F A C H H O C H S C H U L E W E D E L

- University of Applied Science -

BACHELOR-THESIS

in der Fachrichtung

Medieninformatik

Thema:

AI Acceleration in Raytracing

Eingereicht von:	Jonas Sorgenfrei Minf101767 Op de Wisch 1 25436 Moorrege Tel. (+49) 04122 / 83420 E-Mail: minf101767@fh-wedel.de
Erarbeitet im:	7. Semester
Abgegeben am:	
Referent (FH Wedel):	Prof. Dr. Christian-A. Bohn Fachhochschule Wedel Feldstraße 143 22880 Wedel Tel. (+49) 04103 / 804840 E-Mail: bo@fh-wedel.de

AI Acceleration in Raytracing

Forschungsarbeit zu Beschleunigungen des Raytrace-Prozesses durch Maschinelles-Lernen

2019, Bachelor-Thesis von Jonas Sorgenfrei, Fachhochschule Wedel

Zusammenfassung:

In dieser Arbeit wird die Anwendbarkeit von maschinellem Lernen auf den geometrischen Abschnitt des Raytracing/Pathtracing untersucht.
Es wird erforscht welchen Einfluss ein maschinell trainiertes Modell auf den Raytrace-Algorithmus nehmen kann und welche Genauigkeit dieses zurückliefert. Im Anschluss wird diskutiert ob sich anhand von Zeit/Genauigkeit eine praktische Anwendbarkeit eines solchen Modells im Raytrace-Prozess realisieren lassen würde und welche Vor- und Nachteile daraus entstehen würden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listing-Verzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ML	Maschinelles Lernen (engl. Machine Learning)
KI (engl. AI)	Künstliche Intelligenz (engl. Artificial Inteligence)
PBR	Physically Based Rendering
PBRT	Physical Based Ray-Tracer; verweis auf den in [1] beschriebenen Raytracer
RMSE	Root Mean square error (dt. Wurzel der mittleren quadratische Abweichung)

1.

Introduction

Im Rahmen dieser Bachelor-Thesis soll erforscht werden in wie weit sich maschinelles Lernen in den Ray-Trace Prozess integrieren lässt. Insbesondere soll die Geschwindigkeit des Raytracing-Vorgangs in Verbindung mit der Genauigkeit des resultierenden synthetischen Bildes analysiert werden.
Im Gegensatz zu bisherigen Ansätzen der Kombination von maschinellem Learning in den Ray-Trace Prozess, soll in dieser Arbeit nicht die Anwendung des maschinellen Lernens auf das Ausgabe-Bild untersucht werden [2], sondern dies direkt in dem Geometrie-Teil (Schnitt-Test) des Ray-Tracing-Prozesses Anwendung finden.
Insbesondere soll untersucht werden ob sich ein Modell auf eine Szene trainieren lässt um im Folgenden zum Beispiel Schattenstrahlen schnell bestimmen zu können.
<TODO>

1.1. Motivation

<TODO>

1.2. Structur

<TODO>

2.

Basics

Dieser Abschnitt führt in die Thematik der synthetischen Bilderstellung mithilfe von Raytracing ein und gibt ein Überblick über das Thema des maschinellen lernen. Es werden Begrifflichkeiten definiert, die in den folgenden Hauptteilen der Arbeit benutzt werden.

2.1. Raytracing

<TODO>

2.2. Phyiscal Based Rendering

<TODO>

2.3. Acceleration Alogrithms

<TODO>

2.3.1. Bounding Volume Hierarchies

<TODO>

2.3.2. Kd-Trees

<TODO>

2.4. Machine Learning

Dieser Abschnitt beschreibt was maschinelles Lernen ist und definierte bestimmte Begriffe die im Laufe der weiteren Arbeit wiederverwendet werden.

Labels & Features

Bei Features handelt es sich um die Input Parameter. Diese werden beim

maschinellen Lernen als Eingangsparameter genutzt, um damit zu arbeiten. Als Label wird die die Ausgabe/die Ausgaben bezeichnet die geliefert wird, ausgehend von den (Input)-Features.
Dies kann mit einer Funktion $f(x_i,x_{ii}) = y$ verglichen werden, bei der x_i und x_{ii} die Features sind und y das Label. Features und Labels sind die Datenstruktur die verwendet wird um ein Netzwerk zu Trainieren und nach abgeschlossenem Training stellen die Features die (unbekannten) Parameter dar, mit denen das Netzwerk arbeiten kann.
Generell gesagt steigt je nach Komplexität des Problems auch die Anzahl an Features.

Modell

Ein Modell beschreibt die Beziehung zwischen Features und Labels. Ein Model wird im Rahmen des maschinellen Lernens trainiert mit Features die ein Label besitzen und lernt dabei diese Beziehung. Anhand der erlernten Beziehung kann das Modell Schlussfolgerungen bei unbekannten/neuen Features ziehen, die kein Label besitzen. Das Erlernen der Beziehung geschieht durch die Minimierung des Loss (s.u.) und wird als empirische Risiko Minimierung bezeichnet. [3]

Es gibt Verschiedene Arten von Modellen, dabei werden im Rahmen dieser Arbeit hauptsächlich 2 Typen von Modellen unterschieden:

Regression/Klassifikation

Ein lineares Regressions-Modell gibt kontinuierliche Werte zurück während ein Klassifikations-Modell diskrete Werte (z.B. Kategorien) zurück gibt. Ein Klassifikation Modell basiert in der Regel auf logistischer Regression, dabei wird eine Sigmoid Funktion ($y = \frac{1}{1+e^{-\sigma}}$) verwendet um die Prognose eines linearen Modelles (σ) in den Werte Bereich [0,1] zu bringen, was als Wahrscheinlichkeitswert interpretiert werden kann.

Außerdem wird in dieser Arbeit zwischen Regressions-/Klassifikations-Modell und Neuronalem Netzwerk unterschieden.
Bei einem linearen Modell werden die einzelnen Parameter mithilfe von angelernten Skalaren verbunden und die Beziehung definiert. Hierbei muss die Zusammengehörigkeit von Werten vordefiniert sein (z.B. Feature Kreuzungen (Multiplikationen), Funktionen wie Sinus/Cosinus, Potenzfunktionen).

Dies ist in der Regel die schnellste Möglichkeit des Maschinellen Lernens jedoch wird dies bei einer großen Anzahl an Features sehr schnell zu Komplexen/nicht überschaubaren Zusammenhängen die schwer manuell modellierbar sind.
An dieser Stelle können Neuronale Netze ein Vorteil bieten. <TODO NN>

Trainings-Set/Validation-Set/Test-Set

Um sicherzustellen, dass ein Modell generallisierbar bleibt werden die Testdaten in 3 Teilmengen unterteilt, dabei wird das Training-Set direkt zum Trainieren des Models genutzt, das Validation-Set wird genutzt um ein Trainiertes Model zu evaluieren und damit dem Overfitting (Spezialisierung des Models bezüglich der Testdaten) entgegen zu wirken und das Test-Set wird nach abgeschlossenem Training zum manuellen evaluieren des Models in einem 2. Schritt genutzt. Hierbei beinhaltet die Schnittmenge aus Training- und Validation-Set generell 80-90% der Testdaten während das Test-Set den kleineren Anteil der verbleibenden 10-20% darstellt.

Loss

Als Loss wird der Verlust durch schlechte Prognosen des Modells bezeichnet, dieser kann beim Training mithilfe der etikettierten Trainings- und Validations-Daten bestimmt werden. Das Ziel des Trainings ist es ein geringer Verlust über die den Durschnitt aller Werte zu haben.

Squared Loss (L2 loss) / Mean Square Error

Beim Squared Loss handelt es sich um eine Verlust Funktion, die die Differenz zwischen dem Label und der Prognose quadriert.
Die mittlere quadratische Abweichung (engl. Mean Square Error; MSE) bezeichnet den durchschnittlichen quadratischen Verlust pro Stichprobe über den gesamten Datensatz. <TODO QUELLE> Die mittlere quadratische Abweichung ist folgendermaßen definiert:

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prognose(x))^2$$

Formel 2.1 Mean Square Error

D beschreibt den Datensatz mit N Stichproben $\{(x_0,y_0),\ldots,(x_N,y_N)\}$ wobei x die Menge an Features und y das Label darstellt.
Squared Loss wird als Loss Funktion für lineare Regressions Modelle genutzt.

