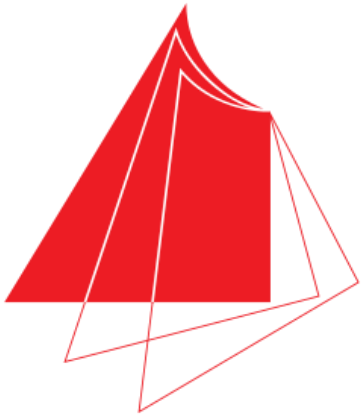

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät für Maschinenbau und Mechatronik

der Hochschule Karlsruhe Technik und Wirtschaft

Genetische Algorithmen zur Optimierung von Hyperparametern eines künstlichen neuronalen Netzes

Bachelorarbeit (B.Eng.)

vom 21.10.2019 bis zum 21.01.2020

vorgelegt von

Christian Heinzmann

geboren am 18.02.1995 in Heilbronn

Martrikelnummer: 52550

Professor: Prof. Dr.-Ing. habil. Catherina Burghart

Co-Professor: Prof. Dr.-Ing. Ferdinand Olawsky

Externer Betreuer: M. Sc. Lukas Kohout

Eigenständigkeitserklärung und Hinweis auf verwendete Hilfsmittel

Hiermit versichere ich wahrheitsgemäß, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen des Berichts, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Datum: _____ Unterschrift: _____

Kurzfassung

Im Kontext der Klassifizierung von Aktion und Objekten, soll die Optimierung von Hyperparametern bei künstlichen neuronalen Netzen mit Hilfe von Genetischen Algorithmen im Rahmen dieser Arbeit durchgeführt werden. Da es für die Hyperparameter keine klaren Regeln gibt und sie je nach Daten und Modell-Architektur anders gewählt werden müssen. Somit ist es für den Entwickler sehr zeitaufwendig, diese Hyperparameter von Hand auszuwählen. Daher ist das Ziel der Arbeit, ein Framework zu entwickeln und zu implementieren, in dem künstliche neuronale Netze automatisiert trainiert, getestet, Hyperparameter angepasst und ausgewertet werden. Dazu wurde ein Konzept entwickelt, in welchem die Hyperparameter als Gene des Genetischen Algorithmus umgesetzt werden. So werden die Hyperparameter nach dem Vorbild der Evolution über Generationen intelligent angepasst. Des Weiteren wurde der State-of-the-Art Algorithmus ausgesucht und implementiert, um den Genetischen Algorithmus vergleichen zu können. Es handelt sich um die Zufallssuche, welche zufällige Individuen im Suchraum testet. Beide Optimierungsalgorithmen wurden zur Optimierung von Hyperparametern in Fully Connected Networks und Convolutional Neural Networks verwendet. Zudem wurden verschiedene Datensätze verwendet, um eine unabhängige Aussage zu erreichen. Der Genetische Algorithmus konnte leider keine Verbesserungen im Vergleich zur Zufallssuche erreichen. Beide Algorithmen finden Hyperparameter, welche gleich gute Klassifizierungsergebnisse liefern. Sie unterscheiden sich nur im Nachkommabereich und sind somit vernachlässigbar klein. Dennoch hat der Genetische Algorithmus einen Vorteil: Über seine letzte Population kann eine zusätzliche Aussage getroffen werden, in welchem Bereich sich die Hyperparameter befinden müssen, um gute Ergebnisse zu erreichen. Dies ist mit der Zufallssuche nicht möglich, da diese nicht genügend gute Individuen findet, um diese Aussage zu treffen. Die wenigen die die Zufallssuche findet, sind ausreichend gut und können von dem Genetischen Algorithmus nicht übertroffen werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren zeigte sich ein enormer Zuwachs an Publikationen und Forschungen zum Thema "Maschinelles Lernen". Die künstlichen neuronalen Netze dominieren dieses Feld, aber ihr Training und ihr Erfolg hängt noch immer von empirisch ausgesuchten Anfangsparametern, den sogenannten Hyperparametern, ab. Zu diesen Hyperparametern können Modell-Architektur, Verlustfunktion, Optimierung Algorithmen, Batchsize, Dropout und Lernrate gezählt werden. Momentan werden diese Hyperparameter meist nach Ermessen des Entwicklers ausgewählt, denn es gibt keine formalen Regeln. Dieses Auswählen und Testen der Hyperparameter beansprucht sehr viel Zeit und Mühe, da die anfängliche Auswahl der Parameter meist suboptimal ist und diese noch einige Male angepasst und getestet werden müssen. Dieses Optimieren der Hyperparameter sollte jedoch nicht zu den Aufgabe des Entwicklers gehören und sollte von Maschinen übernommen werden, welche in Bereich der Optimierung wesentlich effizienter arbeiten. [Fra17, p. 337]

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es die Optimierung von Hyperparametern für künstliche neuronale Netze zu automatisieren. Diese Optimierung soll Mithilfe des Genetischen Algorithmus erfolgen. Dafür muss als erstes ein automatisierter Trainings- und Auswerte-Prozess der künstlichen neuronalen Netze erstellt werden. Anschließend soll ein Konzept zur Optimierung von Hyperparametern mit Hilfe des Genetischen Algorithmus entwickelt und

implementiert werde, wozu der automatische Trainings- und Auswerte-Prozess verwendet werden soll. Die Ergebnisse des Genetischen Algorithmus sollen anschließend mit den Ergebnissen des State-of-the-Art Algorithmus zur Optimierung von Hyperparametern gegenübergestellt und ausgewertet werden. Darüber hinaus soll eine Optimierung der Hyperparameter unter Betrachtung geringer Datenmengen durchgeführt werden.

1.3 Aufbau der Arbeit

Im zweiten Kapitel wird auf die Grundlagen der künstlichen neuronalen Netze eingegangen. Anschließend werden die zu optimierenden Hyperparameter definiert. Nachfolgend werden die Optimierungsalgortihmen erläutert, dazu gehören der Genetische Algorithmus mit seinen 5 Schritten und die Rastersuche bzw. die Zufallssuche.

Im dritten Kapitel wird auf den Stand der Technik eingegangen. Dazu werden die neusten Forschungsergebnisse erläutert. Zu diesen gehört das Populations Basierte Training(PBT) von Google und die NeuroEvolution of Augmenting Topologies(NEAT). Zudem werden Anwendungsbeispiele des Genetischen Algorithmus, welche in der Forschung als auch in der Industrie verwendetet werden aufgezeigt.

Im vierten Kapitel wird auf das Konzept zur Optimierung der Hyperparameter eingegangen. Hierfür wird ein Konzept zum Genetischen Algorithmus zur Optimierung der Hyperparameter eines künstlichen neuronalen Netzes vorgestellt. Anschließend wird auf das Konzept zur Evaluierung und Auswertung mit Hilfe des Dichte-Diagramms eingegangen.

Im fünften Kapitel wird der Systemaufbau erklärt. Es wird auf die verwendete Hardware eingegangen. Anschließend wird die exakte Implementierung des Genetischen Algorithmus, der Zufallssuche und der Evaluation sowie die Auswertung beschrieben.

Im sechsten Kapitel werden die Ergebnisse der Optimierung dargestellt und besprochen. Zudem wird die visuelle Auswertung zu jedem Hyperparameter aufgezeigt und erläutert.

Im siebten und letzten Kapitel werden alle vorherigen Kapitel als Fazit zusammengefasst. Der Ausblick zum Thema bildet den Abschluss der Arbeit.

Kapitel 2

Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen, welche zum Verständnis der vorliegende Arbeit wichtig sind, beschrieben. Zu Beginn erfolgt ein Einstieg in künstliche neuronale Netze, anschließend werden die Hyperparameter und ihre Besonderheiten thematisiert. Dann folgt eine kurze Einführung in die Optimierungsverfahren. Abschließend werden die grundlegende Aspekte von Zufallssuche und Genetischen Algorithmen erläutert.

2.1 Künstliche Neuronale Netze

Im folgenden Unterkapitel wird zuerst der Aufbau eines Neurons erklärt. Anschließend wird der strukturelle Aufbau eines künstlichen neuronalen Netzes(KNN) näher gebracht. Es werden wichtige Funktionen wie die Verlustfunktion und die Fehlerrückführung erklärt. Zum Schluss wird auf die Hyperparameter, welche für die Arbeit essenziell sind, eingegangen.

Künstliche Intelligenz (engl. Artificial Intelligence) sorgte in den letzten Jahren für weitreichende Schlagzeilen. Im Detail versteckt sich hinter dieser künstlichen Intelligenz meist Maschinelles Lernen im speziellen tiefe künstliche neuronale Netze (engl. Deep Artificial neuronal Networks oder Deep Learning). Diese werden für Anwendungen mit großen Datenmengen eingesetzt, bspw. in Bereichen der Bildverarbeitung, Chatbots und Fahrassistenz Systemen. Beim Deep Learning werden die Funktionen durch ein Modell erlernt, welches künstliches neuronales Netz genannt wird. Dieser Begriff stammt ursprünglich aus der Neurobiologie. Bei den künstlichen neuronalen Netzen handelt es sich aber nicht um Nachbildungen des Gehirns, das natürliche Vorbild dient nur als Inspirati-

on. In Abbildung 2.2 ist so ein künstliches neuronales Netz zu sehen, wobei jede Schicht aus einzelnen Neuronen aufgebaut ist. Diese Schichten können dann Schichtweise zusammengefügt werden, wodurch ein Netz entsteht. Diese Verbindungen repräsentieren die Gewichte, über welche einem Netz, lineare als auch nicht lineare Zusammenhänge antrainiert werden. [SSF+16, p. 11]

2.1.1 Aufbau eines Neurons

Abbildung 2.1 zeigt ein Neuron. Dieses hat folgenden schematischen Aufbau Aufbau: Eingänge, Gewichte, Schwellwert, Aktivierungsfunktion und einen Ausgang. In den nachfolgenden Unterkapiteln werden diese ausführlich erklärt, Informationen dazu stammen wenn nicht anderweitig angegeben aus der Arbeit [SSF+16, p. 12].

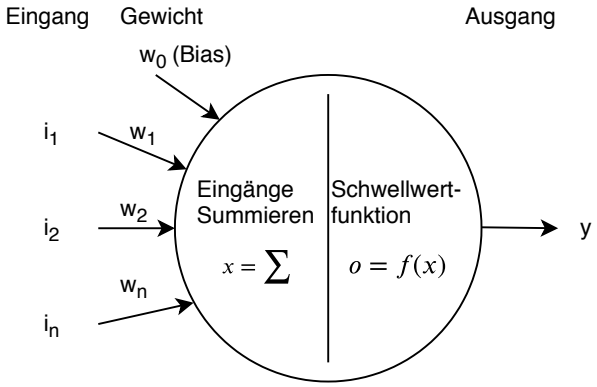

Abbildung 2.1: Aufbau eines Neurons

Eingang Bei den Eingangswerten $i_1, ..., i_n$ handelt es sich um reelle Zahlen, diese werden mit den Gewichten $w_1, ..., w_n$ verrechnet. Ein Neuron hat meist mehrere Eingangsgrößen, welche alle zusammen mit den Gewichten und dem Schwellwert aufsummiert werden. Die Gewichte werden zufällig initialisiert und per Training verbessert, somit handelt es sich um angelernte Werte, welche durch die Fehlerrückführung (engl. Backpropagation) verbessert werden. Auf diese aufsummierten Ergebnisse wird anschließend ein Schwellwert (engl. Bias) w_0 gerechnet, dieser führt zu einem besseren Verhalten beim Trainieren. Beim Schwellwert handelt es sich auch um einen angelernten Wert. Dieser hilft die Flexibilität des Netzes zu erhöhen. Der Schwellwert sorgt dafür, dass ein Ausganswert erzeugt wird, auch wenn kein Eingangswert anliegt. Daraus ergibt sich folgende Übertragungsfunktion:

$$x = \sum_{k=1}^{n} i_k * w_k + w_0 \tag{2.1}$$

Aktivierungsfunktion Die Aktivierungsfunktion kann man sich als Schwellwertfunktion vorstellen. Sie stellt den Zusammenhang zwischen Eingang und Ausgang her. Die entstandene Summe x wird Aktivierung genannt und anschließend über eine Aktivierungsfunktion transformiert:

$$o = f(x) \tag{2.2}$$

Die Aktivierungsfunktion kann dabei verschiedene Formen haben. Für einfache Aufgaben kann Beispielsweise eine Sprungfunktion verwendet werden:

$$\sigma(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{2.3}$$

Für das approximieren von wertkontinuierlichen Funktionen wird die Sigmoid Funktion verwendet.

$$sig(t) = \frac{1}{1 + e^{-t}} \tag{2.4}$$

Bei der Klassifikation werden, ReLU-Layer (2.5) oder Leaky-ReLU Layer (2.6) benutzt, diese verhindern das Explodieren bzw. Verschwinden des Grandienten beim Training:

$$R(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{2.5}$$

$$R(z) = \begin{cases} 1, & z \geq 0 \\ \alpha z, & z < 0 \end{cases} \tag{2.6}$$

Ausgang Die Schwellwertfunktion übergibt einen Wert an den Ausgang. Dieser Ausgangswert kann entweder an andere Neuronen weitergegeben oder als finales Ergebnis verwendet werden.

