Außerdem wäre es nicht mehr nötig, wahllos alle Wölfe in einem Gebiet rund um den Wolfsriss abzuschießen, sondern man könnte dies gezielt auf die Wölfe eingrenzen, die die Tiere gerissen haben.

1.2 Ziel

Zuerst möchte ich ein neuronales Netz mit Python und der Library TensorFlow programmieren, welches Wölfe erkennen kann. Dafür werde ich Trainingsbilder von Wölfen und anderen Tieren herunterladen und ein neuronales Netz programmieren, welches ich mit den Bildern trainieren werde. Dieses werde ich so lange verbessern, bis es eine zufriedenstellende Genauigkeit hat. Anschließend schreibe ich ein Programm, welches das Netz nutzen kann, um zu erkennen ob in einem Bild die Silhouette eines Wolfes zu sehen ist.

Als langfristiges Ziel werde ich den Code auf einen Raspberry-PI übertragen und mit einer Wärmebildkamera und einem Bewegungsmelder verknüpfen. Durch das Auslösen des Bewegungsmelders wird die Wärmebildkamera ein Bild aufnehmen, welches durch das neuronale Netz auf eine Wolfssilhouette überprüft wird. Letztendlich werde ich die Kamera in einem Gebiet mit Wölfen aufstellen und das Netz mit selbst

aufgenommenen Bildern trainieren. Anschließend wird der Code so programmiert, dass eine vorab eingegebene Nummer alarmiert wird, sobald ein Wolf gesehen wird.

2. Vorgehensweise, Materialien und Methoden

2.1 Vorgehensweise

Zuerst habe ich mit der Python-Library TensorFlow (2.2.2) ein neuronales Netz erstellt, welches für die Bilderkennung geeignet. Anschließend habe ich mir von unterschiedlichen Internetseiten, wie zum Beispiel Kaggle Trainingsbilder (2.2.3) heruntergeladen. Ich habe mich dafür entschieden, erst einmal nur Wölfe auf der einen Seite und Hunde und Katzen auf der anderen Seite zu nehmen, da dies zur Demonstrierung vollkommen ausreichend ist. Vor dem richtigen Gebrauch müsste das Trainingsdataset noch um viele weitere Bilder erweitert werden. Um die Genauigkeit des Modells weiter zu verbessern, habe ich mithilfe der rembg Library den Hintergrund meiner Testbilder entfernt (2.2.4) und mithilfe der Pillow Library die Kanten in den Bildern hervorgehoben (2.2.5). Um zu überprüfen, wie gut mein Modell funktioniert, habe ich jeweils nach jedem Schritt immer ein Diagramm der Trainingsgenauigkeit in Abhängigkeit der Menge der Trainingsbilder erstellt (3). Mit diesem Diagramm kann ich erkennen, ob vorgenommene Änderungen das Modell verbessert haben und verschiedene Modelle vergleichen.

2.2 Methoden zur Programmierung

2.2.1 Verwendete Module

Für mein Programm zur Wolfserkennung habe ich zum einen die Bibliotheken OpenCV (Open Computer Vision Library) von Intel und Willow Garage (2024) und Pillow (Python Image Library) von Secret Labs AB (2024) für die Bildverarbeitung verwendet. Diese sind die standardmäßigen Tools für das Arbeiten mit Bildern in Python, für diese sind also keine Tests nötig. Für die Hintergrundentfernung habe ich außerdem die Bibliothek rembg (remove Background) verwendet. Diese wurde auf GitHub vom Nutzer danielgatis (2024) entwickelt. Ein Test für das Programm rembg ist in Abb. 2 zusehen. Zuletzt habe ich die Library TensorFlow, von dem Google Brain Team (2024), verwendet, um das Modell zu erstellen und zu trainieren.

Test für TensorFlow

Um das Modul TensorFlow zu testen, habe ich ein neues Modell darauf trainiert zu erkennen, ob auf einem Bild eine horizontale oder vertikale Linie zu sehen ist. Dies ist eine sehr einfache Aufgabe, diese kann aber sehr gut zeigen, ob das Modul TensorFlow so funktioniert wie es soll. Zuerst habe ich eine Funktion geschrieben, die neue Bilder erstellt auf welchen horizontale bzw. vertikale Linien sind (Diese Funktion kann zusammen mit meinem gesamten anderen Code im meinem GitHub-Repository gefunden werden, welches ich am Ende der Langfassung verlinken werde). Diese habe ich in zwei unterschiedlichen Ordnern gespeichert, und sie in

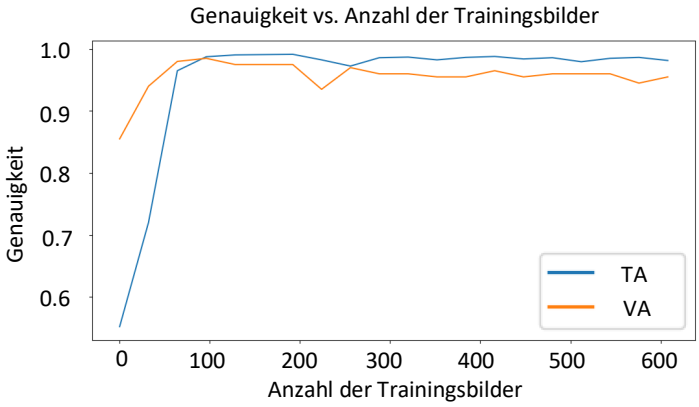
Trainings- und Testbilder aufgeteilt. Danach habe ich ein neues TensorFlow-Modell (Screenshot 1) erstellt und einen Code geschrieben, der mithilfe der TensorFlow-Datagenerator Funktion das Modell trainiert. Während des Trainings habe ich die Trainingsgenauigkeit, von 0 (kein Bild richtig) bis 1 (alle Bilder richtig), des Modells aufgezeichnet und in einem Diagramm in Abhängigkeit der Menge der Trainingsbilder dargestellt (Diagramm 1 & 2). **In den Diagrammen kann man sowohl die Trainingsgenauigkeit in blau (TA= Trainings Accuracy), also welcher Anteil der Trainingsbilder richtig vorhergesagt wird, als auch die Testgenauigkeit in orange (VA: Validation Accuracy) mit Bildern, die das Modell zuvor noch nicht gesehen hat auf der y-Achse, in Abhängigkeit zu der Menge der Trainingsbilder auf der x-Achse.** Ich habe diesen Test zwei Mal durchgeführt. Beim ersten Mal stieg die Genauigkeit sehr schnell und blieb danach bei fast 1, was bedeutet, dass alle Bilder richtig erkannt werden. Beim zweiten Test blieb die Genauigkeit anfangs länger bei 0.5, stieg danach