Log Loss

Log Loss wird als Loss Funktion für logistische Regression wie bei Klassifizierern verwendet.
Die Log Loss Funktion ist folgendermaßen definiert:

$$LogLoss = \sum_{(x,y)\epsilon D} -y * log(prognose(x)) - (1-y) * \log(1 - prognose(x))$$

Formel 2.2 Log Loss

D beschreibt den Datensatz mit N Stichproben $\{(x_0,y_0),\ldots,(x_N,y_N)\}$ wobei x die Menge an Features und y das Label darstellt. Hierbei ist zu beachten, dass das Label y entweder 1 oder 0. Die Prognose (*prognose(x)*) entspricht einem Wert zwischen 0 und 1.

Konvergenz Kurven

<TODO>

Optimizer

Die Aufgabe des Optimierung Algorithmus ist es den Loss des Modells zu verringern und dadurch bessere Prognosen zu garantieren. Dieser wird beim Training des Models angewendet. Optimierung Algorithmus sind in der Regel spezifische Implementation des Gradient Descent Optimizers.

Gradient Descent Optimizer

Der Gradient Descent Optimizer ist ein Optimizer für generell konvexe Problems. Als Gradient wird hier ein Vektor von partiellen Ableitungen der (unabhängigen) Gewichte des Modells bezeichnet. Mithilfe dieses kann die Richtung der größten bzw. kleinste Steigung der Funktion ermittelt werden. Der Gradient wird als ∇f notiert ist für eine gegebene Funktion *f(x, y)* der Vektor der partiellen Ableitungen:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right)$$

Formel 2.3 Gradient, partielle Ableitungen der Gewichte

Der Gradient Descent Optimizer nutzt die negative Richtung des Vektors (Richtung der geringsten Steigung) sowie dessen Betrag multipliziert mit der Learning Rate (s.u.) um schnellst möglich den Loss zu minimieren.

Learning Rate

Bei der Learning Rate handelt es sich um die Schrittgröße mit der ein Optimizer die Loss-Funktion minimiert. Beim Gradient Descent Optimzier wird z.B. die Learning Rate auf den Betrag des Gradienten multipliziert, das skalare Ergebnis dient als Faktor für die Optimierung in Richtung des Gradienten.
Generell gilt, dass eine sehr kleine Learning Rate eine längere Laufzeit beim Training veranlasst, während eine zu hohe Learning Rate dazu führen kann, dass das Optimum nicht erreicht wird. Jedes Problem hat eine eigene optimale Learning Rate (für eindimensionale Probleme ist dies die Inverse 2. Ableitung von f(x); $\frac{1}{f(x)"}$).

Batch/Batch Size

Ein Batch beschreibt die Menge an Stichproben die in einer Iteration des Model Trainings genutzt werden soll. Bei der Batch Size handelt es sich um die Anzahl der Stichproben in der oben beschriebenen Menge. Generell gilt, dass es effizienter ist, den Loss mithilfe eines kleineren Batches zu berechnen, idR besteht dieser aus 10 bis 1000 Stichproben.

Regularization

Generell ist ein größeres/komplexeres Modell stärker auf die Trainingsdaten spezialisiert, wodurch es schnell zu Overfitting kommen kann. Um dieser Spezialisierung entgegen zu wirken, wird Regularisierung (engl. Regularization) genutzt um je nach Komplexität des Modells, das Modelltraining anzupassen ähnlich wie die Loss-Funktion. Dafür gibt es verschiedene Arten der Regularisierung:

> **L1 regularization** bestimmt die Komplexität des Modells anhand der Summe der absoluten Gewichte. Dadurch werden Features, die irrelevant oder kaum relevant sind aus dem Modell entfernt.
>
> **L2 regularization** bestimmt die Komplexität des Modells anhand der Summe der quadrierten Gewichte. Insbesondere wirkt es bei Außreiser, dass diese nicht so stark auf das Modell wirken.

Dropout Regularization bewirkt, dass eine bestimmte zufällige Auswahl an Layern eines Netzwerkes (Neural Network) in einem einzelnen Gradientenschrittes entfernt werden.

Early Stopping ist eine Methode, bei der das Training beendet wird, bevor der Loss des Validation-Datensatzes wieder zu steigen beginnt.

Mithilfe der Regularisierung Rate (λ) kann die Stärke der Regularisierung Funktion auf das Modelltraining beeinflusst werden sodass die Loss Gleichung folgendermaßen ändert. Dies wird als „Strukturierte Risiko Minimierung“ bezeichnet.

$$minimize(loss\ function + \lambda(regularization\ function))$$

Formel 2.4 Strukturierte Risiko Minimierung

Threshold/Confusion Matrix

Klassifikation Modelle liefern als Prognose einen Prozentwert zurück. Um diesen zu einer Klassifikation zwischen zwei Kategorien zu konvertieren, wird ein Threshold Value genutzt, mithilfe dessen die Prognose einer der beiden Kategorien zugeordnet werden kann. Durch die Zuordnung mithilfe des Threshold Wertes kann es zu folgenden Zuständen kommen, die in der Folgenden Confusion Matrix gezeigt werden.

Tabelle 1 Confusion Matrix

True Positive (TP): Das Modell prognostiziert korrekt ein positives Ergebnis.	**False Positive (FP):** Das Modell prognostiziert ein positives Ergebnis, welches jedoch nicht ein negatives Ergebnis ist.
False Negative (FN): Das Modell prognostiziert ein negatives Ergebnis, welches jedoch ein positives Ergebnis ist.	**True Negative (TN):** Das Modell prognostiziert korrekt ein negatives Ergebnis.

Evaluation

Zur Evaluation des Trainierten Models können verschiedene Metriken verwendet

werden. Im folgendem werden die in dieser Arbeit verwendeten Metriken vorgestellt und kurz erläutert.

Accuracy/Precision/Recall/ROC Curve/AUC

Die Genauigkeit (engl. Accuracy) ist definiert als der Prozentsatz, den das Modell richtig prognostiziert.

$$Accuracy = \frac{Anzahl\ der\ korrekten\ Prognosen}{Gesammtanzahl\ der\ Prognosen} = \frac{TP + TN}{TP + FP + TN + FN}$$

Formel 2.5 Accuracy

Bei der Nutzung von Accuracy ist zu beachten, dass diese Metrik nicht stabil gegenüber nicht ausbalancierten Datensätzen ist. (z.B. Klasse 1 zu 2% vertreten Klasse 2 zu 98% vertreten).

Die Präzision (engl. Precision) beschreibt den Anteil der positiv prognostizierten Stichproben die korrekt sind (Precision = TP / (TP+FP)).

Die Treffer-Quote (engl. Recall) beschreibt den Anteil der positiven Stichproben die korrekt prognostiziert wurden. (Recall = TP / (TP+FN)).

Eine ROC-Kurve (Grenzwertoptimierungskurve; engl. receiver operating characteristic curve) kann genutzt werden um die Performanz eines Klassifikations-Modells bei allen Klassifikation Thresholds zu plotten. Dabei werden die Parameter True Positive Rate (siehe Recall) und False Positive Rate (FPR = FP / (FP + TN)) geplottet. Die Fläche unter der geplotteten Kurve wird als AUC (Area under the Roc Curve bezeichnet), dieser Wert kann als weitere Metrik genutzt werden um die Performanz des Modells zu bestimmen, bei einem Modell, dessen Prognosen immer falsch sind, ist die der AUC-Wert 0. Bei einem Modell, dessen Prognosen immer richtig sind, ist der AUC-Wert 1, in allen anderen Fällen liegt der Wert zwischen diesen beiden Werten [0,1].

2.5. Tensorflow

Im Rahmen der Forschungsarbeit dieser Thesis wird das Tensorflow Framework genutzt.

Hierbei handelt es sich um eine Bibliothek zur datenstromorientierten Programmierung, welche einer der meist verwendeten Bibliotheken ist, die im Rahmen des Maschinellen Lernens verwendet wird. Die Bibliothek bietet gleichzeitig Low-Level und High-Level Funktionen zum Definieren, Trainieren und Ausführen von Modellen, außerdem werden Möglichkeiten des Exportes und Importes geboten.