2.1.2 Struktureller Aufbau eines künstlichen neuronalen Netzes

Durch zusammenschließen solcher Neuronen entsteht ein künstliches neuronales Netz, welches auch Multi-Layer-Perzeptron(MLP) genannt wird. Dabei sind grundsätzlich jeg-

liche strukturelle Anordnungen der Neuronen möglich. Besonders verbreitet ist das sogenannte Feedforward Netz(FFW). Bei den FFW Netzen sind die Verbindungen nur in eine Richtung gerichtet, wodurch die Informationen von Vorne(Eingang) nach Hinten(Ausgang) weiter gegeben werden. Dies wird beispielhaft in Abbildung 2.2 gezeigt. Hier ist gut zu sehen, dass ein künstliches neuronales Netz in drei Schichten aufgebaut werden kann. [SSF+16, p. 21]

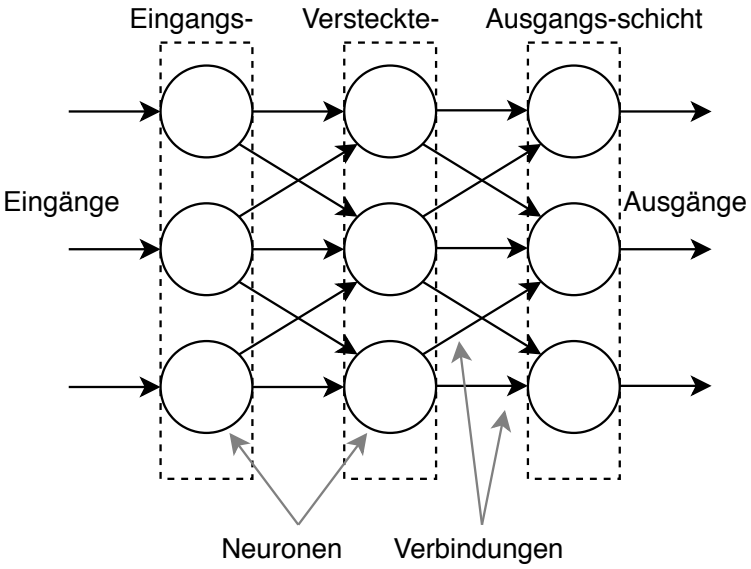

Abbildung 2.2: Künstliches neuronales Netz mit drei Schichten je drei Neuronen

- **Eingangsschicht** Die Neuronen der Eingangsschicht sind nicht wie die in 2.1.1 beschriebenen Neuronen aufgebaut. Die einzige Aufgabe der Eingangsneuronen ist das Verteilen der Eingangsinformationen an die versteckte Schicht, weshalb es immer genau so viele Eingangsneuronen wie Eingangssignale geben muss.

- **Versteckte Schicht** Die versteckte Schicht (engl. hidden Layer) besteht aus den in 2.1.1 erläuterten Neuronen. Sie kann aus einer beliebigen Anzahl von Neuronen aufgebaut sein. Es kann auch beliebig viele dieser Schichten geben. In der Literatur werden neuronale Netze mit mehr als einer versteckten Schicht als Deep Neural Networks bezeichnet.

- **Ausgangsschicht** Die Ausgangsschicht beinhaltet genauso viele Neuronen wie Ausgangsgrößen gewünscht. Aus dieser können dann die Klassifizierungswerte entnommen werden.

2.1.3 Verlustfunktion

Die Verlustfunktion (engl. Lossfunction) stellt ein ausgesuchtes Maß der Diskrepanz zwischen den beobachteten und den vorhergesagten Daten dar. Sie bestimmt somit die Leistungsfähigkeit des künstlichen neuronalen Netzes während des Trainings. Beim Rückgabewert wird von dem Fehler (engl. loss) gesprochen.

2.1.4 Fehlerrückführung

Fehlerrückführung ist ein Optimierungsalgorithmus, welcher auf dem Gradientenabstieg basiert. Dies ist nur möglich, da ein KNN aus verketteten differenzierbaren Gewichten aufgebaut ist und somit die Kettenregel angewendet werden kann. Ziel ist es den Fehler der Verlustfunktion zu minimieren. Hierfür wird ein Muster am Eingang des Netzes angelegt. Der Ausgangswert wird mit dem Soll-Ergebnis verglichen. Der aufsummierte Fehler wird Schichtweise von der Ausgangsschicht zur Eingangsschicht zurück propagiert. Währenddessen werden die Gewichte so geändert, dass der Fehler minimiert wird. Es gibt verschiedene Ansätze der Fehlerrückführung, welche auch Optimierer genannt werden. Diese bestimmen die Regeln, wie die Verlustfunktion zur Aktualisierung der Parameter verwendet wird. [Cho18, p. 83]

2.1.5 Hyperparameter

Als Hyperparameter werden in Bezug auf künstliche neuronale Netze meist die Anfangsparameter bezeichnet, welche zum Traininieren eines Netzes essenziell sind. Für diese gelten keine universellen Werte, sondern müssen je nach Daten und Modell speziell angepasst und verändert werden. Infolgedessen sind nur einige Regeln und grobe Abschätzungen bekannt, in welchen Grenzen sich die Hyperparameter befinden. Zu den Hyperparametern, welche in dieser Arbeit verwendet werden, gehören folgende:

- **Die Lernrate** ist das Maß für die Schrittgröße beim Gradientenabstieg. Eine zu kleine Lernrate kann zu einem sehr langsamen Training führen. Eine zu große Lernrate kann dazu führen, dass der Wert um ein Minimum oszilliert. Dies ist grafisch in Abbildung 2.3 dargestellt. [Fra17, p. 49]

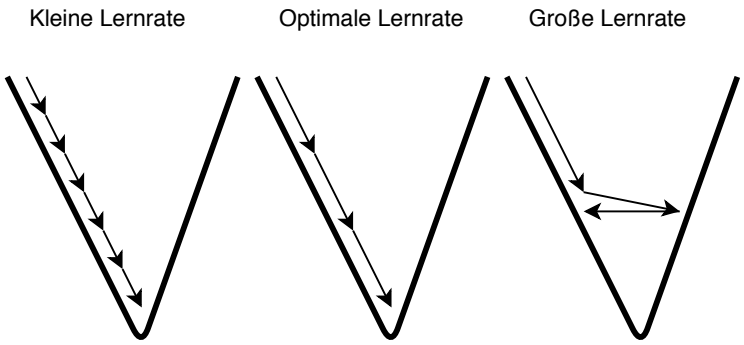

Abbildung 2.3: Links zu kleine Lernrate, mitte optimale Lernrate, rechts zu große Lernrate

- **Der Dropout** gibt an, wie viel Prozent einer Schicht während des Trainings deaktiviert werden. Deaktiviert bedeutet, diese werden ignoriert und für diese Iteration ausgesetzt. Symbolisch zu sehen in Abbildung 2.4. Durch das deaktivieren werden zufällige Neuronen „Null“ gesetzt, dadurch muss das Netz diese deaktivierten Neuronen durch andere Neuronen ersetzen, wodurch die Informationen, welche in den Neuronen gespeichert sind, besser verteilt werden. Dropout wird nur an den versteckten Schichten durchgeführt und kann zu einem verbesserten und robusteren Ergebnis führen[Fra17, p. 109].

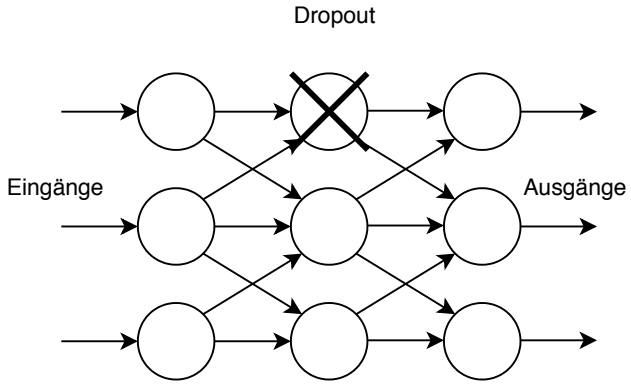

Abbildung 2.4: Abdeckungsgrad entspricht einem Drittel

- **Die Epochen** geben an, wie viele Iterationen aller Trainingsdaten durchgeführt werden sollen. Ein zu langes Training kann zu einem Auswendiglernen der Daten führen und dementsprechend wird nicht mehr richtig generalisiert. Bei einer zu geringen Epochen-Anzahl kann das Netz nicht genügend Informationen lernen, das Ergebnis ist dann entsprechend schlecht [Fra17, p. 109].

- **Die Batchsize** gibt an, in welchen Paketen die Daten dem künstlichen neuronalen Netz zugeführt werden. Künstliche neuronale Netze verarbeiten die Daten nicht auf einmal, sondern teilen sie in sogenannte Stapel ein[Fra17, p. 109].

- **Der Optimierer** (engl. Optimizer) gibt vor, wie sich der Fehler auf die Aktualisierung der Parameter auswirkt. So gibt es beispielsweise einen stochastischer Gradientenabstieg mit Momentum Optimierer (engl. Stochastic Gradient Descent with momentum), der die Trägheit des Gradientenabstiegsverfahren berücksichtigt und damit besser lokale Minima über geht[Fra17, p. 109]. Die Erklärung aller Optimierer ist nicht Teil der Arbeit, es wird daher auf die Arbeit [Rud16] verwiesen, in welcher die wichtigsten Optimierer ausführlich erklärt werden.

- **Die Modell-Achitektur** definiert wie ein Netz aufgebaut ist. Hier werden die Anzahl der Schichten und Neuronen festgelegt. Ein zu kleines Netz kann nicht alle Informationen lernen, die in den Daten vorhanden sind. Dies zeigt sich schnell in schlechten Ergebnissen. Ein zu großes Netz mit zu vielen Schichten und Neuronen kann dazu führen, dass die Genauigkeit bei den Trainingsdaten steigt, was wiederum dazu führt, dass sich die Genauigkeit bei den Testdaten verschlechtert. Zudem ist die Dauer des Trainings wesentlich länger[Fra17, p. 70].

2.2 Optimierung

Als Optimierung wird im Allgemeinen die Maximierung oder Minimierung einer Zielfunktion bezeichnet, wobei eine oder mehrere Restriktionen eingehalten werden sollen. Hierbei existiert ein Lösungsraum, in dem sich sämtliche mögliche Lösungen befinden. Das Optimierungsverfahren soll den Lösungsraum definieren und diesen Lösungsraum nach der Lösung mit dem optimalen Zielfunktionswert absuchen [GLL11, p. 15]. Bei der Hyperparameteroptimierung handelt es sich um ein Mehrzieloptimierungproblem, wodurch sich ein mehrdimensionaler Suchraum ergibt. Eine exakte Berechnung des Optimums ist, wegen des großen Suchraums, meist nicht möglich. Aus diesem Grund ist für diese Aufgabe nur eine heuristische Optimierung möglich. Bei heuristischen Verfahren wird der Lösungsraum nach einem festgelegten Muster durchsucht und bewertet. Die gefundenen Lösungen sind meist nicht die besten Lösungen, sind aber der Kompromiss aus vertretbarem Rechenaufwand und hinreichend gut eingeschätzter Lösung. Trotz allem kann nicht garantiert werden, dass es sich bei der gefunden Lösung um die optimale Lösung handelt[BNR06]. Für diese heuristischen Verfahren zur Hyperpara-

meter Optimierung werden in dieser Arbeit Zufallssuche und Genetischen Algortihmen gegenübergestellt und anhand ihrer Lösungen verglichen und bewertet.

Vereinfachtes Beispiel für Mehrzieloptimerung Angenommen, es soll ein künstliches neuronales Netz mit k Schichten und j Neuronen zur Klassifizierung von einfachen handgeschriebenen Zahlen erstellt werden. Der Entwickler entscheidet sich für ein Netz mit 3 Schichten und jeweils 3 Neuronen. Nach dem Training hat das Netz eine Genauigkeit von 85 Prozent. Nun kann der Programmier/Entwickler nicht sicher sagen, ob für k = 3 und j = 3 die optimale Lösung gefunden wurde. Um dies beurteilen zu können, müssen viele Experimente durchgeführt werden. Die Frage dabei ist, wie die besten Werte für k und j zu finden sind, um die Klassifizierungsgenauigkeit maximieren zu können. Dieser Vorgang, speziell im Zusammenhang mit künstlichen neuronalen Netzen, wird als Hyperparameter-Optimierung bezeichnet. Bei der Optimierung wird mit einem Initialwert gestartet. Dieser ist in den seltensten Fällen die exakte Lösung und muss einige Male verändert werden, um auf ein Optimum zu gelangen. In dieser Arbeit ist dafür die Zufallssuche, bzw. der Genetische Algorithmus zuständig[Gad18, p. 130].