Programmcode für neues TensorFlow-Modell

```
12  model = Sequential()
13  model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
14  model.add(MaxPooling2D(32))
15  model.add(Flatten())
16  model.add(Dense(128, activation="relu"))
17  model.add(Dense(512, activation="relu"))
18  model.add(Dense(256, activation="relu"))
19  model.add(Dense(128, activation="relu"))
20  model.add(Dense(1, activation="sigmoid"))
21
22  model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

(Screenshot 1)

Testdurchlauf 1

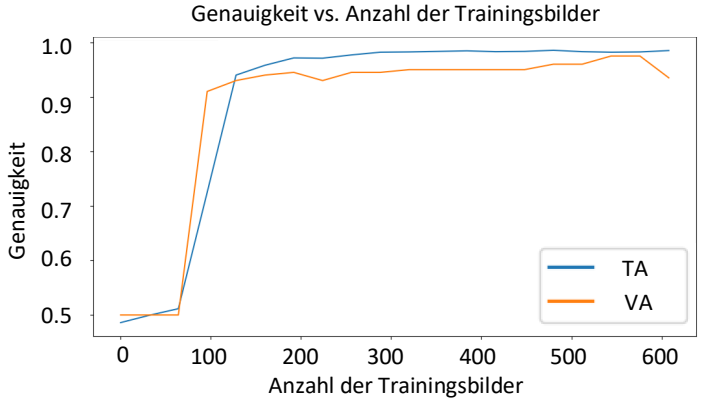


Trainingsgenauigkeit: 100%

Testgenauigkeit: 98%

(Diagramm 1)

Testdurchlauf 2



Trainingsgenauigkeit: 100%

Testgenauigkeit: 95%

(Diagramm 2)

2.2.2 TensorFlow Modell

Für die Wolferkennung habe ich mit der Python-Library TensorFlow ein Modell erstellt. Ich habe mich dafür entschieden, ein für Bilderkennung optimiertes Modell zu nehmen und dieses nur minimal für meine Zwecke anzupassen. Zusätzlich habe ich noch ein eigenes Modell erstellt und dieses ebenfalls trainiert.

2.2.3 Herunterladen der Trainingsbilder

Als Trainings-Dataset für das Modell habe ich mir das Dataset „Dogs vs Wolves“ von https://www.kaggle.com/datasets/harishvutukuri/dogs-vs-wolves, das Dataset „Cats vs Dogs“ von https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset und das Dataset „White wolf Images“ von https://images.cv/dataset/white-wolf-image-classification-dataset heruntergeladen. Diese habe ich dann in Trainings- und Testbilder unterteilt, damit ich die Genauigkeit des Modells akkurat mit Bildern bestimmen kann, die das Modell noch nie gesehen hat.

2.2.4 Hintergrund entfernen

Um das Modell weiter zu verbessern, habe ich den Hintergrund der Trainingsbilder mithilfe der Library rembg entfernt (Beispiel-Bild: Abb 2). Dadurch bekommt das Modell nur die wesentlichen Informationen, da es nicht zwischen irrelevanten Informationen, wie zum Beispiel dem Hintergrund und wichtigen Informationen wie dem Tier selbst unterscheiden kann.

Bild aus meinem Trainingsset

(Abb. 2)

2.2.5 Hervorheben der Kanten

Aus demselben Grund, aus dem ich den Hintergrund der Bilder entfernt habe, reduziere ich nun die Bilder auf die Kanten der Wölfe, da diese die relevanten Informationen an dem Wolf sind (Beispiel-Bild: Abb. 2). Dies tat ich mit dem Pillow FIND_EDGES Filter.

3 Ergebnisse

3.1 Computerexperimente / Funktionstests zur Analyse

In den Diagrammen kann man sowohl die Trainingsgenauigkeit in blau (TA= Trainings Accuracy), also welcher Prozentsatz der Trainingsbilder richtig vorhergesagt werden, als auch die Testgenauigkeit in orange (VA: Validation Accuracy) auf der y-Achse, in Abhängigkeit zu der Menge der Trainingsbilder auf der x-Achse, also wie viele der Testbilder, welche das Modell noch nicht gesehen hat, richtig vorhergesagt werden.

3.1.1 Versuch Nr. 1: Modell 1 mit Hintergrund

(1) Versuchsbeschreibung

Zuerst habe ich die Genauigkeit des auf Bilderkennung optimierten Modells (Screenshot 2) getestet, indem ich es mit insgesamt 640 Bildern trainierte, die nicht vorher bearbeitet wurden.

(2) Ergebnisse

Das Modell erreichte überwiegend eine VA von ca. 75% bis 85% (Diagramm 3). Dieses ist schon relativ gut, ich möchte jedoch eine konstante Genauigkeit erreichen. Es treten Schwankungen auf, die bei 85% und 60% liegen und klärungsbedürftig sind.

(3) Deutung

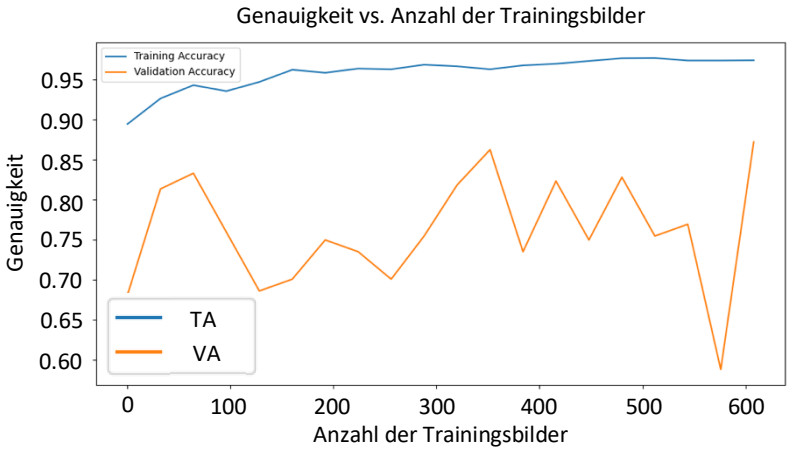
Die Schwankungen erkläre ich mir damit, dass das Modell sich während des Trainings ständig verändert, und es nicht garantiert ist, dass sich das Modell nach jedem Trainingsschritt verbessert. Dies führe ich darauf zurück, dass das Modell mit unterschiedlichen Bildern trainiert wird und das Optimieren auf bestimmte Bilder die gesamte Genauigkeit senken kann.

Auf Bilderkennung optimiertes Model