In der folgenden Grafik von der Tensorflow Website wird die Hierarchie des Tensorflow Toolkits gezeigt.

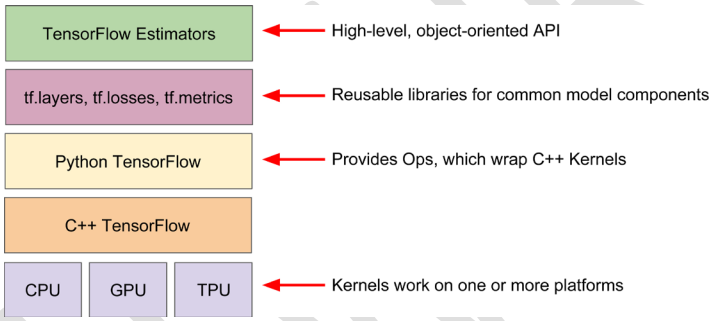

Abbildung 1 Tensorflow Framework Hierarchie

Im Rahmen dieser Arbeit wird hauptsächlich auf der Ebene der TensorFlow Estimators gearbeitet und es werden die vordefinierten Modelle (linear regressor, linear classifier & neural networks) genutzt.

Der folgende Code Ausschnitt zeigt wie Tensorflow in Python verwendet wird in dem ein Linearer Regressor erstellt und Trainiert sowie ausgewertet wird.

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Regressors
4.  estimator = tf.estimator.LinearRegressor(feature_columns)
5.
6.  # trainieren des Models mit Beispieldaten
7.  estimator.train(input_fn=train_input_fn, steps=2000)
8.
9.  # mithilfe des trainierten Models Prognosen erstellen
10. Predictions = estimator.predict(input_fn=predict_input_fn)
```

Listing 2.1 Tensorflow Erstellen, Trainieren und nutzen eines Linearen Regressors

In sämtlichen Testläufen wurde die Tensorflow Version 1.14.0 in der GPU variante genutzt.

3.

Hardware

Sämtliche Training und Testläufe wurden auf einem Computer durchgeführt, welcher folgende Spezifikationen aufweist.

Tabelle 2 Verwendete Computer Hardware

CPU	AMD Ryzen Threadripper 2950X 16-Core Processor 3,50Ghz
GPU	NVIDIA GeForce RTX 2080 Ti Dedizierter GPU-Speicher: 11GB
Arbeitsspeicher	32 GB
Sekundär Speicher	Samsung SSD 970 Pro 512GB Samsung SSD 860 EVO 1TB
Betriebssystem	Windows 10 – Pro 64 Bit

Wie im 2. Abschnitt beschrieben wurde die GPU Variante von dem Tensorflow Framework verwendet. Während sämtlichen konnte folgende Beobachtungen gemacht werden.

Bei jedem Training wurden lediglich 12% der GPU Kapazität genutzt, die meiste Zeit befand sich die GPU im Copy-Modul, bei genaueren Analyse kann erkannt werden, dass der dedizierte GPU-Speicher eine maximale Auslastung hat und sich hier scheinbar das Bootleneck befindet. Die folgende Abbildung zeigt die einzelnen Auslastungen der Grafikkarten Komponenten, hier ist klar die starke Auslastung des dedizierten GPU Speichers zu erkennen. Im direkten Vergleich konnte jedoch festgestellt werden, dass das Training auf der GPU in der og. Systemkonfiguration trotz dieses Bottlenecks schneller ist als auf der CPU.

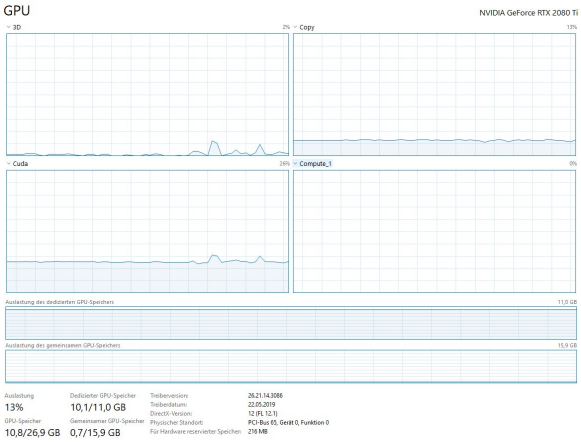

Abbildung 2 GPU Auslastung beim Modell-Training

Als Umgebung wurde Python gewählt, dies stellte sich jedoch an einigen Stellen als problematisch dar, da Python Script zur Laufzeit ausgeführt werden und nicht kompiliert werden mit den entsprechenden Compiler-Checks, sodass an einigen Stellen semantische Fehler oder fehlende Bibliotheks-Importe dazu führten, dass ein Training nach sehr langer Berechnungszeit nicht komplett evaluiert werden konnte. Insbesondere aus diesem Grund wurde direkt nach dem Schritt des Modelltrainings dieses Exportiert, sodass dieses in einem weiteren Schritt manuell Evaluiert werden kann und das trainiert Modell nicht verloren geht.

4.

Dataset

Dieser Abschnitt beschreibt die verwendete Szene für die Testdaten und die darin enthaltene Geometrie sowie die relevanten Szene Parameter.
Im weiteren Verlauf wird beschrieben wie die Testdaten in Features & Label konvertiert werden und der Datensatz in Trainings-, Validation- und Testdatensatz aufgeteilt wird und die Verteilung innerhalb dieser wird analysiert.

4.1. Testszene & Geometrie

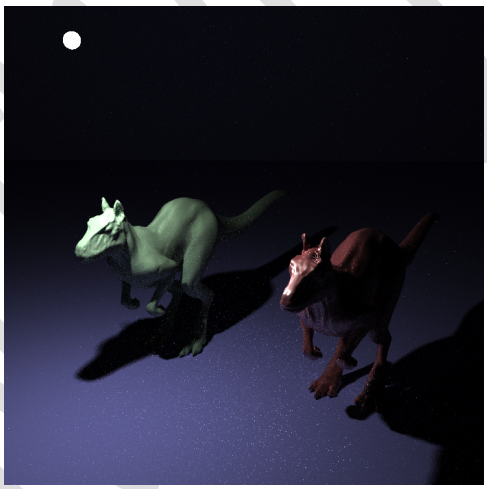

Abbildung 3 Killeroo-Simple Scene

Bei der verwendeten Testszene, aus der der Datensatz für diese Arbeit exportiert wurde handelt es sich um die “Killeroo”-Simple Scene des PBRT-Renderers. Es handelt sich um eine einfache Szene mit 2 Mesh-Modellen, einer Grundfläche und einer Kugel. Außerdem wurden die Render Parameter so angepasst, dass die Anzahl der exportierten Strahlen im Rahmen bleibt, dies kann man in Abbildung 3 an den Artefakten auf dem Boden erkennen.
Im PBRT kann diese Szene auf og. Hardware innerhalb von <TODO> gerendert werden unter Ausnutzung sämtlicher Kerne beim Multi-Threading.

Tabelle 3 Übersicht Szenenparameter Testszene

kamera	Look-At-Matrix	$\begin{matrix} 400 & 20 & 30 \\ 0 & 63 & -110 \\ 0 & 0 & 1 \end{matrix}$	
	Field-Of-View	39	
Objekte		Position	Skalierung
	Killerroo Tier #1 (Mesh)	100 200 -140	
	Killerroo Tier #2 (Mesh)	200 0 0	
	Triangle Mesh	0 0 -140	
	Sphere	150 120 20	Radius 3

4.2. Feature Engineering

In diesem Abschnitt wird beschrieben wie die exportieren Raw Daten aus dem PBRT Raytracer aussehen und wie diese in Feature Vektoren konvertiert werden.

Raw Data