2.3 Rastersuche und Zufallssuche

Die Rastersuche(engl. Grid Search) ist einer der am weiterverbreiteten Optimierungsalgorithmen. Der Suchraum wird durch ein Raster aufgeteilt, je höher die Abtastrate desto feiner ist das Raster. Die Überschneidungen des Gitters sind im Endeffekt die einzelnen Punkte, welche getestet werden. Während des Suchvorgangs wird das Raster nicht verändert. Somit kann es passieren, dass die optimalsten Parameter nicht gefunden werden, zudem ist die Suche sehr langsam bzw. rechenaufwendig. Um eine schnellere Berechnung von Hyperparametern zu garantieren, wurde von James Bergtra und Yoshua Bengio die Zufallssuche untersucht. Sie konnten statistisch feststellen, dass die Zufallssuche schneller bessere Ergebnisse liefert. [BB12] Wie in Abbildung 2.5 zu sehen ist, ist die Abdeckung des Suchraums durch die Zufallssuche, auf der rechte Seite, wesentlich besser als die Rastersuche auf der linken Seite. Dies ist möglich, da die Zufallssuche zufällig innerhalb des Rasters sucht und nicht nur die Überschneidungen des vorgegebenen Rasters untersucht.

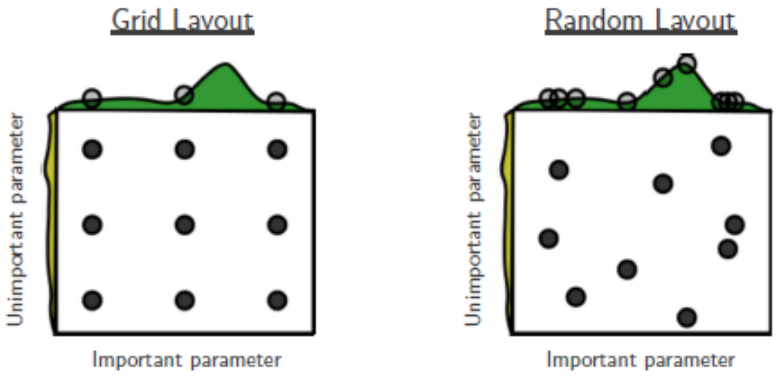

Abbildung 2.5: Linke Seite: "Grid Search", Rechte Seite: "Random Search"[BB12]

2.4 Genetische Algorithmen

Im folgenden Unterkapitel wird näher auf die 5 Schritte des Genetischen Algorithmus(GA) 1 eingegangen. Zuerst wird auf den Aufbau und die Initialisierung einer Population eingegangen. Anschließend folgt eine Zusammenfassung der Fitnessfunktion. Darauf folgend werden verschiedene Möglichkeiten für die Eltern Selektion dargelegt. Anschließend wird die Vermehrung durch Crossover und Mutation erklärt. Zum Schluss wird auf den Austausch der Populationen eingegangen.

Genetische Algorithmen sind der Evolution von Darwin nachempfunden. Es geht hierbei um das ÜberlebeFon des Fittesten (engl. Survial of the Fittest) wobei es in dieser Arbeit nicht um das biologische Leben geht. Der Algorithmus ist der in der Natur vorkommenden Evolution ähnlich, weshalb auch viele Begriffe an die biologische Evolution angelehnt werden. Genetische Algorithmen sind heuristische Suchansätze. Im Wesentlichen zeichnet sie eine probabilistische Eltern Selektion als primären Suchoperator aus. Als weiterer Operator wird auf die Mutation zurückgegriffen. Dieser garantiert eine Erreichbarkeit aller Punkte im Suchraum und erhält die Grunddiversität in der Population. Es gibt zwei verschiedene Algorithmen, zum einen den Standard-GA, welcher nach einer Generation die komplette Elternpopulation durch die Kinderpopulation austauscht. Dieser besteht in der Regel immer aus den gleichen fünf Schritten:
Schritt 1, Initialisieren der Anfangspopulation.
Schritt 2, Fitness berechnen mit Hilfe der Fitnessfunktion.
Schritt 3, Selektieren der Eltern.
Schritt 4, Vermehren durch Crossover und Mutation.
Schritt 5, Austausch der Populationen.
Diese Schritte sind in Abbildung 2.6 noch einmal zum leichteren Verständnis als Fluss-

diagramm dargestellt. Im Gegensatz dazu steht der Steady-State-GA, welcher sich durch eine überlappende Population auszeichnet. Pro Generation wird nur das schlechteste Individuum durch ein neues Individuum ausgetauscht. Der Steady-State-GA wird in der Arbeit nicht verwendet und wird deswegen nicht weiter erklärt, da er für die kleinen Suchräume geeignet ist. [Wei15, p. 128] [Kra17, p. 11].

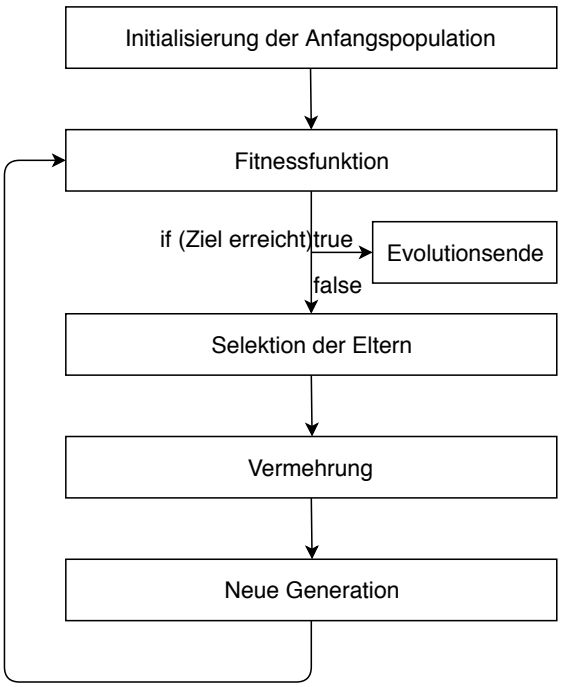

Abbildung 2.6: Ablaufdiagramm eines genetischen Algorithmus mit 5 Schritten

Algorithm 1: Genetischer Algorithmus

```
Initialisieren der Anfangspopulation;
while Fitness ≤ Abbruchbedingung do
    Fitness aller Individuen berechnen;
    Selektieren der Eltern;
    Vermehren durch Crossover und Mutation;
    Austausch der Populationen;
end
```

2.4.1 Aufbau und Initialisierung der Population

Der klassische genetische Algorithmus basiert auf einer Reihe von Kandidatenlösungen. Die Größe der Population ist somit auch die Anzahl der Lösungen. Jede Lösung kann als einzelnes Individuum gesehen werden und wird durch einen Chromosomenstrang repräsentiert. Ein Chromosom besteht wiederum aus beliebig vielen Genen, wobei jedes Gen einen Parameter repräsentieren kann. Der Aufbau ist grafisch in Abbildung 2.7 dargestellt. Um die Grundlagen nahe, des später folgenden Konzepts zuhalten, wird der Ablauf per Dezimal-Genen erklärt[Gad18, p. 134].

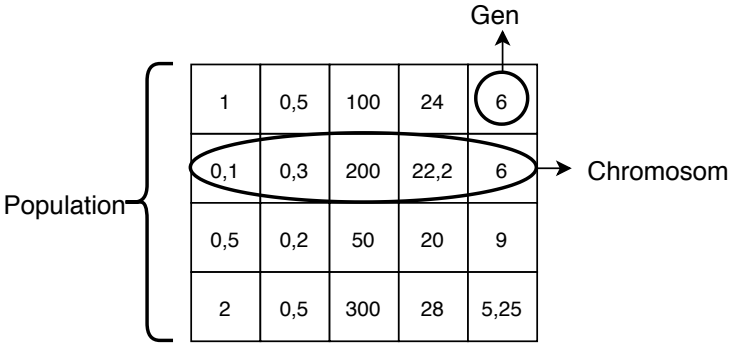

Abbildung 2.7: Beispiel einer Population mit 4 Individuen (Chromosom) mit 5 dezimalen Genen

Diese Anfangspopulation (Generation 0) wird zufällig initialisiert, um die größtmögliche Abdeckung des Suchraums zu gewähren. Die erste Generation besitzt eine sehr geringe Fitness, welche sich im Verlauf des Trainings stetig steigert bis sie das Maximum erreicht.

2.4.2 Fitnessfunktion

Die Fitnessfunktion (engl. Fitnessfunction) bewertet das Individuum anhand seiner Funktionstauglichkeit, bezogen auf die vorhandene Aufgabe. Dabei wird das Chromosom (Individuum) als Ganzes bewertet, somit wird keine Aussage über einzelne Gene getroffen. Die Fitnessfunktion muss für jede Anwendung speziell geschrieben werden. Der Rückgabewert der Fitnessfunktion entspricht einer reellen Zahl. Ein höherer Fitnesswert steht immer für eine höhere Qualität des Individuums, bzw. eine bessere Lösung[Gad18, p. 135].

2.4.3 Selektion der Eltern

Bei dem Schritt Selektion der Eltern (engl. Select Parents) geht es, darum einen Elternpool zu generieren, aus welchem die neue Generation erstellt wird. Deshalb ist es wichtig, nur die qualitativ hochwertigsten Individuen auszuwählen. Es gibt verschiedene Ansätze bei der Selektion. Die elementar Wichtigsten werden im Folgenden genannt und erläutert. Informationen zu den Selektionsmethoden wurden aus den Arbeiten [SPM15] [ES+03, p. 80] entnommen.

- **Auswahl proportional zur Fitness (engl. Fitness Proportionate Selection(FPS))** Die Eltern werden proportional nach ihrer Fitness ausgewählt und zum Elternpool hinzugefügt. $f(a_i)$ ist die Fitness des Individuums a_i in der Population. $ps(a_i)$ ist die Wahrscheinlichkeit des Individuums, a_i als Elternteil ausgewählt zu werden.

$$ps(a_i) = \frac{f(a_i)}{\sum_{j=1}^{n} f(a_j)}; i \in 1, 2, ..., n \tag{2.7}$$

  Wobei n die Anzahl der Individuen einer Population enstspricht. Diese Wahrscheinlichkeit ps wird als Anteil auf einem Rouletterad, wie Abblidung 2.8 zeigt, umgesetzt. Auf dem zufällig die Eltern aus den Individuen $a1, .., an$ „ausgedreht“ werden. Dieser Ansatz hat leider das Problem, dass Individuen, welche sich am Anfang als gut beweisen, schnell die ganze Population übernehmen. Dies kann dazu führen, dass eine mögliche bessere Lösung durch den Algorithmus im Suchraum nicht gefunden wird.

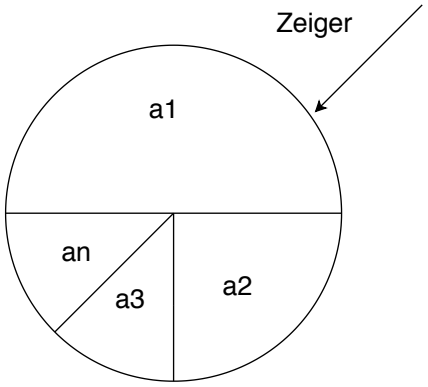

Abbildung 2.8: Rouletterad mit proportionalem Anteil der Individuen anhand ihrere Fitness

- **Ranking Selektion** Diese Selektion wurde von Backer als Verbesserung der Fitness proportional Selection entwickelt [Bak85]. Dabei werden die Eltern nicht di-

rekt nach ihrer Fitness ausgewählt. Die Fitness dient nur zum Einteilen in eine Rangliste. Anhand dieser Rangliste wird nun die Verteilung auf dem q berechnet. Dazu gibt es zwei Ansätze:

Das lineare Ranking:

$$p_i = \frac{1}{N}(n^- + (n^+ - n^-)\frac{i-1}{N-1}; i \in 1, ..., N \tag{2.8}$$

Wobei p_i die Wahrscheinlichkeit des Individuums ist, selektiert zu werden. $\frac{n^-}{N}$ ist die Wahrscheinlichkeit, des schlechtesten Individuums selektiert zu werden und $\frac{n^+}{N}$ ist die Wahrscheinlichkeit, des besten Individuums selektiert zu werden.

Das exponentielle Ranking:

$$p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}; i \in 1, ..., N \tag{2.9}$$

die Summe $\sum_{j=1}^{N} c^{N-j}$ normalisiert die Wahrscheilichkeit um Sicherzustellen, dass $\sum_{i=1}^{N} p_i = 1$. Wobei die Berechnungen 2.8 und 2.9 nur den Anteil eines Individuums auf dem Rouletterad verändern.

- **Turnier Selektion** In diesem Verfahren, welches in Abbildung 2.9 zusehen ist, werden zufällig k Individuen der Population ausgewählt. Diese k Individuen treten wie in einem Turnier gegeneinander an. Der Gewinner ist das Individuum mit dem besten Fitnesswert, dieser wird dann auch als Elternteil ausgewählt. Hierbei wird auf den Elternpool verzichtet und direkt ein Kind aus zwei Gewinnern erstellt. Eingesetzt wird dies bei kleineren Populationen mit weniger als 20 Individuen.