```
12  base_model = MobileNetV2(weights='imagenet', include_top=False)
13  x = base_model.output
14  x = GlobalAveragePooling2D()(x)
15  x = Dense(1024, activation='relu')(x)
16  x = Dense(1, activation='sigmoid')(x)
17  model = Model(inputs=base_model.input, outputs=x)
18  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

(Screenshot 2)

Testdurchlauf



(Diagramm 3)

Trainingsgenauigkeit: 95%

Testgenauigkeit: 75% - 85%

3.1.2 Versuch Nr. 2: Model 1 ohne Hintergrund

(1) Versuchsbeschreibung

Ich habe dasselbe Modell, wie in Versuch Nr.1 mit Bildern ohne Hintergrund getestet, mit der Erwartung, dass es besser werden müsste, da ich die unwichtigen Informationen im Vorhinein herausfilterte.

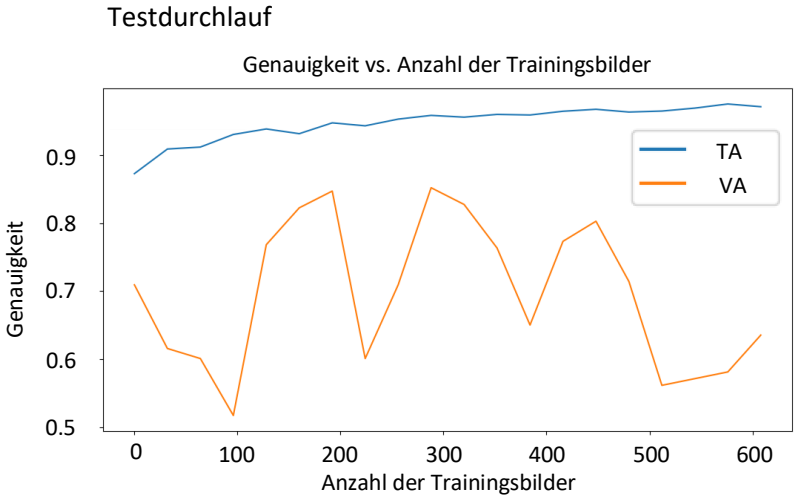
(2) Ergebnisse

Entgegen meinen Erwartungen sank die Genauigkeit des Modells. Die besten Ergebnisse, die das Modell erzielte, sind zwar genauso gut, wie die des ersten Versuches, jedoch gab es deutlich höhere Schwankungen und die Genauigkeit des Modells sank sogar am Ende des Lernprozesses noch unter 60% (Graph: Diagramm 4).

(3) Deutung

Die unerwartete Änderung in der Genauigkeit des Modells lässt sich damit erklären, dass das Modell im ersten Versuch anhand des Hintergrundes erkannte, ob es sich bei dem Bild um einen Wolf oder um einen Hund oder

eine Katze handelt. Hunde oder Katzen werden in der Regel in Innenräumen oder auf Wegen aufgenommen, wohingegen Wölfe im Wald fotografiert werden. Das Modell hat die Wölfe also an ihrer Umgebung erkannt, was jedoch nicht zielführend ist, da dieser, wenn das Programm eingesetzt wird, immer gleich sein wird.



(Diagramm 4)

Trainingsgenauigkeit: 95%

Testgenauigkeit: 60% - 85%

3.1.3 Versuch Nr. 3: Modell 1 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

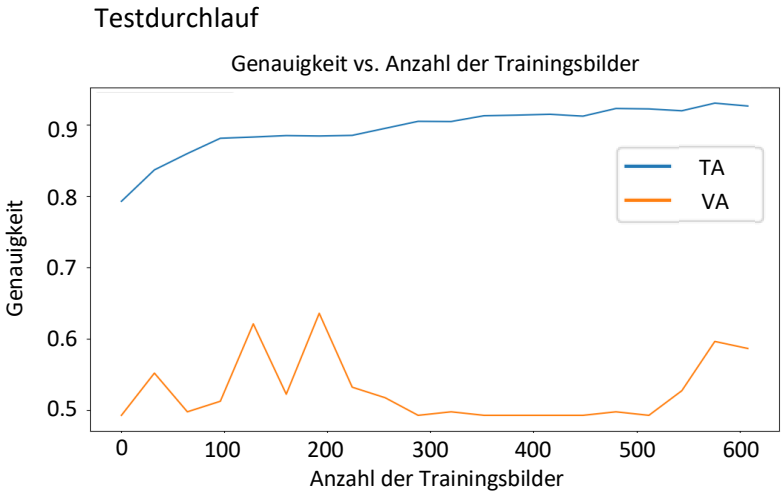
Im dritten Versuch trainierte ich das Modell aus den vorherigen beiden Versuchen mit Schwarz-Weiß-Bildern ohne Hintergrund und mit verstärkten Kanten. Bei diesem Versuch erwartete ich nach dem letzten Versuch keine erfolgreichen Ergebnisse.