```
0 : {
        origin: [(float)origin.x, (float)origin.y, (float)origin.z]
        direction: [(float)direction.x, (float)direction.y, (float)direction.z]
        hit: (boolean)hit
}
```

In der oberen Beispielstichprobe wird dargestellt, welche Parameter beim Rendern der Szene exportiert wurden. Es handelt sich hier um den Strahlentest bzw. deren Ergebnis. Wie in Abschnitt 2.1. beschrieben ist, basiert das Raytracing auf der Strahlenverfolgung und ein Strahl kann durch den Ursprung (origin) und die Richtung (direction) definiert werden. Bei diesen beiden Eigenschaften handelt es sich jeweils um 3-er Vektoren. Des Weiteren befindet sich in den Raw Daten noch das Feld hit, welches angibt ob der Strahl eine der Szenen-Geometrien schneidet oder nicht.
Wenn der Wert des Feldes hit 0 ist, wird die Szenen-Geometrie nicht geschnitten,

wenn der Wert 1 ist, trifft der Strahl die Szenen Geometrie.
Wie man an der Beispielspiel Stichprobe aus den Raw Daten erkennen kann sind sämtliche Parameter, die als Features verwendet werden sollen bereits gleitkomma-Werte sodass für diese kein Mapping genutzt werden muss.
Es könnte in Betracht gezogen werden, die Werte mithilfe von <TODO DISCUSS BUCKETIZING/BINNING >

<TODO SCALING FEATURES>

<TODO DISCUSS KONVERTING, CLAMPING, FITTING>

Feature Vektor #1
[origin.x, origin.y, origin.z, direction.x, direction.y, direction.z]

Feature Vektor #2 (\w Synthetic Features)
[origin.x, origin.y, origin.z, direction.x, direction.y, direction.z, angle.x, angle.y, angle.z]

Feature Vektor #3 (\w Feature Crosses)
<TODO>

Als Label wurde jeweils der “hit”-Wert aus den Raw-Daten verwendet. Bei den Klassifizierung Modellen wird der hit-Wert verwendet um daraus kategorische Label zu erstellen.

4.3. Datenteilung

Der komplette Datensatz aller Schnittest beim Rendern der Szene ist eine CSV-Datei mit insgesamt ~16.8 Mio. Schnitttest, die Dateigröße beträgt ~1.2 Gb.
Die folgenden Datenplots zeigen die Verteilung der Features innerhalb der Datensätze.

4.3.1. Kompletter Datensatz

Items: `16.844.643`

Features

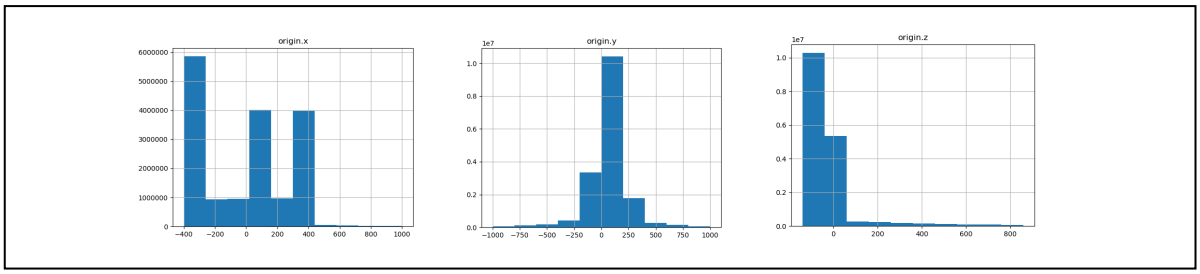

Abbildung 4 Plot Ursprungskoordinaten der Strahlen; Kompletter Datensatz

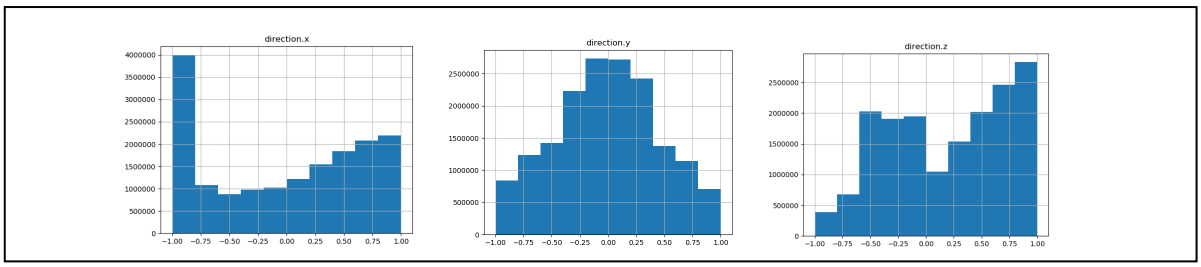

Abbildung 5 Plot Richtungsvektoren der Strahlen; Kompletter Datensatz

<TODO>

Label

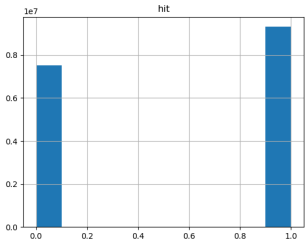

Abbildung 6 Plot Verteilung positive & negativer Schnitttests der Strahlen; Kompletter Datensatz

<TODO>

Tabelle 4 Übersicht Parameter-Verteilung im kompletten Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0
Mean	-12.8	56.6	-42.3	0.0	0.0	0.2	0.6
Std	315.9	185.5	144.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	1.2	-150.0	-0.8	-0.3	-0.3	0.0
50%	64.0	54.8	-91.6	0.1	0.0	0.3	1.0
75%	226.9	117.5	30.0	0.6	0.3	0.7	1.0
Max	1000.0	1000.0	860.0	1.0	1.0	1.0	1.0

An dieser Tabelle sowie den vorherigen Verteilungsdiagrammen kann erkannt werden, dass sämtliche Features und Labels frei von willkürlichen Werten („Magic Values“) sin, die außerhalb der erwarteten Wertebereiche liegen.

Um für die Trainings und Evaluierung die Datenmenge zu minimieren, wurde eine kleine Teilmenge der og. Daten erzeugt, welche etwas mehr als 1 Mio. Stichproben enthält.

4.3.2. Kleiner Datensatz

<TODO>

Items: 1.048.575

Features

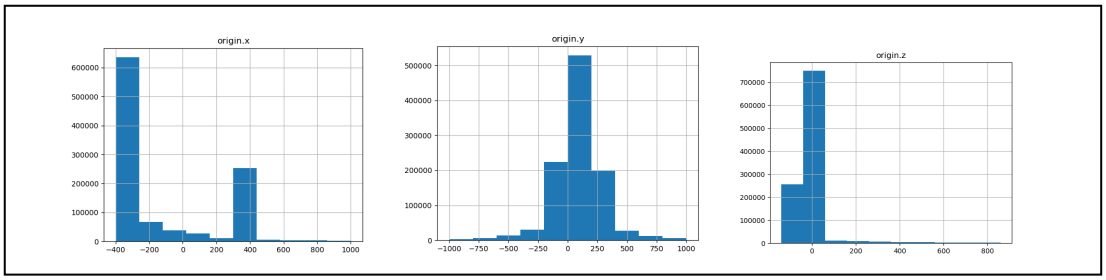

Abbildung 7 Plot Ursprungskoordinaten der Strahlen; Kleiner Datensatz

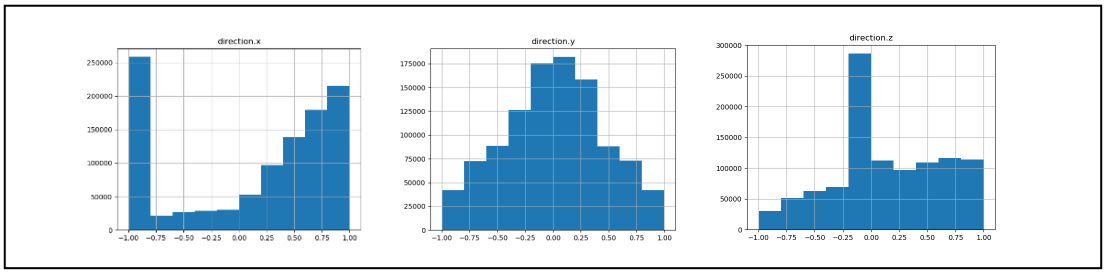

Abbildung 8 Plot Richtungsvektoren der Strahlen; Kleiner Datensatz

<TODO>

Label

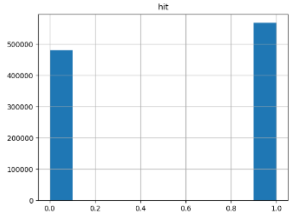

Abbildung 9 Plot Verteilung positive & negativer Schnitttests der Strahlen; Kleiner Datensatz

<TODO>

Tabelle 5 Übersicht Parameterverteilung im kleinen Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0

Mean	-141.7	76.5	-4.5	0.1	0.0	0.1	0.5
Std	348.1	207.0	105.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	-10.5	-11.0	-0.8	-0.1	-0.1	0.0
50%	-400.0	54.8	18.6	0.4	0.0	0.0	1.0
75%	396.7	186.3	30.0	0.8	0.5	0.5	1.0
Max	999.9	999.9	860.0	1.0	1.0	1.0	1.0

<TODO>

5.

AI Models

Dieser Abschnitt beschreibt die verschiedenen Modelle die Trainiert wurden in diversen Testdurchläufen. Insbesondere ist auf die verschiedenen genutzten Parameter zu achten die verwendet wurden.

In allen Trainings-/Testdurchläufen wurden die in Abschnitt 4 genannten Parameter (Ursprungspunkt und Richtung) für jeden Strahl als Features für das Modell verwendet. Der binäre Wert Hit wurde als Label verwendet für das Training und nachfolgende Evaluierung. Als Erinnerung ein mit 0 gelabelter Datensatz bedeutet, dass der Strahl die Geometrie nicht trifft, ein mit 1 gelabelter Datensatz bedeutet, dass der Strahl die Szenen-Geometrie schneidet. Ein Großteil der Modelle wurde nach dem Training Exportiert um für mögliche weitere Test zur Verfügung zu stehen, die Modell-ID hilft die Modelle den Testdurchläufen zuzuordnen und ist in den kommenden Abschnitt als oberstes bei den einzelnen Testen notiert.

Die Beschreibungen der einzelnen Durchläufe der Modelle ist hier nach Kategorie der Modelle sortiert, in der Forschungsarbeit selber wurde jedoch nach Reihenfolge der Modell-IDs vorgegangen. Insbesondere wurde zwischen lineare Classifier und Neural Networks öfters gewechselt, um diese beiden Modelltypen besser vergleichen zu können.

Es ist ebenfalls zu beachten, dass das gesamte Training in Perioden unterteilt wurde, sodass nach jeder Periode das der Fortschritt des Modelltrainings analysiert werden kann. Das aktuelle Modell wird mit dem Trainings- und Test-Set Evaluiert und die Ergebnisse werden für das Plotten der Konvergenz Kurven zwischen gespeichert. Durch diesen Monitor-Schritt verlängert sich das Modelltraining um ca. den doppelten bis dreifachen Zeitfaktor.

5.1. Linearer Regressor

Die ersten Testversuche wurden mit einem Linearen Regressor durchgeführt.
Ein linearer Regressor liefert ein floating Point-Wert zurück der jedoch keine direkte Aussage über mögliche Schnitte mit der Geometrie aussagt. Durch manuelles Testen konnten Werte außerhalb des erwarteten Werte Bereiches [0,1].

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Regressors
4.  estimator = tf.estimator.LinearRegressor(feature_columns)
```