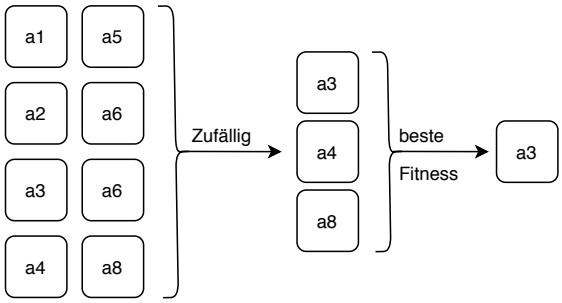

Abbildung 2.9: Turnier Selektion mit k = 3 Individuen und dem Gewinner Individuum 3

2.4.4 Vermehrung

Aus dem Elternpool werden nun Nachkommen (neue Individuen) geschaffen. Alleine durch die Paarung(engl. Crossover) von qualitativ hochwertigen Individuen wird erwartet, dass die Nachkommen eine bessere Qualität besitzen als die ihrer Eltern. Als zweite Verbesserung wird noch die Mutation einzelner Gene angewendet. In diesem Abschnitt werden die verschiedenen Ansätze für Crossover und Mutation näher erklärt.

Crossover Hierbei werden aus zwei Eltern Chromosomen, die Chromosomen der Kinder zusammengesetzt. Informationen zu Crossover wurden aus der Arbeit [US15] entnommen. Die bedeutendsten Cossover-Varianten sind:

- **One-Point-Crossover** Bei Ein-Punkt-Crossover wird zufällig ein Punkt im Chromsomenstrang festgelegt. Ab diesem Punkt wird der Chromosomenstrang dann aufgeteilt und anschließend mit dem Crossover des anderen Elternteils zusammengesetzt. Ein einfaches Beispiel ist im oberen Teil der Abbildung 2.10 zu sehen.

- **k-Point-Crossover** Eine Abwandlung des Ein-Punkt-Crossover ist das Zwei-Punkt-Crossover oder k-Punkt-Crossover. Hier wird der Chromsomenstrang an k Punkten aufgeteilt und anschließend mit dem Anteil des zweiten Elternteil wieder zusammengesetzt. Im mittleren Teil der Abbildung 2.10 ist ein k = 2 Crossover oder auch zwei-punkt-crossover (engl. two-point-crossover) zu sehen.

- **Uniform-Crossover** Eine weitere grundlegende Operation beim Crossover ist die Uniform-Crossover [Sys89]. Hier wird für jedes Gen zufällig entschieden aus welchem Elternteil das Gen entnommen wird. Dies ist im unteren Teil der Abbildung 2.10 veranschaulicht.

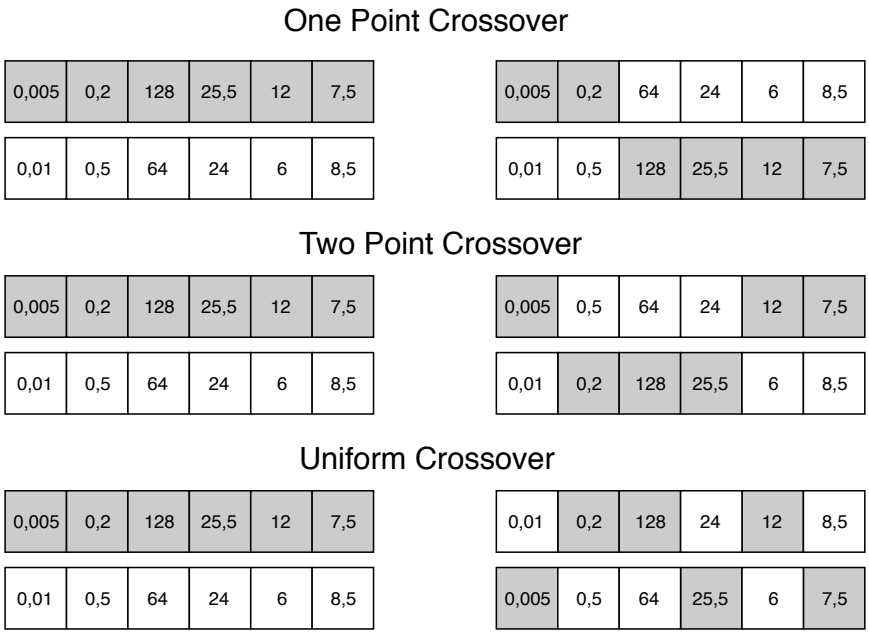

Abbildung 2.10: Die wichtigsten drei Crossover Operationen: oben one-point Crossover, mitte two-point Crossover, unten Uniform Crossover. Auf der linken Seite sind die Eltern abgebildet und auf der rechten Seite die neu erzeugten Kinder

Mutation Hierbei wird jedes Gen des Individuums zufällig mit einer zufälligen Mutation versehen. Zum einen gibt es eine einstellbare Wahrscheinlichkeit, ob ein Gen verändert werden soll und zum anderen wie groß die Mutation sein soll. Durch diese Mutation wird eine höhere Diversität in die nachfolgende Generation übergeben. Diese Mutation macht es möglich, einen größeren Suchraum abzudecken und somit die Werte genauer anzupassen, um so auf die optimale Lösung zu kommen. Es wird ein Zufallswert aus der Normal- oder Gauß-Verteilung zum Gen hinzuaddiert. Durch diese Wahrscheinlichkeitsverteilung erhalten viele Gene nur kleine Veränderungen, aber auch größere Mutationen sind möglich. Um die vorher mit Crossover neu bestimmten Individuen, welche schon eine hohe Grundfitness besitzen, nicht zu stark zu verändern, wird ein Gen nur um wenige Prozent verändert. Dies ist in Abbildung 2.11 zu sehen[Wei15, p. 60] [GLL11, p. 140].

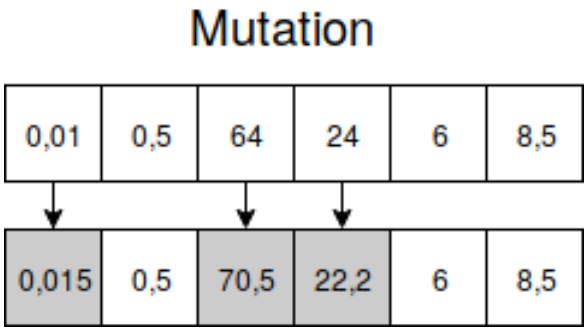

Abbildung 2.11: Beispielhafte, zufällige Mutation

2.4.5 Neue Generation

Der letzte Schritt des Genetischen Algorithmus besteht aus dem Austausch der Generationen. Die neue Kind-Generation tauscht nun die alte Eltern-Generation aus. Anschließend folgen die gleichen 4 Schritte so lange, bis der gewünschte Fitnesswert erreicht ist. Nach dem Erreichen der Abbruchbedingung, kann aus der letzten Generation das qualitativ hochwertigste Individuum ausgesucht und als Lösung verwendet werden.

2.5 Zusammenfassung

In diesem Kapitel wurden die Grundlagen erklärt, welche zum Verständnis dieser Arbeit notwendig sind. Zuerst wurden die künstlichen neuronalen Netze erklärt, welche dem Aufbau des menschlichen Gehirns gleichen. Das KNN kann aus den Daten so lineare, als auch nicht lineare zusammenhänge erlernen. Des Weiteren wurde der Genetische Algorithmus und die Zufallssuche erklärt. Bei beiden handelt es sich um Optimierungsalgorithmen für eine Mehrzieloptimierung. Die Zufallssuche ist ein zufälliges abtasten innerhalb eines Suchraums. Der Genetische Algorithmus ist der Evolution von Charles Darwin nachempfunden. Er basiert auf dem Prinzip des „Survival of the Fittest“, dies bedeutet, dass die besten Individuen sich durchsetzen und aus diesen mit Crossover und Mutation eine neue Generation erstellt wird.

Kapitel 3

Stand der Technik und Forschung

Durch die gestiegene Rechenleistung der heutigen Computer ist auch die Anwendungshäufigkeit von Genetischen Algorithmen deutlich gestiegen. Es werden nicht nur neue Ansätze und Grundlagen erforscht, sondern auch erste Anwendungen entwickelt. Diese finden nicht nur im IT-Bereich, sondern auch in den Naturwissenschaften Anwendung. Der erste Abschnitt befasst sich mit aktuellen Forschungsschwerpunkten im Bereich der Genetischen Algorithmen. Im zweiten Abschnitt wird auf die Anwendungen zu Genetischen Algorithmen eingegangen.

3.1 Stand der Forschung

3.1.1 Deep Mind

Den größten Fortschritt im Bereich der Optimierung von Hyperparametern verzeichnete Googles Deepmind. Deepmind setzt dabei auf einen populationsbasierenden Algorithmus, welcher den Genetischen Algorithmen ähnelt. Das PopulationBasedTraining [JDO+17], wird vor allem im Bereich der künstlichen Intelligenz für Brett- und Computerspiele erforscht. So konnte Deepmind bei verschiedenen künstlichen neuronalen Netzen wie AlphaGo(Go) [SHM+16] und AlphaStar(Starcraft2) [ACT19] deutliche Verbesserungen erreichen. Das PBT wird aber nicht nur für Spiele genutzt, sondern auch für Anwendungen im Bereich des autonomen Fahrens. Dort wurde in Zusammenarbeit mit der Google-Tochter Waymo eine Verbesserung der Zuverlässigkeit ihrer künstlichen neuronalen Netze ausgearbeitet. Sie konnten eine Reduzierung um 24% der falsch positiven Klassifikationsergebnisse gegenüber den von hand-modifizierten Netzen erreichen

und dies bei gleichbleibender Recallrate. Mithilfe der neu berechneten Hyperparameter, wurden die alten KNN von Waymo deutlich übertroffen und das bei der Hälfte an Trainingszeit und Ressourcen. Somit hat Google Deepmind in mehreren Fällen gezeigt, dass ihre PBT eine deutliche Verbesserung in der Optimierung von Hyperparametern ist. [hC19]

3.1.2 PopulationBasedTraining

Populationsbasiertes Training (PBT) ist eine Kombination aus Random-Search (Siehe Kap.2.3) und Genetischen Algorithmen (Siehe Kap.2.4). Der zu Grunde liegende Suchalgorithmus ist ein heuristischer Suchansatz mit einer Kandidatenlösungsmenge. Des Weiteren wurden Methoden wie „Survival of the Fittest“ und Mutation aus dem Genetischen Algorithmus übernommen. Darüber hinaus wurde ein online Anpassung der Hyperparameter während des Trainings implementiert. Schlechte Worker (Worker sind gleichzusetzen mit Individuen von GA) werden durch Worker mit besserem Fitnesswert ersetzt(exploit). Sie werden nicht einfach ersetzt, sondern erhalten kleine Mutationen(explore). Diese Worker müssen nicht von Beginn neu trainiert werden, sondern können auf die Gewichtungen der Eltern zurückgreifen. Dies ist gut in Abbildung 3.1 zusehen. Die Online-Anpassung ist dabei die größte Verbesserung. Es kann mit einem Pool beim Multiprocessing verglichen werden. Nach Beenden der Berechnung der Fitness wird direkt mit der Fitness eines neues Workers begonnen, ohne dass dabei auf andere Worker gewartet werden muss[JDO+17].

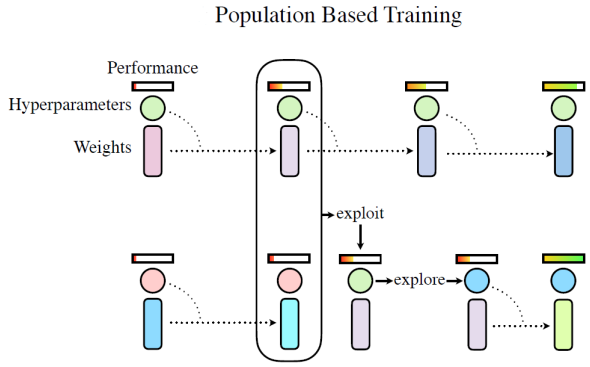

Abbildung 3.1: Population Based Training [JDO+17]

3.1.3 NeuroEvolution of Augmenting Topologies(NEAT)

NEAT ist ein Optimierungsverfahren für Modell-Architekturen von KNNs. Dabei wird die Topologie der Netze und Gewichte verbessert. Die Entwickler von NEAT konnten mit ihrem Algorithmus zeigen, dass es möglich ist, Netze zu optimieren und gleichzeitig komplexer zu gestalten. Der Algorithmus baut auf den Genetischen Algorithmen auf. Das Netz wird so einfach wie möglich initialisiert, somit wird mit schlechter Performance gestartet und über Generationen wird das Netz komplexer. Es wird mit mehr Neuronen und Verbindungen versehen und somit qualitativ verbessert. In Abbildung 3.2 ist ein Chromsomenstrang zu sehen mit den Verbindungs-Genen (engl. Connection Genes) aus welchen die Neuronen-Gene (engl. Node Genes) erstellt werden können. Über die Mutation können einzelne Verbindungen(Gewichte) hinzugefügt werden, zu sehen in Abbildung 3.3.