(2) Ergebnisse

Wie erwartet ist die Genauigkeit des dritten Versuches sehr schlecht. Bis auf ein paar Glückstreffer blieb das Modell die gesamte Zeit bei 50% Genauigkeit (Graph: Diagramm 5).

(3) Deutung

Die extrem schlechtere Genauigkeit, als beim vorherigen Modell lässt sich damit erklären, dass das Modell die Wölfe anhand von anderen Lichtverhältnissen oder anderen nebensächlichen Informationen erkannt hat, die wenn man das Modell richtig einsetzen würde, nicht da wären.



(Diagramm 5)

Trainingsgenauigkeit: 90%

Testgenauigkeit: 50% - 60%

3.1.4 Versuch Nr. 4: Modell 2 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Im vierten Versuch habe ich ein neues Modell (Screenshot 3) erstellt. Auffällig an diesem Modell ist, dass es viele Hiddenlayers mit jeweils sehr vielen Neuronen hat.

(2) Ergebnisse

Die Genauigkeit des Modells beim vierten Versuch war deutlich besser, als die des letzten Versuches. Die Trainingsgenauigkeit ist jedoch schon ziemlich hoch, wohingegen die Testgenauigkeit des Modells noch sehr niedrig ist und zwischen 60% und 70% schwankt (Diagramm 6).

(3) Deutung

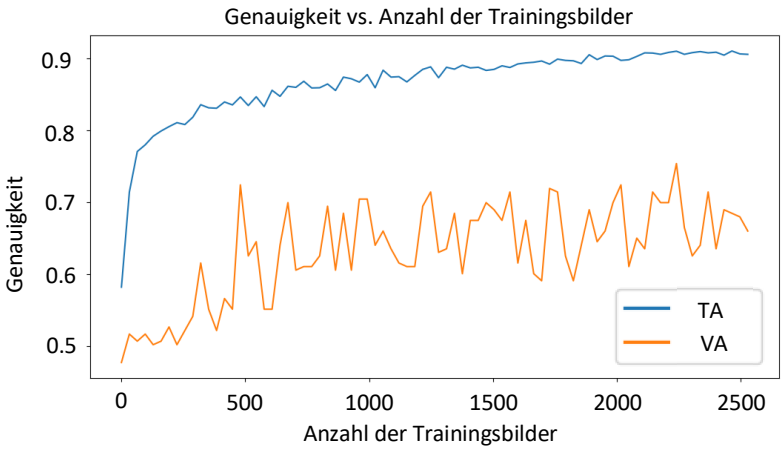
Ich vermute, dass die niedrige Genauigkeit des Modells, durch die hohe Anzahl an Hiddenlayers bedingt ist.

Modell 2

```
4   model = Sequential()
5   model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
6   model.add(MaxPooling2D(32))
7   model.add(Flatten())
8   model.add(Dense(128, activation="relu"))
9   model.add(Dense(512, activation="relu"))
10  model.add(Dense(256, activation="relu"))
11  model.add(Dense(128, activation="relu"))
12  model.add(Dense(1, activation="sigmoid"))
13  model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

(Screenshot 3)

Testdurchlauf



Trainingsgenauigkeit: 90%

Testgenauigkeit: 60% - 70%

(Diagramm 6)

3.1.5 Versuch Nr. 5: Modell 3 mit Schwarz-Weiß-Bildern und verstärkten Kanten:

(1) Versuchsbeschreibung

Um meine Hypothese nach dem letzten Versuch zu begründen, habe ich ein neues Modell (Screenshot 4) mit nur einem einzigen Hiddenlayer und 400 Neuronen mit denselben Testbildern, wie in den letzten Versuchen getestet.

(2) Ergebnisse

Die Genauigkeit des fünften Versuches ist erstaunlich besser, als die des letzten Versuches. Die Testgenauigkeit steigt am Ende sogar auf 85%. Die Trainingsgenauigkeit ist jedoch immer noch zwischen 80% und 85%, das heißt mit mehr Training könnte ich die Genauigkeit des Modells noch weiter steigern (Graph: Diagramm 7).

(3) Deutung

Da das Modell mit weniger Hiddenlayer bessere Ergebnisse erzielt hat als das vorherige, hat sich meine Hypothese bestätigt.

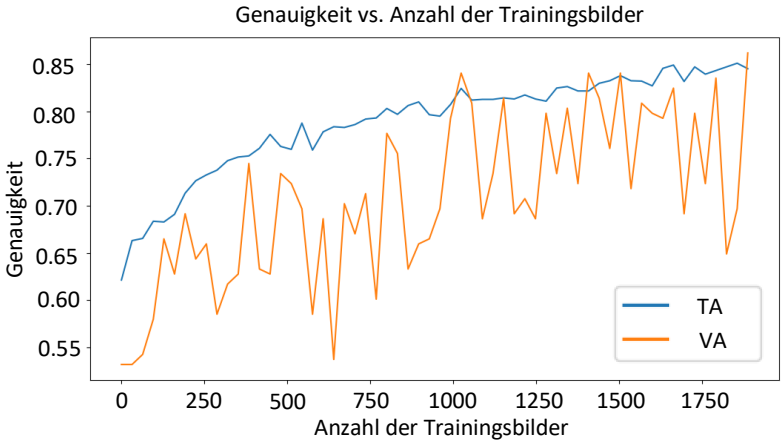
Modell 3

```
model = Sequential()
model.add(Conv2D(30, (3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(400, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

(Screenshot 4)

Testdurchlauf



Trainingsgenauigkeit: 85%

Testgenauigkeit: 75% - 85%

(Diagramm 7)

3.1.6 Versuch Nr. 6: Modell 4 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Da sich meine Hypothese bestätigte, trainierte ich ein Modell (Screenshot 5) mit 128 Neuronen mit denselben Bildern, um zu testen ob dieses bessere Ergebnisse erzielen kann. Ich änderte außerdem die Anzahl der Outputs von zwei zu einem.