Listing 5.1 Tensorflow Linearer Regressors

Test 1 - Linearer Regressor

Modell-ID: 1

Bei diesem 1. Test wurde ein naives herangehen gewählt. Hauptsächlich ging es darum eine generelle Pipeline zu erstellen um die in Abschnitt 4 Beschriebenen Datensätze so zu formatieren damit diese in einem Training mit Tensorflow genutzt werden können. Außerdem wurden hier die Trainings-Set und Validation-Set Größen sehr klein gewählt, damit generelle Programmier-Fehler schneller erkannt werden konnten. Es wurde der für lineare Regression typische Gradient Descent Optimizer gewählt um den Loss zu verringern. Da das Modell ungeclampte floating Point Werte zurückgeliefert, wurde für das Training-Monitoring der RMS-Error verwendet. Zu beachten ist, dass keine Feature Corssing gewählt wurde noch wurden die Input-Parameter in anderer Form verarbeitet. Die nachfolgenden Tabelle 6 zeigt die verwendeten Parameter des Modells.

Tabelle 6 Modell-Parameter Linearer Regressor Test 1

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	400000	100000

Das Training diese Modells dauerte 1 Stunde und 10 min.
Abbildung 10 zeigt die Konvergenzkurven der beiden Datensätze über die 10 Perioden des Trainings.

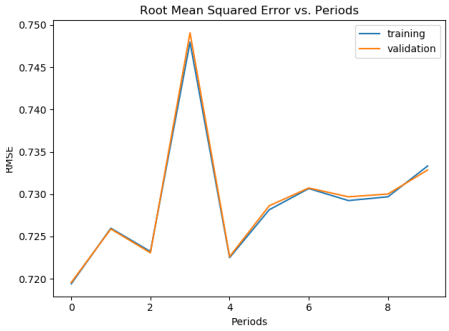

Abbildung 10 Linearer Regressor Test 1 Konvergenz-Kurve

Es kann klar erkannt werden, dass dieser naive Ansatz kein gutes Ergebnis liefert, da der Loss im Laufe der Periode immer größer wird von einem Anfänglichen LogLoss von .72 bis zu einem Finalen LogLoss von .73. Auch ist der Peak in Periode 3 zu beachten. Das Modell scheint ein recht zufälliges Ergebniss zurück zu liefern, was insbesondere an den steigenden Konvergenz Kurven zu erkennen ist und zusätzlich an den Schwankenden Abständen der Trainings- und Validation-Kurve.
Anhand dieser sehr negativen Ergebnisse wurde auf eine zusätzliche Evaluiierung anhand eines Trainings-Datensatzes verzichtet. Jedoch konnte an dieser Stelle die Infrastruktur für weitere Modelle erstellt werden, die mithilfe der Definierten Input-Funktionen und Tensorflow arbeiten können. Da es sich bei diesem Modell um ein erster Prototyp handelte, wurde dieses Modell in diesem Schritt nicht exportiert.

Test 2 – Linear Regressor

Model-ID: 2

In einem weiteren Versuch wurde auf der bereits beschriebenen Struktur aufgebaut und hauptsächlich wurden die Größe des Datensatzes verändert.
Die generellen Modellparameter wurden nicht verändert, wie in der Parameter Tabelle 7 zu sehen ist. Es wurde weiterhin der Gradient Descent Optimizer genutzt um den Loss zu minimieren.
Lediglich der Programmcode der Input-Funktion wurde etwas verändert, was sich ggf. auf die Resultate des Modells auswirkt, jedoch sind die Parameter dieselben wie in Absatz 4.2 beschrieben. Zusätzlich wurde in diesem Test eine Permutation der Testdaten erstellt, sodass eine größere Zufälligkeit vorliegt.

Tabelle 7 Modell-Parameter Linearer Regressor Test 2

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,0001	500	10	671088	167772	209715

Tabelle 7 zeigt insbesondere, dass die Größe des Trainings-Sets um den Faktor 1.6 erhöht wurde.

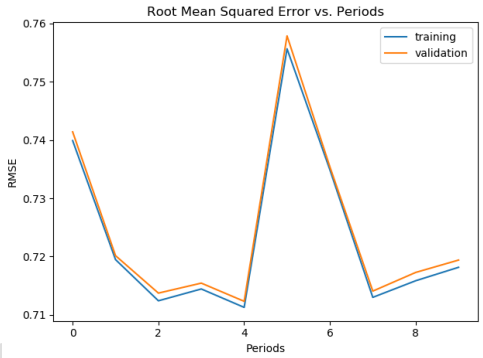

Abbildung 11 Linearer Regressor Test 2 Konvergenz-Kurve

Durch die gestiegene Größe des Trainings-Sets verlängerte sich bei diesem Testdurchlauf ebenfalls die Trainings-Dauer. Die gesamte Dauer des Trainings und des Monitorings während des Trainings betrug bei diesem Durchlauf 6 Stunden.
Das Modell wurde nach dem Training mit dem Testdatensatz evaluiert und ein finaler Root Mean Squared Error von 0.73 konnte ermittelt werden. Dieser ist geringfügig schlechter als der RMSE der in der letzten Periode des Trainings an dem Trainings-Set und Validation-Set ermittelt werden konnte.

Abbildung 11 zeigt, dass sich das Modell durch das Training in den ersten 4 Perioden verbessert. Der Peak zwischen der 4 und 7 Periode ist wieder ein Indiz, dafür, dass ein lineares Regressions Modell an dieser Stelle scheinbar willkürlich die Gewichte ändert dies einen fall um knapp 0.05 Schritte verursacht. Jedoch bleibt im Gegensatz zu den Fluktuationen im Test 1 die Kurve des Validations-Sets wie zu erwartend kontinuierlich etwas über der Trainings-Set-Kurve.