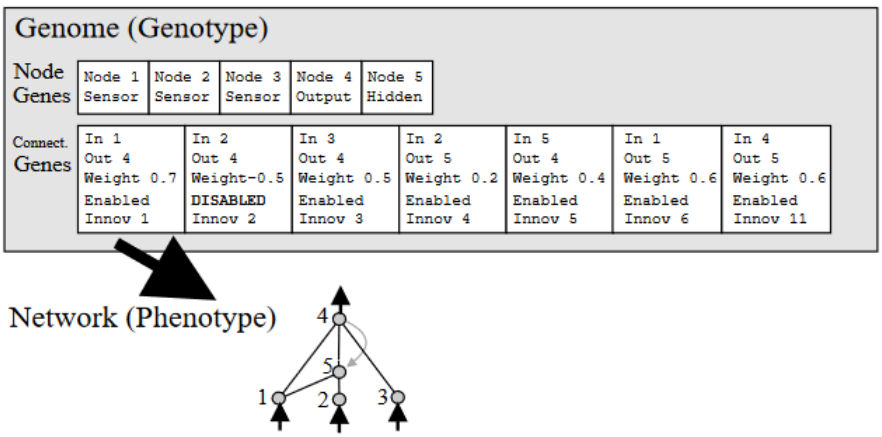

Abbildung 3.2: Aufbau eines Chromosomes bei NEAT [SM02]

Somit wird das Netz komplexer. Durch Hinzufügen von Gewichten (weights) wächst die Länge des Chomosomstrangs. Es werden zufällig auch einzelne Verbindungen ausgeschaltet, um ihre Nützlichkeit zu überprüfen. Sie können zu späteren Zeitpunkten zufällig wieder aktiviert werden. Dies ist in Abbildung 3.3 im Chromenstrang zusehen und als „DISAB“ markiert. Für die Eltern-Auswahl werden die Netze mit ähnlicher Struktur in sogenannte Spezies eingeteilt. Nun wird von jeder Spezies die schlechtere Hälfte an Individuen gelöscht. Damit ist die Eltern-Selektion nicht direkt von der Fitness abhängig und gibt schlechteren Individuen eine Überlebenschance und somit auch die Möglichkeit, sich weiter zu entwickeln. Crossover funktioniert bei NEAT ähnlich wie beim klassischen

Genetischen Algorithmus. Der Unterschied liegt darin, dass die Gene addiert werden. Dies bedeutet, Verbindungen, die nur in einem Individuum vorkommen, werden an die Nachkommen übergeben. Wenn ein Gen ausgeschaltet ist, wird es im Nachkommen mit hoher Wahrscheinlichkeit auch ausgeschaltet sein. Da nun die Nachkommen gebildet wurden, kann aus diesen eine neue Generation gebildet werden und der Optimierungsprozess beginnt von Neuem. Diese Optimierung ist nicht Teil der Arbeit und dient nur zur Information für vergleichbaren Forschungsarbeiten.[SM02]

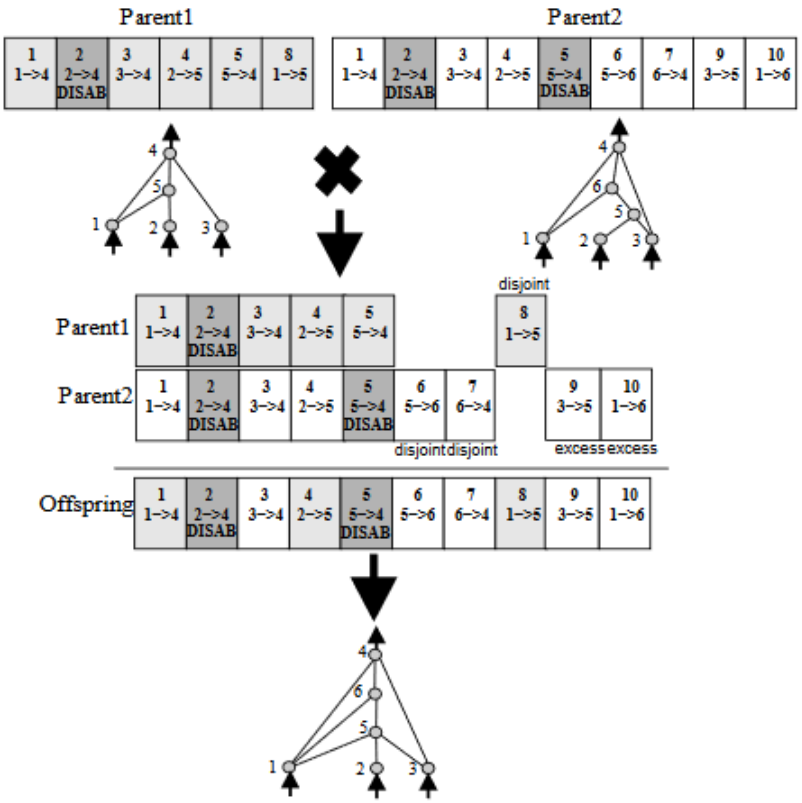

Abbildung 3.3: Crossover bei Neat [SM02]

3.2 Stand der Technik

3.2.1 Software Testing mit GA

Die Softwarebewertung spielt eine entscheidende Rolle im Lebenszyklus eines Software-Produktionssystems. Die Erzeugung geeigneter Daten zum Testen des Verhaltens der Software ist Gegenstand vieler Forschungen im Software-Engineering. So wurde von Keshavarz und Kollegen die Qualitätskontrolle mit Kriterien zur Abdeckung von Anwendungspfaden betrachtet und hierfür ein neues Verfahren auf der Grundlage eines genetischen Algorithmus zur Erzeugung optimaler Testdaten entwickelt[KJ11].

3.2.2 Temperatur Schätzungen

Genetische Algorithmen werden auch zur Berechnung von Temperaturverläufen eingesetzt [SKK12]. Als Eingangsdaten werden verschiedene Daten benutzt, wie beispielsweise: Sonnenaktivität, Vulkanausbrüche und Greenhouse-Gasanteil in der Atmosphäre. Als Ausgabe erhalten wir dann eine Abschätzung der Temperatur für die nächsten Jahre. Von den gleichen Autoren gibt es auch eine Berechnung der Abschätzungen des Wärmeflusses zwischen Atmosphäre und Meereis in Polarregionen [SKV15]. All diese Vorhersagen wurden mithilfe von der Genetischen Programmierung berechnet. Als Trainingstag wurden Aufzeichnungen von Temperaturverläufen und anderen Eingangsdaten der letzten Jahrhunderte verwendet.

3.2.3 Generatives Design

In den letzten Jahren ist die Komplexität von CAD Programmen (engl. Computer-Aided Design) deutlich gestiegen. Diese enthalten nun auch Implementierungen von Generativen Design-Werkzeugen. Mithilfe dieser Generativen Design-Werkzeugen ist es möglich, aus klassischen Bauteilen bionische Bauteile zu entwicklen. Hierfür werden die Bauteile über Iterationen optimiert. So kann aus einem Bauteil, ein wesentlich leichteres und materialsparenderes Modell entwickelt werden. In Abbildung 3.4 sind verschiedene Schritte des Generativen Designs zu sehen. Die Fertigung ist meist nur mit additiven Fertigungsmethoden möglich, da die berechneten Strukturen keinen Fertigungsregeln unterliegen[Cal08].

Abbildung 3.4: Additives Design Beispiel von Siemens [sie19]

NASA - Antenne Dieser Generative Algorithmus wurde schon 2006 von der NASA erforscht. Hierbei wurde eine Weltraumantenne mit Hilfe von evolutionären Algorithmen entwickelt. In Abbildung 3.5 sind zwei Prototypen der Weltraumantenne zu sehen. Die Entwicklung einer solchen Antenne braucht enormes Wissen in verschiedenen Domänen wodurch die Entwicklung sehr arbeitsintensiv und somit auch Zeitaufwändig wäre. Deshalb benutzen die Forscher einen populationsbezogenen Suchansatz, um Umgebungsstrukturen und elektromagnetische Auswirkungen mit einzubeziehen. Die entwickelte Antenne wurde produziert und auch auf der Space Technology 5 Mission genutzt [HGLL06].

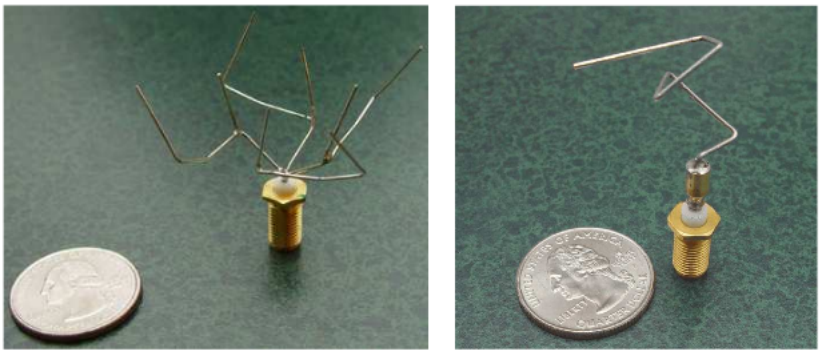

Abbildung 3.5: Foto der Prototypen [HGLL06]

3.3 Zusammenfassung

In diesem Kapitel wurde der Stand der Technik und Forschung für Optimierung und Genetische Algorithmen erläutert. Dazu wurde erst das Populations Basiertes Training(PBT) von Google Deepmind erläutert, welches auf den Genetischen Algorithmen basiert und zur Optimierung von Hyperparametern bei künstlichen neuronalen Netzen angewendet wird. Des Weiteren wurde NeuroEvolution of Augmenting Topologies(NEAT) vorgestellt, dies wird auch zur Optimierung von KNN verwendet. NEAT basiert auch auf dem Genetischen Algorithmus, verwendet aber einen anderen Ansatz wie das PBT oder der klassische GA. Bei NEAT wird die Modell-Architektur optimiert. Dazu wird mit dem kleinstmöglichen Netz angefangen und anschließend über viele Iterationen Verbindungen und Neuronen hinzugefügt, so dass sich die Qualität des Netzes verbessert. Abschließend wurden einige Beispiele aufgezählt, in welchen GA angewendet werden. Diese sind: Temperaturabschätzung, Software Testing und Generatives Design.

Kapitel 4

Konzept

Der Genetische Algorithmus(GA) wird zur Optimierung der Hyperparameter von künstlichen neuronalen Netze benutzt, da der GA in großen Suchräumen effizient arbeitet. Ein Nachteil des GA ist der hohe Rechenaufwand, weshalb im Konzept- und Implementierungsteil die Rechenzeiten beachtet werden. Um ein Konzept zu entwickeln, werden zuerst die Anforderungen an den GA definiert. Darauf aufbauend wird das Konzept des Genetischen Algorithmus erläutert. Abschließend wird auf das Konzept der Evaluierung und auf die Auswertung eingegangen.

4.1 Anforderungen

Um einen Genetischen Algorithmus umzusetzen, sind folgende Anforderungen an die Arbeit gestellt:

- Automatisches Trainieren eines künstlichen neuronalen Netzes
- Automatisches Evaluieren dieser Netze
- Automatisches Speichern der Ergebnisse
- Automatischen Auswerten der Ergebnisse

Diese Anforderungen sind zur Umsetzung des Genetischen Algorithmus essentiell. Um den GA zu evaluieren, ist ein Algorithmus zum Vergleichen notwendig. Für diesen wurde die Zufallssuche ausgewählt, da dieser in der Literatur zur Optimierung der Hyperparameter empfohlen wird. Für beide Algorithmen gelten die gleichen, genannten Anforderungen, weshalb die spätere Implementierung möglichst modular aufgebaut wird. Somit

entstehen aussagekräftige Ergebnisse, die abschließend verglichen und ausgewertet werden.

4.2 Genetischer Algorithmus

Für die Anwendung der genetischen Algorithmen zur Optimierung von Hyperparametern von KNN's, wird im Folgenden ein Konzept vorgestellt. Dieses basiert auf den in Kapitel 2.4 beschriebenen 5 Schritten:

1. Initialisieren der Anfangspopulation
2. Fitness berechnen mit Hilfe der Fitnessfunktion
3. Selektieren der Eltern
4. Vermehren durch Crossover und Mutation
5. Neue Generation

Diese Schritte sind in Abbildung 2.6 als Flussdiagramm dargestellt. Um den Genetischen Algorithmus zur Optimierung von Hyperparametern zu verwenden, werden als Gene die Hyperparameter des künstlichen neuronalen Netzes eingesetzt. Somit steht ein Individuum für ein künstliches neuronales Netz mit speziellen Hyperparametern. Dies ist in Abbildung 4.1 zu sehen. Hierdurch werden die Gene bzw. Hyperparameter über Generationen (mehrere Iterationen) optimiert, so dass die Klassifikationsgenauigkeit steigt. Außerdem wird geprüft, inwiefern sich der Genetische Algorithmus zur Optimierung der Modell-Architektur eignet. Der Fokus der Arbeit liegt auf der Optimierung der klassischen Hyperparameter: Lernrate, Dropout, Epochen, Batchsize, Optimierer. Diese sind im Grundlagenkapitel 2.1.5 definiert worden. In der Literatur wird kein Aussage zur Funktionstauglichkeit der Methoden getätigt, weswegen Experimente zu ihrer Funktionstauglichkeit durchgeführt werden. In diesen Experimenten wird der GA zur Optimierung der Hyperparameter eines künstlichen neuronalen Netzes zur Bildklassifizierung eingesetzt. Das Experiment, sowie die exakt implementierten Methoden, werden in Kapitel 5.2 detailliert beschrieben. Nachfolgend werden die Konzepte der einzelnen Schritte erläutert. Das Wissen zu den Grundlagen der Schritte und der Methoden des Genetischen Algorithmus wird für dieses Kapitel als vorausgesetzt angenommen und kann im Grundlagenkapitel 2.4 nachgelesen werden.