(2) Ergebnisse

Entgegen meinen Erwartungen lief der sechste Versuch deutlich schlechter als der fünfte. Die Diagramme beider Versuche ähneln sich sehr, die Testgenauigkeit des neuen Versuches erhöht sich, im Gegensatz zum vorherigen Versuch jedoch nicht wirklich (Graph: Diagramm 8).

(3) Deutung

Die deutliche Verschlechterung des Modells kann entweder durch die kleinere Menge an Neuronen oder durch die neue Anzahl an Outputs bedingt sein.

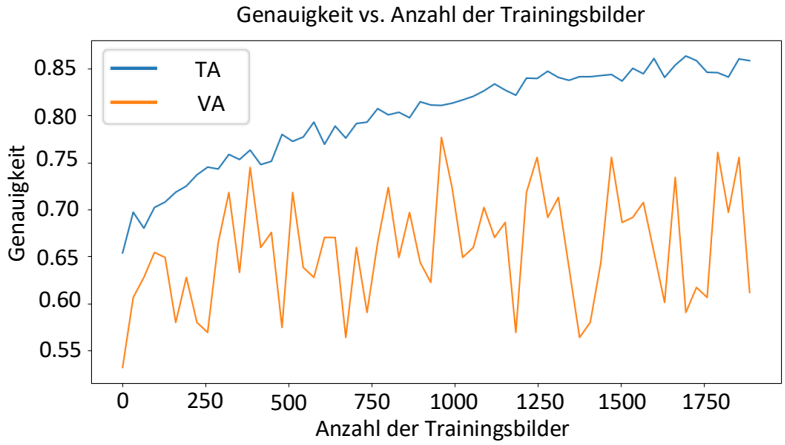
Modell 4

```
model = Sequential()
model.add(Conv2D(30, (3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

(Screenshot 5)

Testdurchlauf



(Diagramm 8)

Trainingsgenauigkeit: 85%

Testgenauigkeit: 60% - 75%

3.1.7 Versuch Nr. 7: Modell 5 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Um zu überprüfen, was die Ursachen für den Genauigkeitseinbruch beim letzten Modell sind, habe ich den letzten Versuch wiederholt und nur die Anzahl der Output Neuronen wieder auf 2 erhöht.

(2) Ergebnisse

Das Modell erzielte keine besseren Ergebnisse als das Vorherige. Die Lernkurve sieht fast identisch aus. (Den Testdurchlauf und den Graphen habe ich aus Versehen gelöscht)

(3) Deutung

Die deutlich schlechtere Genauigkeit bei Versuch sechs sind hauptsächlich der geringeren Anzahl an Neuronen bedingt.

3.1.8 Versuch Nr. 8: Modell 6 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

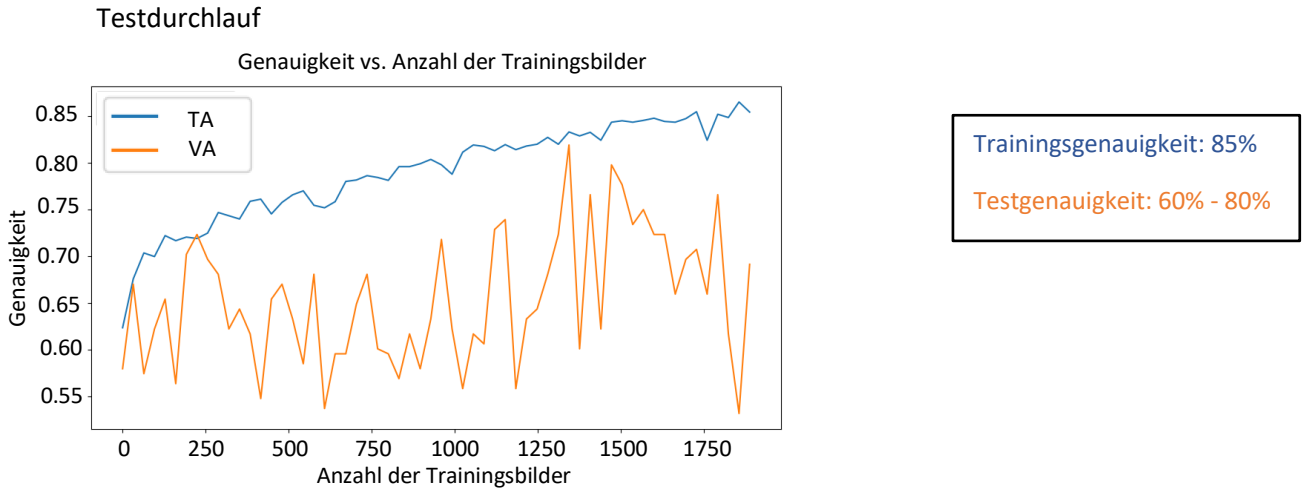
Da 128 Neuronen sehr schlechte Ergebnisse erzielten, habe ich den siebten Versuch mit 256 Neuronen wiederholt.

(2) Ergebnisse

Die Genauigkeit des Models war geringfügig besser, als die des vorherigen Versuches. Die Genauigkeit überstieg 80% jedoch fast nie (Graph: Diagramm 9)

(3) Deutung

Da von den letzten Versuchen diejenigen mit vielen Neuronen am besten abgeschnitten haben, vermute ich, dass ein Hiddenlayer mit vielen Neuronen die beste Genauigkeit erzielt.



Trainingsgenauigkeit: 85%

Testgenauigkeit: 60% - 80%

(Diagramm 9)

3.1.9 Versuch Nr. 9: Modell 3 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Da der dritte Versuch bis jetzt die besten Ergebnisse erzielte, die Genauigkeit jedoch noch nicht am Limit war, habe ich das dritte Modell nochmal mit doppelt so vielen Trainingsbildern trainiert.

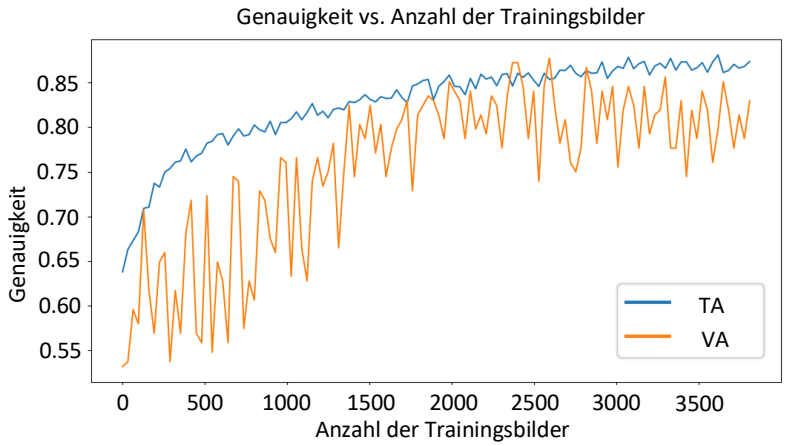
(2) Ergebnisse

Die Genauigkeit des Modells ist erstaunlich gut. Am Ende schwankt sie zwischen 75% und 85% (Graph: Diagramm 10).

(3) Deutung

Da die Genauigkeit in den letzten 500-1000 Trainingsschritten nicht weiter gestiegen ist, glaube ich, dass auch mehr Trainingsbilder die Genauigkeit des Modells nicht weiter verbessern können.

Testdurchlauf



Trainingsgenauigkeit: 87%

Testgenauigkeit: 80%

(Diagramm 10)

3.1.10 Versuch Nr. 10: Modell 7 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Da die Versuche mit einem Layer die besten Ergebnisse erzielten, diejenigen mit wenig Neuronen in diesem Layer jedoch schlechter war, wiederholte ich den vorherigen Versuch mit 1000 statt 400 Neuronen im Hiddenlayer.