Anhand der beschriebenen Testfälle konnte erkannt werden, dass sich ein lineares Regressions Modell nicht optimal für die Lösung dieses Problems eignet,

insbesondere kann hier nicht eindeutig die Ausgabe des Modells gedeutet werden, da der Wertebereich wie in der Einleitung beschrieben an einigen Stellen stark von der Binären Einstufung ob ein Strahl die Geometrie schneidet oder nicht abweicht. Weitere Versuche, das Problem mit einem linearen Regressor (ggf. mit anderen Parametern) wurden nicht unternommen.

5.2. Linear Classifier

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Klassifizierers
4.  estimator = tf.estimator.LinearClassifier(feature_columns)
```

Listing 5.2 Tensorflow Linearer Classifier

Da es sich bei dem vorliegenden Problem generell um ein Klassifizierungsproblem handelt ist ein linearer Klassifizierer der zweite Ansatz der für dieses Problem gewählt wurde. Hierbei wird das Label als Kategorien betrachtet, es gibt demnach die Kategorie 0, die für ein nicht treffen der Szenen Geometrie steht. Die Kategorie 1 steht für ein treffen der Szenen Geometrie. Das Modell liefert eine Wahrscheinlichkeitsprognose der Zugehörigkeit der Stichprobe (des Strahls) zu jeder der beiden Kategorien, die Summe beider Prozentwerte summiert sich auf 1 auf. Wie in Absatz 2.4 beschrieben wird für lineare Classifier als Loss Function LogLoss verwendet, dies bedeutet auch, dass die Plots nicht direkt mit den Konvergenz Kurven aus den Testdurchläufen des Linearen Regressors verglichen werden können.

Test 3 – Linear Classifier

Der erste Test mit einem linearen Klassifizier Modelle wurde mit ähnlichen Parametern wie denen des linearen Regressor durchgeführt. Es wurde weiterhin der Gradient Decent Optimizer zur Minimierung des Loss beim Training gewählt. Tabelle 8 liefert eine Übersicht über die genutzten Parameter.

Tabelle 8 Modell-Parameter Linear Classifier Test 3

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Die Dauer des Trainings dieses Modell betrug 2 Stunden und 30 min.

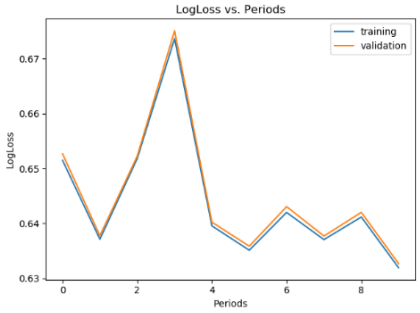

Abbildung 12 Linear Classifier Test 3 Konvergenz-Kurve

Abbildung 12 zeigt, wie das sich Modell mit seinen Prognosen über das Training entwickelt. Wieder ist zu sehen, dass das Modell etwas schlechter beim Validation-Set ist als beim Trainings-Set, was jedoch erwartet wurde. Für die Evaluierung wurde (wie bei der Verlust Funktion des Modells) der Log Loss verwendet, daher ist hier, wie in der Einleitung erwähnt, ein neuer Wertebereich auf der Y-Achse des Plots, als bei dem Linearen Regressor was ein Vergleich das bisherige Modell mit diesem erschwert. Zwar kann sich das Modell über die 10 Perioden von einem Wert von 0.65 auf knapp 0.63 verbessern, jedoch sind die Schwankungen und ebenfalls der Peak in Periode 3 zu beachten. Eine kontinuierliche Verbesserung des Modells, welche an einer gleichmäßig sinkenden konvergenz-Kurve zu erkennen wäre, ist aus diesem Testlauf nicht ersichtbar. Eine Evaluierung an dem Test-Set wurde in diesem Durchlauf nicht durchgeführt.

Test 4 – Linear Classifier

Dieser Test baut auf dem vorherigen Test 3 auf. Die Parameter wurden beibehalten, wie in Tabelle 9 Modell-Parameter Linear Classifier Test 4 erkennbar. Außerdem wurde weiterhin der Gradient Descent Optimizer verwendet.
Zusätzlich wurde nach abgeschlossenem Training die Genauigkeit (engl. accuracy) sowie die Fläche unter der Grenzwertoptimierungskurve (AUC) berechnet. Die genaue Bedeutung des Accuracy-Wertes wird in Absatz 6 diskutiert, für die folgende Testfälle kann die Accuracy als Qualitätsmaß für das Modell angesehen werden.

Tabelle 9 Modell-Parameter Linear Classifier Test 4

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Das Modelltraining dauerte an dieser Stelle 3 Stunden. Die zeitliche Schwankung bei gleichen Modell-Parametern & -Struktur lässt sich durch das Betriebssystem sowie parallel ausgeführte Anwendungen erklären.

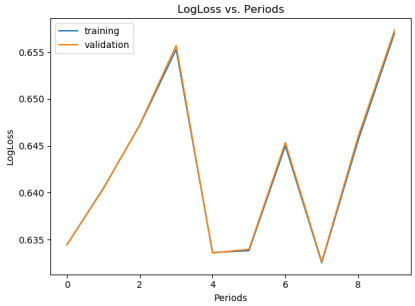

Abbildung 13 Linear Classifier Test 4 Konvergenz-Kurve

Es ist an Abbildung 13 zu erkennen, dass es keine extremen Peaks bei diesem Training gibt (die Skalierung der Log Loss Ausgabe ist feiner), jedoch ist das Modell weiterhin sehr fluktuierend und innerhalb der 10 Perioden ist keine klare Verbesserung erkennbar sein.
Die Accuracy nach dem Modell Training konnte mit 0.64 festgestellt werden. Dementsprechend liegt das Modell mit 64% der Prognosen richtig, dies ist jedoch nur marginal besser als ein Algorithmus der Zufällig die Strahlen klassifiziert und im Schnitt 50% richtige Ergebnisse liefert. Die Fläche unter der ROC Kurve konnte als der Wert 0.68 ermittelt werden.

Test 6 – Linear Classifier

In diesem Test wurden dieselben Parameter wie in Test 4 genutzt. Dieser Test galt der Stabilität des Modells, es sollte untersucht werden ob es große Schwankungen insbesondere bezüglich der neu eingeführten AUC- und Accuracy-Werte gibt. Da sich die Parameter nicht unterscheiden und wieder der Gradient Descent Optimizer genutzt wurde wird auf die genauere Analyse dieser an dieser Stelle verzichtet.

Tabelle 10 Modell-Parameter Linear Classifier Test 6

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Das Training des Modells dauerte ebenfalls 3 Stunden.

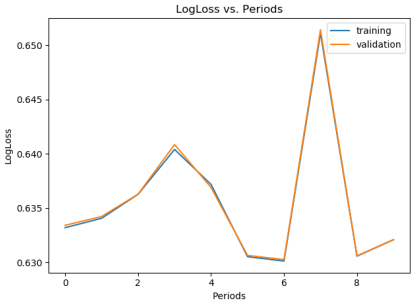

Abbildung 14 Linear Classifier Test 6 Konvergenz-Kurve

Der finale AUC-Wert betrug 0.68 und der Accuracy-Wert 0.66.
An dem Ergebnis dieses Trainings (siehe Abbildung 14) ist festzustellen, dass sich das Modell nur bedingt von dem vorherigen unterscheidet. Hier ist eine Stärke des linearen Klassifizierungsmodells fest zu stellen, bei ähnlichen Parametern, erlernt sich dieses Modell in einer ähnlichen weise bei individuellen Trainings, insbesondre im Vergleich zu Neuronalen Netzwerken ergibt sich hier ein Vorteil dieser Modelle. Eine genauere Analyse und Diskussion ist in Abschnitt 6 zu finden.