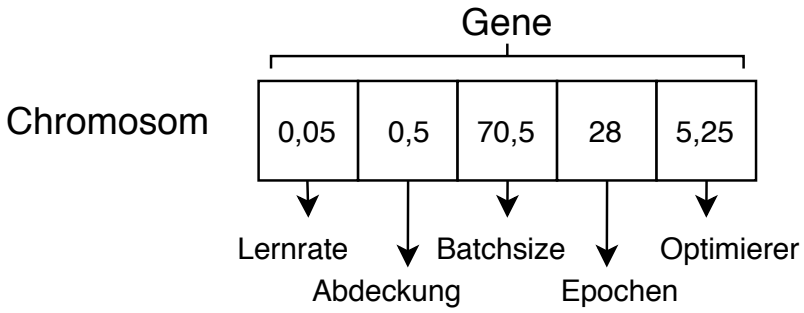

Abbildung 4.1: Beispielhafte Zeichnung des Chromosoms eines Individuums, in dem die Hyperparameter als Gene realisiert wurden

4.2.1 Initialisierung der Anfangspopulation

Die Anfangspopulation wird zufällig innerhalb eines bestimmten Wertebereichs initialisiert. Die jeweiligen Wertebereiche sind für jeden HP individuell und müssen durch Experimente bestimmt werden. Wenn dieser Wertebereich nicht eingehalten wird, kann es dazu führen, dass die trainierten künstlichen neuronalen Netze nicht brauchbar sind. Anschließend wird die Anfangspopulation zufällig, in einem für jeden Hyperparameter speziellen Rahmen initialisiert. Diese Randbedingungen beschränken den Suchraum gezielt. Gleichzeitig müssen sie so gewählt werden, dass alle zielführenden Lösungen enthalten sind. Durch die Reduzierung des Suchraums, gestaltet sich die anschließende aufwändige Berechnung effektiver. Die exakte Größe der Population sowie die Rahmenvorgaben der Hyperparemeter wird im Implementierungsteil 5.2.1 angeführt und begründet.

4.2.2 Fitnessfunktion

Die Fitnessfunktion ermittelt die Performance der Individuen, bezogen auf ihre Aufgabe. Das Maß mit dem die Performance gemessen wird, wird als Fitnesswert bezeichnet. Bei der Optimierung von Hyperparametern ist der Fitnesswert mit der Genauigkeit der Klassifikation auf dem Testdatensatz gleichzusetzen. Es soll versucht werden, die Genauigkeit zu Maximieren, wobei das Maximum der Genauigkeit bei 1 liegt. Dementsprechend ist die Fitnessfunktion, das Trainieren eines Netzes mit den entsprechenden Hyperparametern und einem anschließendem Zurückgeben der Klassifizierungsgenauigkeit. Die exakte Fitnessfunktion wird im Implementierungskapitel 5.2.2 definiert. Darüber hinaus sollen Versuche durchgeführt werden, in denen der Fitnesswert nicht nur von der Klassifizierungsgenauigkeit abhängt, sondern aus einer Kombinationen von Interferenz-

geschwindigkeit und Klassifikationsgenauigkeit. Somit könnten Modell-Architekturen für spezielle Anwendungen mit schneller Performance oder schnellem Training gefunden werden.

4.2.3 Selektion der Eltern

Zum Erstellen einer neuen Population müssen erst die Eltern ausgesucht werden. Da in der Literatur keine bestimmte Selektion der Eltern favorisiert wird. Werden im Rahmen der Arbeit für die Eltern Selektion die Fitness proportional Selektion und die Ranking Selektion verglichen. Die Methode mit den besten Ergebnissen wird anschließend zur Evaluation eingesetzt. Bei diesen Methoden handelt es sich um die State-of-the-Art Methoden, die im Grundlagen Kapitel 2.4.3 beschrieben wurden. Die verwendeten Methoden werden im Implementierungskapitel 5.2.3 beschrieben.

Vermehrung

Nachdem die Eltern Ausgewählt sind, können aus diesen die neue Generation erstellt werden, dazu kommen die zwei Methoden Crossover und Mutation zur Anwendung. Auf unterschiedlichen Umsetzungen der Methoden wird im Kapitel 2.4.4 eingegangen. Für die Methoden des Crossovers gibt es in der Literatur keine Aussagen über ihre Funktionstauglichkeit, weshalb für das **Crossover** die Ein-Punkt-Crossover gegenüber Zwei-Punkt-Crossover und Uniform-Crossover verglichen werden. Als **Mutation** wird eine zufällige Mutation mit einer Normalverteilung benutzt. Die verwendeten Methoden werden im Implementierungskapitel 5.2.3 beschrieben.

4.2.4 Neue Generation

Mit Hilfe der zuvor durchgeführten vier Schritten wurden die neue Individuen ausgewählt. Diese bilden die neue Generation, welche die alte Generation austauscht. Die Größe der neuen Populationen entspricht der Anfangspopulation. Um die Optimierung zu verbessern, wird geprüft, ob die Übernahme einzelner Individuen vorteilhaft ist. Die Optimierung kann nach dem Austauschen der Generationen erneut durchgeführt werden. Die 5 Schritte können für beliebig viele Generationen erfolgen oder bis zum Erreichen eines bestimmten Fintesswerts durchgeführt werden.

4.3 Evaluierungs- und Auswertungskonzept

Für diese Arbeit ist eine ausführliche Evaluation und Auswertung essentiell. Daher ist es wichtig ein Konzept zur späteren Evaluierung und Auswertung zu entwickeln. Dieses wird speziell auf die Algorithmen der Arbeit angepasst. Nachfolgend wird auf die Konzepte der Evaluierungen der Methoden und Optimierung eingegangen.

4.3.1 Evaluierung der Optimierung

Um den Genetischen Algorithmus zur Optimierung der Hyperparameter zu evaluieren, werden die Ergebnisse des GA mit den Ergebnissen der Zufallssuche gegenübergestellt. Um eine Gegenüberstellung durchführen zu können, müssen die Laufzeiten der Algorithmen gleich gesetzt werden. Dies ist über die Trainingsvorgänge möglich, da diese 95% der Rechenzeit beanspruchen. Die genaue Bestimmung der Iterationen wird in Kapitel 5.4 beschrieben. Darüber hinaus wird die Optimierung an zwei verschiedenen Netzen durchgeführt, damit ein aussagekräftiges Ergebnis zustande kommt. Des Weiteren wird eine Optimierung mit kleinen Datensätzen durchgeführt. Hierbei wird geprüft, ob sich die Optimierung der Hyperparameter, mit der Optimierung von großen Datensätzen unterscheidet.

4.3.2 Auswertung

Alle Ergebnisse werden automatisch mit den dazugehörigen Konfigurationen und Berechnungen in einer Datei abgespeichert. Aus dieser Datei wird dann automatisch eine Zusammenfassung aller Ergebnisse erstellt. Zudem sollen aus den Ergebnissen Diagramme erstellt werden, welche eine intuitive Auswertung ermöglichen.

4.4 Zusammenfassung

In diesem Kapitel wurde das Konzept zur Optimierung der Hyperparameter von künstlichen neuronalen Netzen vorgestellt. Zuerst wurden die Anforderungen für die Arbeit bestimmt. Die Idee besteht darin die Hyperparameter als Gene des Genetischen Algorithmus umzusetzen. Somit steht jedes Individuum für ein KNN mit speziellen Hyperparametern. Diese Gene bzw. Hyperparameter werden mit Hilfe von Crossover und Mutationen über Generationen optimiert. Somit sollten bessere Ergebnisse bei der Klassifizierung von Bilddaten geliefert werden. Zudem wird die Optimierung für zwei unterschiedliche Datensätze und Netze eingesetzt, wodurch ein aussagekräftiges Ergebnis entsteht. Des Weiteren werden die Ergebnisse automatisch mit ihren dazugehörigen Konfigurationen abgespeichert und ausgewertet.

Kapitel 5

Implementierung

In diesem Kapitel werden die Implementierungen dieser Arbeit erläutert. Zuerst wird der Systemaufbau erklärt und alle verwendeten Module ausführlich beschrieben. Anschließend wird auf die verwendete Hardware eingegangen. Im Folgenden werden die implementierten Methoden des GA und der Zufallssuche genauer betrachtet, indem die exakten Methoden und künstlichen neuronalen Netze erklärt werden. Zum Schluss wird auf die implementierten Evaluationen und Auswertungen eingegangen.

5.1 Systemaufbau

Das Gesamtsystem wird mit Python als Programmiersprache umgesetzt. Für die Implementierung werden zusätzliche Python Pakete verwendet. Für das Erstellen und Trainieren der künstlichen neuronalen Netze wird auf Tensorflow zurückgegriffen. Tensorflow ist eine End-to-End-Open-Source Plattform für maschinelles Lernen (ML). Zudem bietet Tensorflow ein umfassendes und flexibles Ökosystem aus Werkzeugen und Bibliotheken für ML-Anwendungen. Darüber hinaus hat Tensorflow eine starke Community, welche vorallem im ML-Bereich viele Open-Source-Projekte entwickelt und veröffentlicht. Für die Zufallssuche wird SkLearn verwendet. Bei SkLearn handelt es sich um ein simples und effizientes Werkzeug zur prädiktiven Datenanalyse. Dies beinhaltet Funktionen wie: Klassifikation, Regression, Clustering, Support Vektor Maschine. Mit Hilfe von SkLearn werden die bewertungsmatrixen zum Auswerten der Algorithmen umgesetzt. Zum Darstellen von Diagrammen wird Seaborn und Matplotlib verwendet. Der Genetische Algorithmus wurde eigenständig implementiert. Für diesen wurden nur die Pakete Numpy und Pandas verwendet. Numpy ist ein Paket für Multi-Dimensionen Arrays und bietet

eine große Anzahl an mathematischen Funktionen. Pandas ist wie Numpy für Arrays und Datenframes zuständig. Im Detail ist Pandas für Datenstrukturen und Manipulation solcher Datenarrays entwickelt worden. Da die Rechenzeit ein entscheidender Faktor der Optimierung darstellt, wird das Paket Multiprocessing verwendet. Dies führt vor allem bei Multi-Core-Servern zu schnelleren Berechnungen. Das Gesamtsystem wurde für CPU entwickelt. Darüber hinaus ist der Genetischen Algorithmus als GPU (engl. graphics processing unit) optimierte Implementierung vorhanden. Dies beschleunigt die Berechnungen von großen KNNs deutlich. All diese Einstellungen können per Argumente an das Python Skript übergeben werden. Somit ist es einfach möglich andere Berechnungen durchzuführen, auch solche, die in dieser Arbeit nicht besprochen werden. Eine ausführliche Auflistung aller verwendeten Pakete ist in Abbildung 5.1 zu sehen.

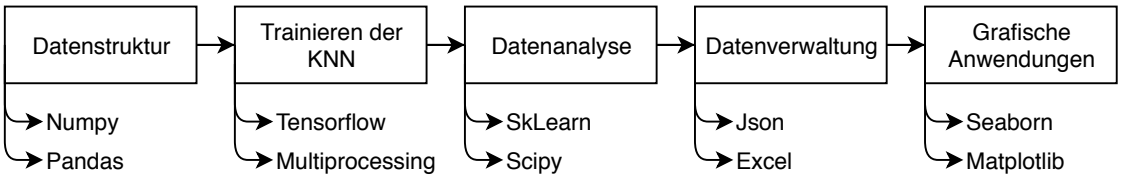

Abbildung 5.1: Datenfluss und verwendete Python Pakete, die zur Umsetzung der Arbeit verwendet wurden

5.1.1 Hardware

Um die Optimierungen durchzuführen stehen drei Hardwaremöglichkeiten zur Auswahl. Dazu gehört ein Server mit i5-5550 Prozessoren mit 40 Kernen ohne GPU, ein Server mit einer Grafikkarte (Gtx1080Ti) um die Berechnungen durchzuführen und eine Server mit einem i7-7700 Prozessor mit 8 Kernen ohne GPU. Diese sind tabellarisch in Tabelle 5.1 aufgeführt. Im späteren Kapitel 5.4 wird auf die Benchmarks der Hardware eingegangen.