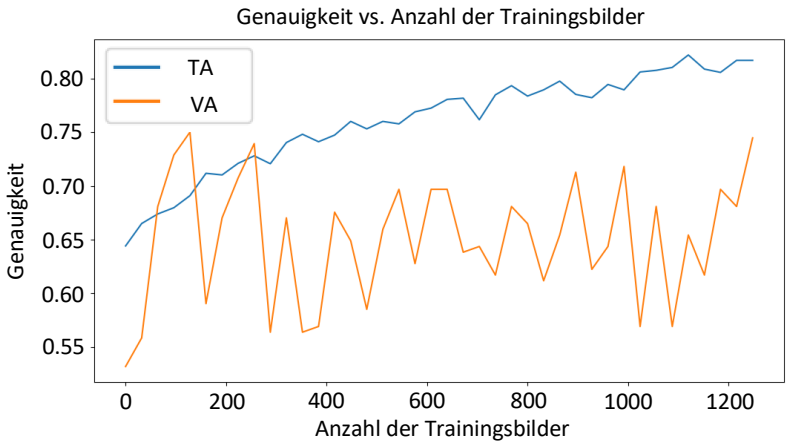
(2) Ergebnisse

Entgegen meinen Erwartungen war die Genauigkeit des 10. Versuches deutlich schlechter als die des 9. Versuches. Die Testgenauigkeit schwankt zwischen 60% und 70% (Graph: Diagramm 11).

(3) Deutung

Die schlechtere Genauigkeit des Modells wird an der großen Menge an Neuronen liegen.

Testdurchlauf



Trainingsgenauigkeit: 80%

Testgenauigkeit: 57% - 75%

(Diagramm 11)

3.1.11 Versuch Nr. 11: Modell 8 mit Schwarz-Weiß-Bildern und verstärkten Kanten

(1) Versuchsbeschreibung

Da die Genauigkeit des letzten Versuches schlechter war, wiederholte ich diesen mit 516 Neuronen im Hiddenlayer.

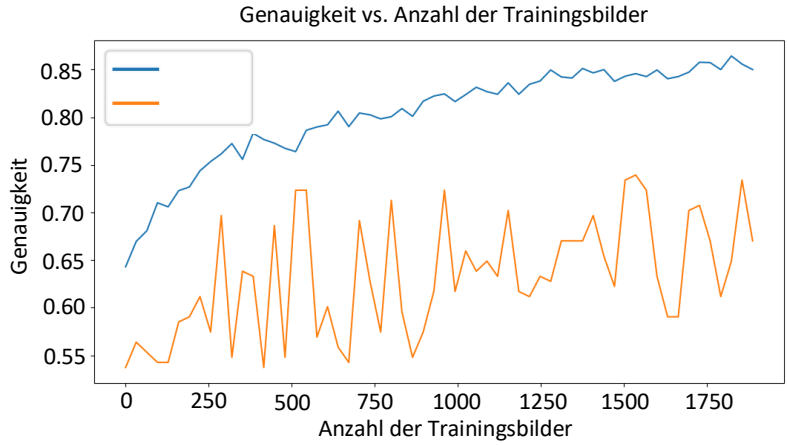
(2) Ergebnisse

Die Genauigkeit des 11. Versuches war minimal besser als die des 10. Versuches. Die Testgenauigkeit schwankt zwischen 60% und 75% (Graph: Diagramm 12).

(3) Deutung

Da der 11. Versuch minimal bessere Ergebnisse erzielte als der 10. Versuch, vermute ich, dass die optimale Anzahl bei 400 Neuronen liegt, da diese Anzahl bisher die besten Ergebnisse erzielte. Die optimale Anzahl wird jedoch nicht deutlich größer als 400 Neuronen sein, weil ansonsten der 11. und 10. Test besser angeschnitten hätten.

Testdurchlauf



Trainingsgenauigkeit: 85%

Testgenauigkeit: 60% - 73%

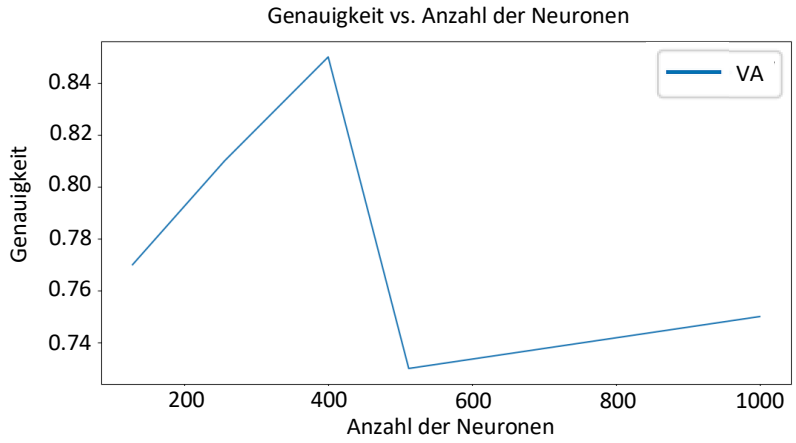
(Diagramm 12)

3.1.12 Zusammenfassung der Ergebnisse

(1) Optimale Anzahl der Neuronen

Wie man in dem Graphen (Diagramm 13) sehen kann, haben 400 Neuronen im Hiddenlayer die besten Ergebnisse erzielt. Die Genauigkeit mit weniger Neuronen ist etwas geringer; die Genauigkeit mit mehr Neuronen ist deutlich geringer. Auffällig ist jedoch, dass die Genauigkeit mit steigender Anzahl an Neuronen über 400 wieder steigen. Die Ursache den geringfügigen Anstieg werde ich bis zum Wettbewerb auch noch überprüfen.

Anzahl der Neuronen im Vergleich zur Genauigkeit



(Diagramm 13)

4 Ergebnisse und Zielerreichung

4.1 Ergebnisdiskussion

Testdurchlauf	Trainingsgenauigkeit	Testgenauigkeit
TensorFlow Test	100%	95% - 98%
3.3.1 Versuch Nr. 1: Modell 1 mit Hintergrund	95%	75% - 85%
3.3.2 Versuch Nr. 2: Modell 1 ohne Hintergrund	95%	60% - 85%
3.1.3 Versuch Nr. 3: Modell 1, SW-Bildern u. verstärkten Kanten	90%	50% - 60%
3.1.4 Versuch Nr. 4: Modell 2, SW-Bildern u. verstärkten Kanten	90%	60% - 70%
3.1.5 Versuch Nr. 5: Modell 3, SW-Bildern u. verstärkten Kanten	85%	75% - 85%
3.1.6 Versuch Nr. 6: Modell 4, SW-Bildern u. verstärkten Kanten	85%	60% - 75%
3.1.8 Versuch Nr. 8: Modell 6, SW-Bildern u. verstärkten Kanten	85%	60% - 80%
3.1.9 Versuch Nr. 9: Modell 3, SW-Bildern u. verstärkten Kanten	87%	80%
3.1.10 Versuch Nr. 10: Modell 7, SW-Bildern u. verstärkten Kanten	80%	57% - 75%
3.1.11 Versuch Nr. 11: Modell 8, SW-Bildern u. verstärkten Kanten	85%	60% - 73%