Test 8 – Linear Classifier

In diesem Durchlauf wurden in besondere ein alternativer Optimierung Algorithmus gewählt, der AdamOptimizer. Beim Adam (adaptive moment estimation) Algorithmus handelt es sich um eine Abwandlung des stochastischen Gradient Descent Algorithmus, der laut der Autoren effizient ist, wenig Speicherverbrauch hat und insbesondere für Probleme mit großen Datenmengen vorgesehen ist [4].
Insbesondere wird die Learning Rate für jedes Gewicht des Netzwerkes individuell angepasst während des Trainings.
Des Weiteren wurden die Batch-Size in diesem Durchlauf erhöht, sodass mehr Werte in den Optimierungsschritt einbezogen werden. Die Learning Rate wurde initial auf

0,001 gesetzt, was einer Erhöhung um den Faktor 10 gegenüber den bisherigen Durchläufen ist, insbesondere soll dadurch das Modell schneller konvergieren.
Auf der anderen Seite wurde jedoch die Schrittanzahl auf 800 erhöht, was generell die Trainingszeit verlängert. Zusätzlich zur Evaluierung mit einem Validation-Set wurde ein Test-Set gewählt, das zum unabhängigen Vergleich für verschiedene Modelle dienen soll.

Tabelle 11 Modell-Parameter Linear Classifier Test 8

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,001	800	40	671088	167772	209715

Durch die veränderten Parameter verlängerte sich die Trainings Zeit des Modells, das Training dauerte 3 Stunden und 30 Minuten.

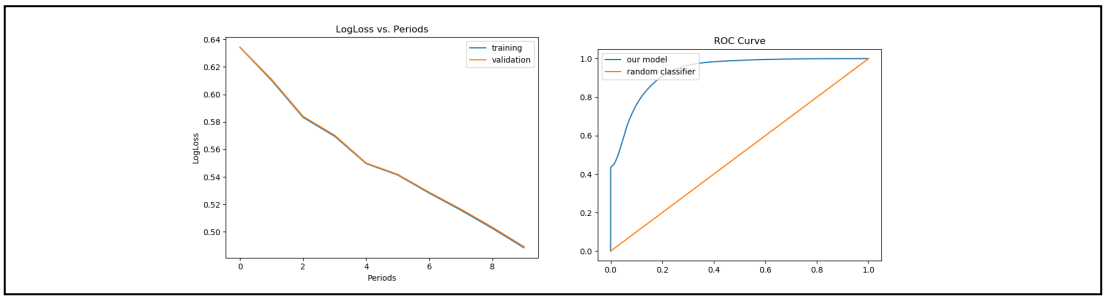

Abbildung 15 Linear Classifier Test 8 Konvergenz-Kurve (links), ROC Kurve (rechts)

Das Modell lieferte nach abgeschlossenem Training einen AUC-Wert von 0.93 und einen Accuracy-Wert von 0.80 beim Validation-Set, die ROC des Validation-Set ist in Abbildung 15 (rechts) geplottet. Beim Test-Set konnte ebenfalls ein AUC-Wert von 0.93 ermittelt werden, der Accuracy-Wert ist minimal schlechter und beträgt 0.79, was jedoch zu erwartet wurde.
Dieser Durchlauf mit dem Adam Algorithmus zur Optimierung konnte sehr positive Werte liefern, insbesondere kann an der Konvergenz Kurve auf der linken Seite in Abbildung 15 ein erwartetes Trainingsverhalten des Modells erkannt werden. Es konnte eine kontinuierliche sinkende Konvergenz Kurve ermittelt werden über die gesamten Trainingsperioden. Über 10 Perioden konnte der LogLoss von 0.64 auf 0.50 gesenkt werden. Eine Korrektheit der Prognosen von ca. 80% konnte bei dem

Validation-Set sowie dem Test-Set ermittelt werden.
Die Ursprung der starke Verbesserung der Modellergebnisse zu den vorherigen Durchläufen wurden durch die Änderung der Optimierungs-Algorithmus angenommen.

Test 9 – Linear Classifier

Aufbauend auf den sehr positiven Ergebnissen des Test 8 wurde in diesem Durchlauf weiterhin der AdamOptimizier genutzt insbesondere zur Validierung der Thesis, dass dieser die starke Verbesserung gegenüber dem generell Gradient Descent Optimizer gebracht hat. Hauptsächlich wurden in diesem Durchlauf minimal die Learning Rate und die Schrittanzahl des Optimier Algorithmus angepasst, alle weiteren Parameter wurden wie in Tabelle 12 ersichtlich gegenüber dem vorherigen Durchlauf beibehalten.

Tabelle 12 Modell-Parameter Linear Classifier Test 9

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Es konnte wieder eine Verlängerung der Trainingsdauer gemessen werden, die Trainingsdauer betrug in diesem Durchlauf 4 Stunden, dies ist hauptsächlich durch die Erhöhung der Schrittzahl um 100 zurückzuführen.

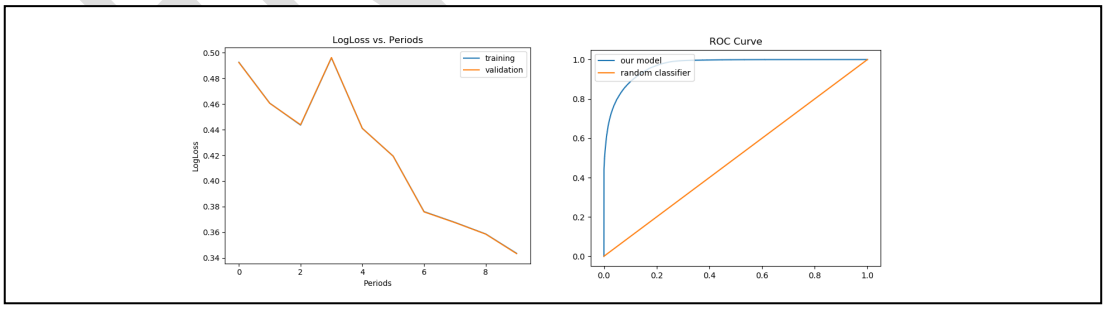

Abbildung 16 Linear Classifier Test 9 Konvergenz-Kurve (links), ROC Kurve (rechts)

Beim Validation-Set konnte nach abgeschlossenem Training des Modells ein AUC-Wert von 0.97 und eine Accuracy von 0.87 ermittelt werden, dieselben Werten konnte bei dem Test-Set ermittelt werden.

Im Vergleich zum vorhergehenden Test konnten die Genauigkeit um weitere 7% zum vorhergehenden Modell verbessert werden. In Abbildung 16 kann erkannt werden, dass die ROC Kurve stärker steigt und sich der AUC-Wert weiter dem Wert 1.0 annähert. Die Konvergenzkurve sinkt stetig über das Training abgesehen von dem Peak in Periode 3, es ist jedoch fest zu stellen, dass sich die Kurve des Trainings-Set kaum von der des Validation-Sets unterscheidet, dies wurde dadurch erklärt, dass eine generell starke Ähnlichkeit zwischen allen Strahlen besteht von denen 64% Prozent für das Training verwendet wurden (siehe Tabelle 12).
Es ist zu bemerken, dass das Training dieses Modells bereits bei einem LogLoss-Wert von knapp 0.50 startet, dieses ist der Wert, den das Model des vorhergehenden Tests 8 in der letzten Periode erreicht. Hier wird angenommen das das Modell 9 auf den des Modells 8 aufbaut und dessen Werte initial übernimmt, da sich die beiden Modelle in demselben temporären Speicherbereich befanden. Dafür spricht auch, dass Modell 8 bis zur letzten Periode eine stetig sinkende Konvergenzkurve besitzt.
Außerdem könnte diese Hypothese den Peak in Periode 3 des Modells 9 erklären, der dadurch entsteht, dass die Historie des Modells 8 nicht übernommen wurde.
Mit knapp 88% Trefferqoute konnte dieses Modell als bestes Modell dieser Testreihe bewertet werden.

5.3. Neuronal Network

```
1.  import tensorflow as tf
2.
3.  # erstellen eines DNNClasifier regressor Objektes
4.  # hidden_units gibt die Anzahl der Layer und die jeweils
5.  # enthaltenen Neuronen an
6.  estimator = tf.estimator.
    DNNClassifier(feature_columns,hidden_units=[512,512,24])
```

Listing 5.3 Tensorflow Deep Neural Network Klassifizierer

Test 5 – Deep Neural Network Classifier

Gradient Descent Optimizer

Tabelle 13 Modell-Parameter Deep Neural Network Classifier Test 5

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size

10	0,001	1000	40	671088	167772	209715

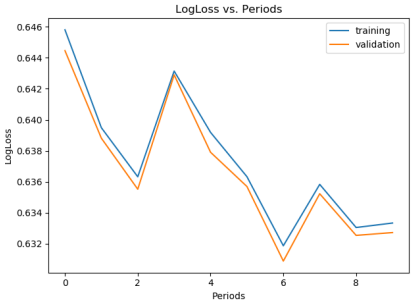

Abbildung 17 Deep Neural Network Classifier Test 5 Konvergenz-Kurve

AUC on the validation set: 0.75

Accuracy on the validation set: 0.59