Tabelle 5.1: Zur Verfügung stehende Hardware

Hardwarekomponenten	**Server 1**	**Server 2**	**Server 3**
Prozessor	E5-2630	i7-7700	i7-7700
Logische Kerne	40	8	8
Arbeitsspeicher	256 GB	8 GB	32 GB
Grafikkarte	-	-	GTX 1080 Ti (11 GB)

5.1.2 Datensatz

Um die künstlichen neuronalen Netze zu trainieren wurden Bilddaten als Daten-Grundlage ausgewählt. Bei den verwendeten Datensätzen handelt es sich um gängige Datensätze, die oft zum Vergleichen von Klassifikationsergebnissen von KNN genutzt werden. Folgende Datensätze wurden ausgewählt: Mnist Faschion, Mnist Digits und Cifar10. Diese drei Datensätze werden benutzt, da sie sich sehr ähnlich sind. Alle drei besitzen eine gleichmäßige Verteilung der Daten in Bezug auf die Klassen. Die Datensätze besitzen jeweils 10 Klassen. Die Besonderheit der Datensätze liegt auf ihrer guten Vergleichbarkeit. Die einzigen Unterschiede sind die Klassen und Bildformate. Alle verwendeten Datensätze werden wie folgt aufgeteilt: 65% Trainingsdaten, 20% Validierungsdaten, 15% Testdaten. Somit wird das Overfitten der Netze verhindert, da die Daten klar getrennt werden. Nachfolgend werden die Datensätze genauer beschrieben.

Mnist Fashion Datensatz Zur Evaluierung der Methoden des Genetischen Algorithmus wurde der „Mnist Fashion“ Datensatz benutzt. Dieser Datensatz enthält 10 Klassen für verschieden Kleidungsstücke. Er beinhaltet die Klassen: T-shirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot. Der Mnist Fashion Datensatz besteht aus 70.000 Schwarz-weiß Bildern mit je 28x28 Pixeln. Ein Beispiel aus dem Datensatz ist in Abbildung 5.2 auf der linken Seite zusehen.

Mnist Digits Datensatz Zum Evaluieren der Optimierung des Fully Conneceted Netzes wird der „Mnist Digits“ Datensatz benutzt. Dieser besteht aus handgeschrieben Ziffern zwischen 0 und 9. Der Mnist Digits Datensatz besteht aus 70.000 Schwarz-weiß Bildern mit je 28x28 Pixeln. Ein Beispiel des Mnist Digits Datensatze ist in der Mitte der Abbildung 5.2 zusehen. Um die Evaluation der Optimierung nicht nur mit einem Datensatz durchzuführen, wird ein dritter Datensatz verwendet.

Cifar 10 Datensatz Hierbei handelt es sich um den CIFAR 10 Datensatz. Er besitzt die Klassen: Flugzeug, Auto, Vogel, Katze, Reh, Hund, Pferd, Schiff und LKW. Der Datensatz ist aus 60.000 farbigen Bildern (RGB) mit je 32x32 Pixel aufgebaut. Ein Beispiel ist auf der rechten Seite der Abbildung 5.2 zusehen.

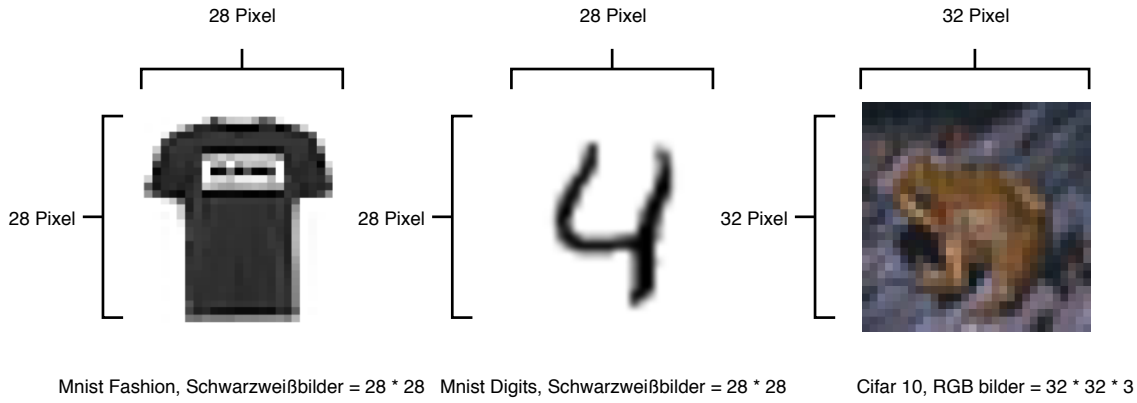

Abbildung 5.2: Beispiele aus den verwendeten Datensätze

5.2 Genetischer Algorithmus

Die Implementierung des Genetischen Algorithmus entspricht den 5 Schritten des Genetischen Algorithmus, das Konzept wurde in Kapitel 4.2 beschrieben. In den folgenden Unterkapitel werden die implementierten Methoden im Detail besprochen. Zusätzlich wird auf speziellen Umsetzungen und Eigenschaften eingegangen. Da es zu den meisten Methoden in der Literatur wenig Informationen zu ihren Auswirkungen auf das Endergebnis gibt, wird ein Experiment zur Bestimmung der zielführenden Methoden durchgeführt. Für dieses Experiment wurde eine Optimierung der Hyperparameter durchgeführt. Als Netz wird ein Fully-Connected Netz mit einem Layer und 128 Neuronen verwendet, die Größe des Netzes wird im Abschnitt 6.6 ausführlich beschrieben. Es wurde diese Größe gewählt, da die spätere Evaluation mit der gleichen Größe an Netz durchgeführt wird. Als Datengrundlage wurde der Mnist Fashion Datensatz verwendet, linke Seite der Abbildung 5.2. Die Methoden mit den die besten Ergebnisse gefunden wurden, werden zur Evaluierung des Genetischen Algorithmus eingesetzt. Das Wissen zu den Grundlagen der Methoden des Genetischen Algorithmus wird für dieses Kapitel als voraussetzt angenommen und kann im Grundlagenkapitel 2.4 nachgelesen werden.

5.2.1 Initialisierung der Anfangspopulation

Die Größe der Anfangspopulation kann beliebig bestimmt werden. Im späteren Evaluationsabschnitt 5.4 wird die genauen Größe definiert, da diese im direkten Zusammenhang mit der Laufzeit steht. Der Suchraum in dem die Hyperparameter zufällig initialisiert werden, ist in Tabelle 5.2 zusehen. Die Grenzen des Suchraums wurde mithilfe der Literatur und des beschriebenen Experiments bestimmt. Die Initialisierung im Suchraum ist zufällig mit einer gleichmäßigen Verteilung umgesetzt. Diese ist als Formel in Eq. 5.1 abgebildet.

$$p(x) = \frac{1}{b-a} \tag{5.1}$$

a und b bilden die Intervallgrenzen.

Tabelle 5.2: Suchraum des Genetischen Algorithmus

Hyperparameter	**Minimum**	**Maximum**
Prozessor	0.000005	0.1
Logische Kerne	0.05	0.5
Arbeitsspeicher	10	80
Grafikkarte	0	7

Hierbei steht jede ganze Zahl beim Optimierer für einen eigenen Optimierer. Hierbei entspricht: 0 = adam, 1 = SGD, 2 = RMSprop, 3 = Adagrad, 4 = adadelta, 5 = adammax, 6 = nadam, 7 = ftrl

5.2.2 Fitnessfunktion

Die Fitnessfunktion beinhaltet den rechenaufwendigsten Teil der Arbeit. In dieser werden die KNNs berechnet. Dazu werden die Gene (Hyperparameter) als Input verwendet. Anschließend werden die Daten geladen, das Netz trainiert und anschließend evaluiert. Der Rückgabewert ist die Klassifizierungsgenauigkeit, die dem Fitnesswert entspricht. Die Fitnessfunktion ist in Abbildung 5.3 als Whitebox dargestellt. Die Fintessfunktion wird für unterschiedliche KNN erstellt. Diese Netze sind im Kapitel 5.4 beschreiben.

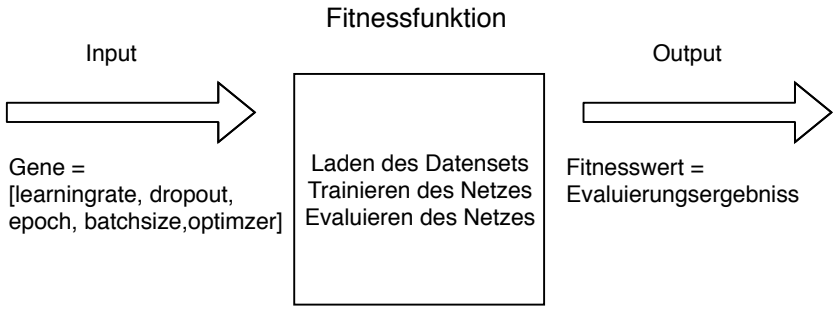

Abbildung 5.3: Beispielhafte Whitebox der Fitnessfunktion mit Genen als Input und Klassifikationsgenauigkeit als Output

5.2.3 Selektion der Eltern

Aus dem Experiment hat sich gezeigt, dass bei der Selektion der Eltern die Fitness proportional Selektion(FPS) die besseren Ergebnisse im Vergleich zu der Ranking Selektion liefert. Aus diesem Grund wird die FPS bei der Evaluation des Genetischen Algorithmus verwendet.

Vermehrung

Für das **Crossover** wurde Two-Point Crossover und Uniform-Crossover implementiert. Für die **Mutation** wurde die Gauss-Mutation mit der Gaussdichteverteilung, mit Sigma gleich 0.1 (Eq. 5.2) und eine Mutation mit gleichmäßiger Verteilung (Eq. 5.1) implementiert. Für die spätere Evaluation der Optimierung mit dem GA wurde Two-Point Crossover und die Gauss-Mutation ausgewählt, da sie in den Experimenten die zielführendsten Ergebnisse lieferten.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.2}$$

5.2.4 Neue Generation

Die Population der neuen Generation wird mit den genannten Methoden ausgewählt. Diese sind in Abbildung 5.4 noch einmal dargestellt. Die Experimente zeigten, dass die Optimierungsergebnisse sich steigern lassen, wenn die besten zwei Individuen der alten Generation, ohne Mutation und Crossover, in die nächste Generation übergeben werden. Dies kann für beliebig viele Generationen durchgeführt werden oder bis zum Erreichen einer Abbruchbedingung.

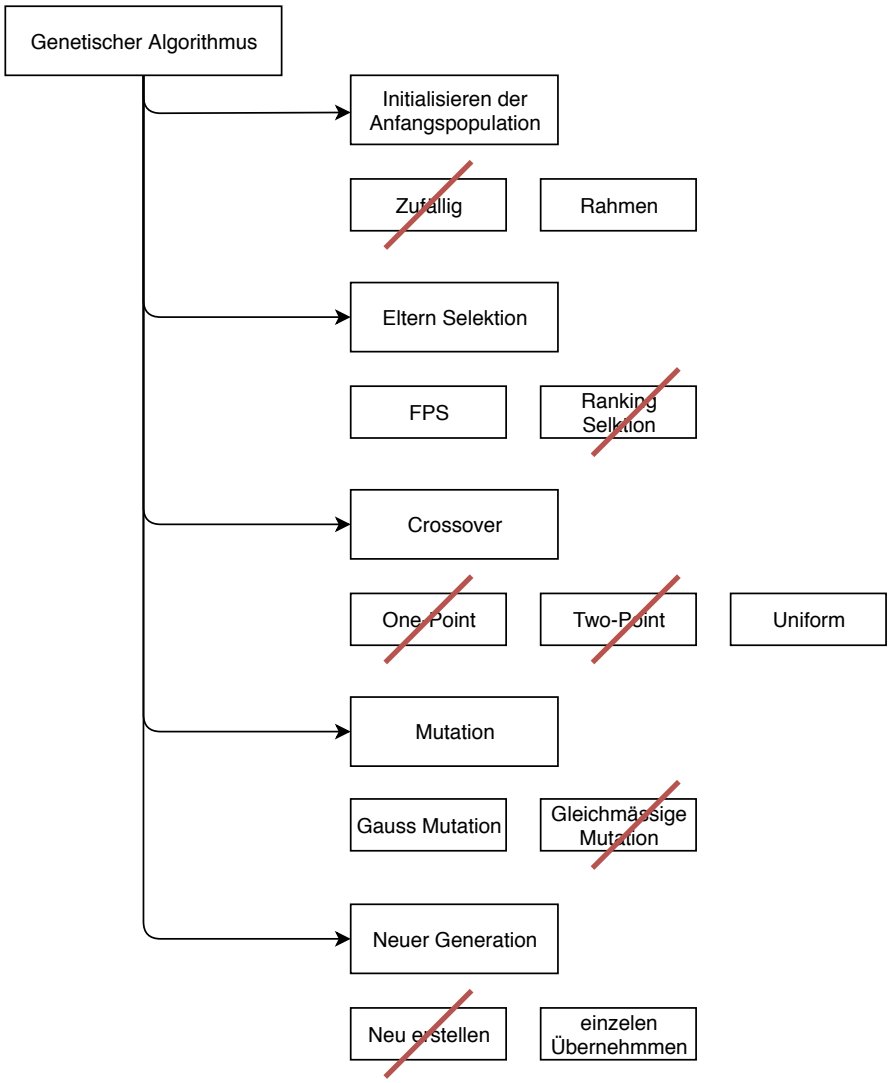

Abbildung 5.4: Implementierte und verwendete Methoden des Genetischen Algorithmus

5.3 Zufallssuche

Die Zufallssuche wird mit Hilfe der Python Bibliothek SkLearn umgesetzt. Die implementierte Zufallssuche ist, wie im Grundlagenkapitel 2.3 beschrieben, aufgebaut. Für den Suchraum der Zufallssuche gelten die gleichen Rahmenbedingungen wie für den Genetischen Algorithmus. Im Vergleich zum GA wird der Suchraum bei der Zufallssuche in ein Raster aufgeteilt. Dieses Raster ist in Tabelle 5.3 zusehen.