Die beste von mir erzielte Genauigkeit von 80% - 85% ist für erste Testdurchläufe bereits zufriedenstellend, aber motivieren mich noch weiterzuarbeiten!

Meine Kriterien zum Vergleichen der einzelnen Netze sind ebenfalls gut, da die bloße Genauigkeit nicht ausreicht, um neuronale Netze zu vergleichen, da die Menge der Trainingsdaten auch eine große Rolle spielt. Wenn man eine hohe Genauigkeit erreichen möchte, muss man auf zwei Dinge gleichzeitig achten: das Netzwerk und die Trainingsdaten.

4.2 Zielerreichung

Mein Ziel war es ein neuronales Netz mit Python zu programmieren, welches automatisch Wölfe erkennen kann. Dies ist mir mit einer zufriedenstellenden Genauigkeit (siehe 4.1) gelungen. Ich bin motiviert, diese noch weiter zu steigern. Ich habe bereits eine große Menge an Trainings- bzw. Testbildern heruntergeladen und ein neuronales Netz mit TensorFlow programmiert und dieses mit den Trainingsbildern trainiert. Weiterführend habe ich schon sehr viele Computerexperimente durchgeführt (siehe 3.1.1 bis 3.1.11) und konnte die einzelnen Tests durch Auswertungen miteinander vergleichen und den Aufbau des neuronalen Netzes immer weiter verbessern. Die Lösung für ein Problem im Zusammenleben zwischen Menschen, Nutztierhaltung und Wölfen könnte in der Nutzung von sinnvollen, KI nutzenden Schutzmaßnahmen und nicht im willkürlichen Töten von Tieren, die einen Bestandteil unseres regionalen Lebensraumes sind, und historisch gesehen auch immer schon waren, liegen.

4.3 Ausblick

Weitergehend werde ich das neuronale Netz noch weiter verbessern. Anschließend werde ich die Hardware bauen. Da der relevanteste erste Schritt in der Zielsetzung das Programmieren eines neuronalen Netzes ist, habe ich dies zuerst gemacht und habe vor, mich zukünftig dem zweiten Schritt in der Zielsetzung mit der Konstruktion eines Kamerasystems auseinanderzusetzen. Das Kamerasystem wird so gebaut sein, dass es nur Fotos aufnimmt, wenn ein installierter Bewegungsmelder eine Bewegung wahrnimmt. Diese Bilder werden dann anschließend in das neuronale Netz eingespeist, welches erkennt, ob in dem Bild ein Wolf erkennbar ist. Danach werde ich das Programm mit selbst aufgenommenen Bildern trainieren, da diese vielfältiger sind und besser geeignet, da das letztendliche System nur solche Bilder in natürlich Umgebung verarbeiten wird. Ich werde außerdem eine Funktion in den Code einbauen, dass ein Verantwortlicher benachrichtigt wird, wenn ein Wolf erkannt wird. Die Kamera wird außerdem immer, wenn sie glaubt, einen Wolf erkannt zu haben, dem Verantwortlichen ein Video des Tieres schicken, sodass dieser sehen kann, ob ein Wolf gesichtet wurde oder es ein Fehlalarm war. Ich werde das Programm deswegen auch so konfigurieren, dass es auch eine Nachricht an den Verantwortlichen sendet, wenn es sich nicht komplett sicher ist, da dieser im Zweifel immer entscheiden kann, ob ein Wolf gesehen wurde oder nicht. Der Nutzen für den Verantwortlichen ist also, bei sehr geringem Aufwand sehr hoch.