```
AUC on the test set: 0.75
Accuracy on the test set: 0.59

hidden_units=[1024, 512, 256]
```

Average time in model.py: 2722.4229044437407s

Test 7 – Deep Neural Network Classifier

Gradient Descent Optimizer

Tabelle 14 Modell-Parameter Deep Neural Network Classifier Test 7

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,001	1000	40	671088	167772	209715

```
FtrlOptimizer
```

L1 – REGULARIZATION STRENGTH = 0.1

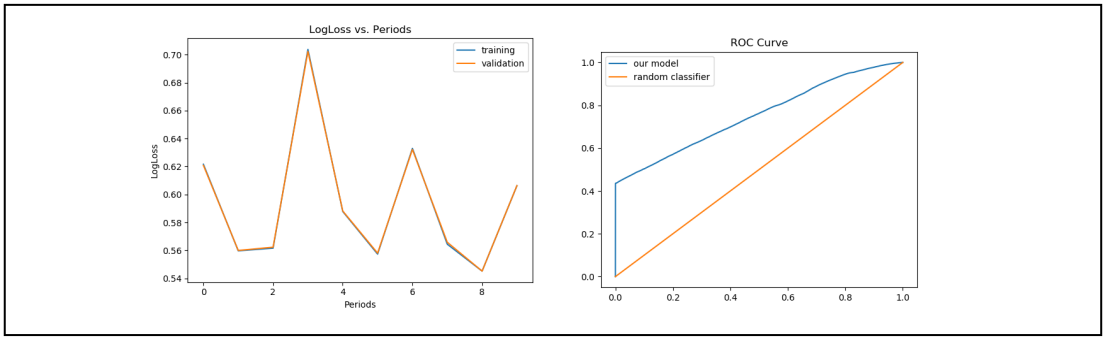

Abbildung 18 Deep Neural Network Classifier Test 7 Konvergenz-Kurve (links), ROC Kurve (rechts)

Average time in model.py: 6788.012552928924s

```
AUC on the validation set: 0.50
Accuracy on the validation set: 0.54
```

```
hidden_units=[1024, 512, 512, 256]
```

Test 10 – Deep Neural Network Classifier

Tabelle 15 Modell-Parameter Deep Neural Network Classifier Test 10

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

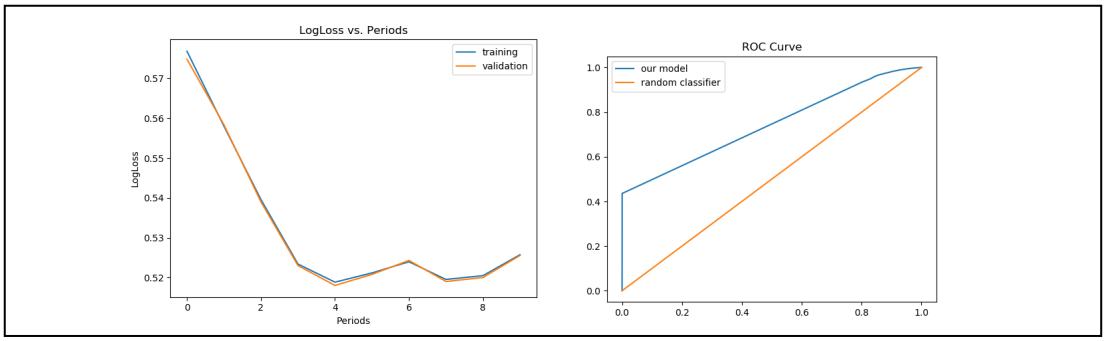

Abbildung 19 Deep Neural Network Classifier Test 10 Konvergenz-Kurve (links), ROC Kurve (rechts)

```
AdamOptimizer
```

```
hidden_units=[30, 30, 20, 20, 10]

AUC on the validation set: 0.74
Accuracy on the validation set: 0.70
AUC on the test set: 0.74
Accuracy on the test set: 0.69
```

Test 11 – Deep Neural Network Classifier

```
AdamOptimizer
```

Tabelle 16 Modell-Parameter Deep Neural Network Classifier Test 11

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

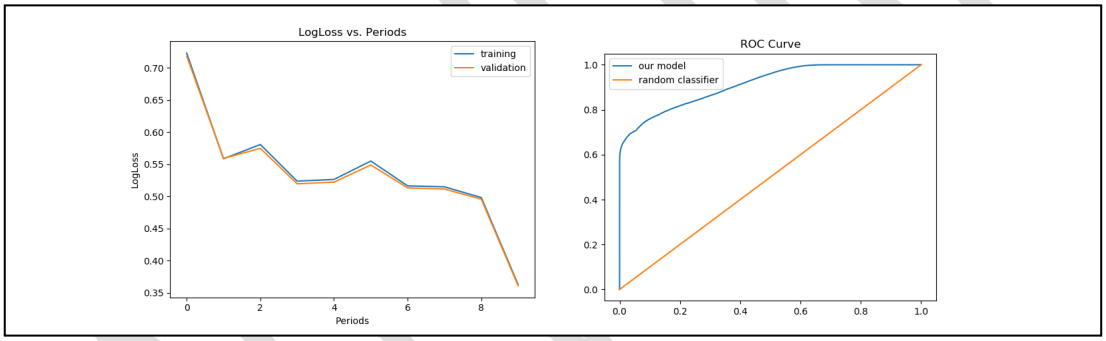

Abbildung 20 Deep Neural Network Classifier Test 11 Konvergenz-Kurve (links), ROC Kurve (rechts)

```
hidden_units=[ [300, 300, 200, 200, 100]
AUC on the validation set: 0.91
Accuracy on the validation set: 0.82
AUC on the test set: 0.94
Accuracy on the test set: 0.85
```

Test 12 – Deep Neural Network Classifier

Tabelle 17 Modell-Parameter Deep Neural Network Classifier Test 12

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

```
hidden_units=[ [600, 600, 400, 400, 200]
```

6.

Accuracy & Speed

Dieser Abschnitt beschreibt und vergleicht die Genauigkeit und Geschwindigkeiten der Modelle.
<TODO>

Tabelle 18 Raytracing Modell Confusion Matrix

True Positive (TP): Ein Strahl, der die Szenen Geometrie trifft, wird als ein solcher prognostiziert.	**False Positive (FP):** Ein Strahl, der die Szenen Geometrie nicht trifft, wird als fälschlicherweise als ein Strahl prognostiziert, der die Szene schneidet.
False Negative (FN): Ein Strahl, der die Szenen Geometrie trifft, wird fälschlicherweise als ein Strahl prognostiziert, der die Szene nicht schneidet.	**True Negative (TN):** Ein Strahl, der die Szenen Geometrie nicht trifft, wird als ein solcher prognostiziert.

7.

Practical Applications

Dieser Abschnitt diskutiert die praktische Anwendbarkeit der aus dieser Arbeit resultierenden Erkenntnisse.
<TODO>

8.

Further Work

<TODO>

Literaturverzeichnis

[1] M. Pharr, W. Jakob und G. Humphreys, Physically Based Rendering: From Theory To Implementation, Morgan Kaufmann, 2016.

[2] A. S. Kaplanyan, C. R. A. Chaitanya, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai und T. Aila, „Interactive Reconstruction of Monte Carlo Image Sequences using aRecurrent Denoising Autoencoder,“ [Online]. Available: https://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf. [Zugriff am April 2019].

[3] Google Developers, „Machine Learning Crash Course,“ Google, 2019. [Online]. Available: https://developers.google.com/machine-learning/crash-course/. [Zugriff am July 2019].

[4] D. P. Kingma and J. Ba, "Adam: A Method for stochastik Optimization," ICLR, 2015.

Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum	Unterschrift (Vor- und Nachname)