Als längerfristiges Ziel habe ich vor die Wölfe von meinem System in freier Natur zu trainieren, d.h. ich möchte versuchen ihnen beizubringen, dass sie keine Nutztiere des Menschen reißen sollen.

5. Literaturverzeichnis

-danielgatis (Entwickler) (12.2023 aktualisiert): rembg https://github.com/danielgatis/rembg/releases (heruntergeladen am 2.1.2024)

-Grit Klempow (Autor) (25.01.2023): Lauter Ruf nach Wolfsabschüssen Stader Tageblatt

-Google Brain Team (Entwickler) (4.9.2023 aktualisiert): TensorFlow https://de.wikipedia.org/wiki/TensorFlow (heruntergeladen am 2.1.2024)

-harishvutukuri (Entwickler) (2020): Dogs vs wolves https://www.kaggle.com/datasets/harishvutukuri/dogs-vs-wolves (heruntergeladen am 04.01.2024)

-Imagenet (Entwickler) (keine Angabe): https://images.cv/dataset/white-wolf-image-classification-dataset (heruntergeladen am 04.01.2024)

- Institut für Ökologie und Evolution, Universität Bern (keine Angabe): Abb. 1 https://infoplattform-grossraubtiere.ch/bei-ihrer-rueckkehr-achten-woelfe-auf-das-beuteangebot/

-Intel, Willow Garage (Entwickler) (18.12.2023 aktualisiert): OpenCV https://de.wikipedia.org/wiki/OpenCV (heruntergeladen am 2.1.2024)

-Secret Labs AB (Entwickler) (17.10.2023 aktualisiert): Python Imaging Library https://en.wikipedia.org/wiki/Python_Imaging_Library (heruntergeladen am 2.1.2024)

-shaunthesheep (Entwickler) (2020): Microsoft Cats vs Dogs https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset (heruntergeladen am 04.01.2024)

-Zörb, Paul (Entwickler) (2024): mein GitHub-Projekt: https://github.com/Paul193247/Jugendforscht-Wolfserkennung

-Zörb, Paul (Ersteller) (2024): Diagramm 1

-Zörb, Paul (Ersteller) (2024): Diagramm 2

-Zörb, Paul (Ersteller) (2024): Diagramm 3

-Zörb, Paul (Ersteller) (2024): Diagramm 4

-Zörb, Paul (Ersteller) (2024): Diagramm 5

-Zörb, Paul (Ersteller) (2024): Diagramm 6

-Zörb, Paul (Ersteller) (2024): Diagramm 7

-Zörb, Paul (Ersteller) (2024): Diagramm 8

-Zörb, Paul (Ersteller) (2024): Diagramm 9

-Zörb, Paul (Ersteller) (2024): Diagramm 10

-Zörb, Paul (Ersteller) (2024): Diagramm 11

-Zörb, Paul (Ersteller) (2024): Diagramm 12

-Zörb, Paul (Ersteller) (2024): Diagramm 13

-Zörb, Paul (Ersteller) (2024): Screenshot 1

-Zörb, Paul (Ersteller) (2024): Screenshot 2

-Zörb, Paul (Ersteller) (2024): Screenshot 3

-Zörb, Paul (Ersteller) (2024): Screenshot 4

-Zörb, Paul (Ersteller) (2024): Screenshot 5

6. Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich bei meinem Projekt unterstützt und motiviert haben. Zuerst gebührt mein Dank Herrn Prof. Dr. Carmesin, der mein Projekt betreut und mir bei der Organisation des Projektes und dem Setzen der Zwischenziele geholfen hat. Ein besonderer Dank gilt Frau von Bagen, die mir mit viel Geduld, Interesse und Hilfsbereitschaft zur Seite stand. Zu guter Letzt möchte ich mich bei meiner Familie bedanken, die stets ein offenes Ohr für mich hatte.