Tabelle 5.3: Suchraum der Zufallssuche

Hyperparameter	**Min**							**Max**
Lernrate	0.00005	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1
Dropout	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Epochen	10	20	30	40	50	60	70	80
Batchsize	8	16	32	40	48	56	64	72
Optimizer	0	1	2	3	4	5	6	7

Jede ganze Zahl des Optimizer steht für einen eigenen Optimierer. Hierbei entspricht 0 = adam, 1 = SGD, 2 = RMSprop, 3 = Adagrad, 4 = adadelta, 5 = adammax, 6 = nadam, 7 = ftrl.

5.4 Evaluierung

Zur Evaluierung wird die Berechnungszeit der beiden Algorithmen gleichgesetzt. Da das Trainieren der KNN 95% der Berechnungszeit der Algorithmen ausmacht, wird die Berechnungszeit über die Anzahl der Trainingsvorgänge begrenzt. Durch Versuche ergab sich die Evaluierung von 50 und 250 Iterationen als zielführend. Eine weitere Verdopplung der Iterationen von 250 auf 500, zeigte bei den Versuchen keine Verbesserung der Ergebnisse. Trotz des doppelten Rechenaufwandes. Aus diesem Grund ist die Berechnung von 500 Iterationen nicht umgesetzt worden. Für die Zufallssuche ergibt sich somit 50 und 250 Iterationen. Für den Genetischen Algorithmus ergibt sich für 50 Iterationen eine Populationsgröße von 25 Individuen mit 2 Generationen. Für 250 Iterationen wird im GA eine Populationsgröße von 50 Individuen und 5 Generationen gewählt. Um aussagekräftige Ergebnisse zu erhalten wurden zwei KNNs implementiert:

5.4.1 Hyperparameter Optimierung - Fully Connected Network

Es wird ein Fully Connected Network zur Klassifizierung von handgeschriebenen Ziffern implementiert. Darüber hinaus werden zwei verschiedene Größen des künstlichen neuronalen Netzes implementiert, um mögliche Unterschiede der Modellgröße auf die Optimierung zu erkennen. Für das Fully-Connected-Network wurden diese Größen von KNNs implementiert:

- **kleines KNN** mit einem Layer und 128 Neuronen. Die exakte Modell-Architektur ist in der nachfolgenden Auflistung zusehen. Die 28x28 Pixel der Eingangsdaten werden zu 784 Pixel mit nur einer Dimension flachgedrückt (engl. flatted). Anschließend kommt die Versteckteschicht (engl. hidden Layer) mit 128 Neuronen und einer Dropoutschicht, die für jedes Neuron durchgeführt wird. Abschließend kommt die Ausgangsschicht mit 10 Klassen, wodurch sich 10 Neuronen in der Ausgangsschicht ergeben. Dies entspricht einer Anzahl von 101.770 trainierbaren Parametern. Somit ist dieses Netz im Vergleich zu State-of-the-Art Netzen relativ klein. Diese Netz ist beispielhaft als Figur 5.5 und als Auflistung in der Liste 5.1 zusehen.

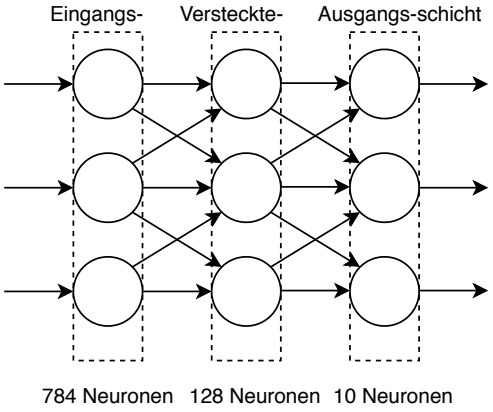

Abbildung 5.5: Beispielhafte Darstellung des kleinen KNNs

Beispielhaft wird die Trainingsdauer des kleinen Netzes für 10 und 80 Epochen evaluiert. Bei 10 Epochen dauerte der Trainingsvorgang 34 Sekunden und für 80 Epochen 252 Sekunden. Diese Werte wurden auf einem Intel i7-7700 3,6 GHz erreicht.

Listing 5.1: Exakte Modell-Architektur des kleinen Fully-Conneted-Networks

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 101,770

- **großes KNN** mit zwei Layer je 128 Neuronen. Die exakte Modell-Architektur ist in der nachfolgenden Auflistung zusehen. Die 28x28 Pixel der Inputdaten werden zu 784 Pixel mit nur einer Dimension flachgedrückt. Anschließend kommt die erste Verstecktеschicht mit 128 Neuronen und einer Dropoutschicht, die für jedes Neuron durchgeführt wird. Nun kommt erneut eine Verstecktеschicht und Dropout mit 128 Neuronen. Abschließend kommt die Ausgangsschicht mit 10 Klassen, wodurch sich 10 Neuronen in der Ausgangsschicht ergeben. Dies entspricht einer Anzahl von 134.794 trainierbaren Parametern. Somit ist das Netz im Vergleich zu State-of-the-Art Netzen relativ klein. Diese Netz ist beispielhaft als Figur 5.6 und als Auflistung in der Liste

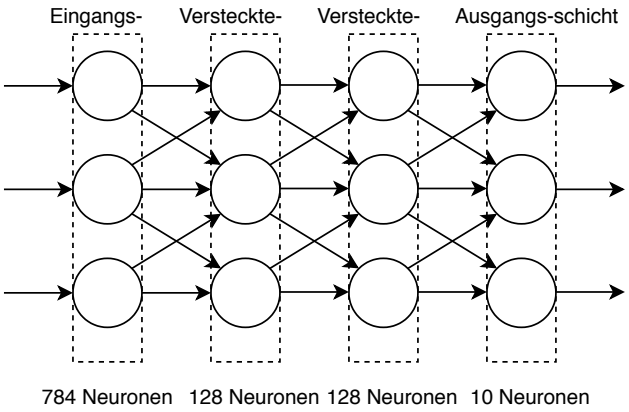

Abbildung 5.6: Beispielhafte Darstellung des großen KNNs

Beispielhaft wird die Trainingsdauer des großen Netzes für 10 Epochen und 80 Epochen evaluiert. Bei 10 Epochen dauerte der Trainingsvorgang 44 Sekunden und für 80 Epochen 339 Sekunden. Diese Werte wurden auf einem Intel i7-7700 3,6 GHz erreicht.

Listing 5.2: Exakte Modell-Architektur des großen Fully-Conneted-Networks

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1290

Total params: 134,794

5.4.2 Hyperparameter Optimierung - Convolutional Neural Network

Es wurde ein faltendes neuronales Netz (engl. Convolutional Neural Network - CNN) zur Klassifizierung von kleinen RGB Bildern aus dem Cifar10 Datensatz implementiert. Die exakte Modell-Architekturen ist in der nachfolgenden Auflistung zusehen. Beim CNN werden die Pixel nicht flachgedrückt, sondern können als Matrix geladen werden. Anschließend kommen verschiedene Kombinationen aus Convolutionalschicht, Activationsschicht, Dropoutschicht und Maxpoolingschicht. Diese Schichten besitzen deutlich mehr

Parameter. Dieses CNN besitzt 1.250.858 trainierbare Parameter und besitzt damit fast 10 Fach so viele trainierbaren Parametern wie die FCN.

Beispielhaft wird die Trainingsdauer des CNN-Netzes für 10 Epochen und 80 Epochen evaluiert. Bei 10 dauerte der Trainingsvorgang 15 Minuten und für 80 Epochen 125 Minuten. Diese Werte wurden auf einem Intel i7-7700 3,6 GHz erreicht.

Listing 5.3: Exakte Modell-Architektur des Convolutional Neural Network

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 30, 30, 32)	9248
activation_1 (Activation)	(None, 30, 30, 32)	0
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 15, 15, 64)	18496
activation_2 (Activation)	(None, 15, 15, 64)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	36928
activation_3 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 64)	0
dropout_2 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_2 (Dense)	(None, 512)	1180160

activation_4 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
activation_5 (Activation)	(None, 10)	0

Total params: 1,250,858

5.4.3 Hyperparameter Optimierung - kleiner Datensatz

Bei der Optimierung mit kleinen Datensätzen wird geprüft, ob sich andere Ergebnisse beim Optimieren der Hyperparameter mit kleinerem Datensatz zeigen. Dazu werden die gleichen Datensätze und Netze wie in den vorherigen Evaluierungen verwendet. Nur wird in diesem Fall, die Trainingsdatensätze um 90% der vorhanden Daten verkleinert. Der Anteil an Testdaten bleibt gleich, so werden die Endergebnisse auf dem gleichen Daten evaluiert.

5.4.4 Versuch der Modell-Architektur Optimierung

Abschließend wird ein Versuch zur Optimierung der Modell-Architektur eines klassifizierungs Netzes implementiert. Für diese Optimierung wird ein Fully-Connected-Network verwendet. Zur Umsetzung der Modell-Architektur Optimierung werden die Neuronen pro Schicht als Gene umgesetzt, zusehen in Abbildung 5.7. In diesem Versuch werden die Gene mit drei Schichten umgesetzt. Für die Neuronen pro Schicht, wird ein Minimum von 10 Neuronen und ein Maximum von 256 Neuronen, festgelegt. Mithilfe des Versuches soll eine optimale Modell-Architektur gefunden werden. Es werden die Hyperparameter des ersten Versuchs verwendet.

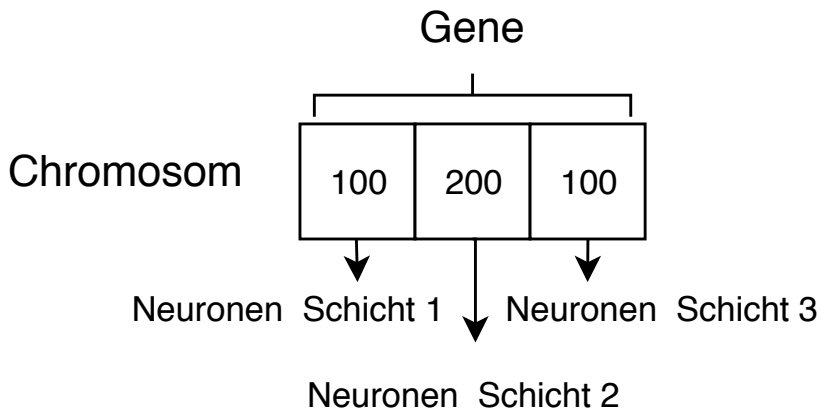

Abbildung 5.7: Beispielhafte Zeichnung des Chromosom eines Individuums in welchem die Neuronen als Gene realisiert wurden

5.4.5 Auswertung

Alle Ergebnisse werden automatisch mit den dazugehörigen Konfigurationen und Berechnungen in einer Json-Datei abgespeichert. Aus dieser Json-Datei wird dann automatisch eine Zusammenfassung aller Ergebnisse erstellt und anschließend in einer Excel Datei abgespeichert. Des Weiteren werden Dichte Diagramme aus den Json-Dateien zu den einzelnen Hyperparametern automatisch erstellt und gespeichert. In diesen werden die Dichteverteilungen der einzelnen Hyperparameter in Bezug auf die Fitness dargestellt. Somit können die Hyperparameter intuitiv ausgewählt werden. Der Ablauf der Auswertung ist schematisch in Abbildung 5.8 abgebildet.

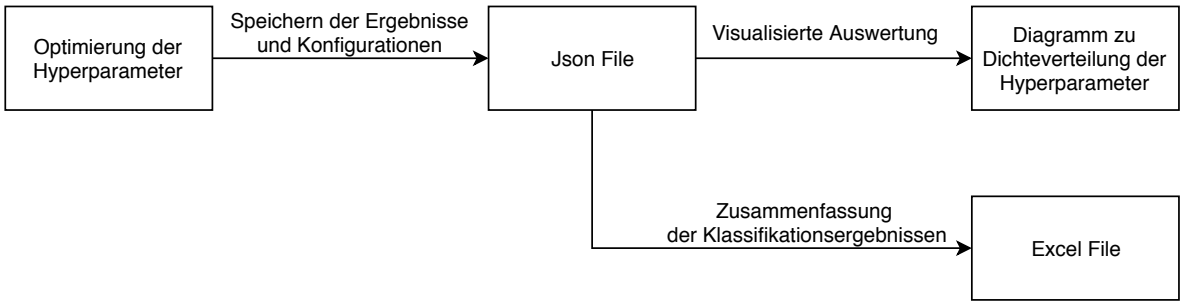

Abbildung 5.8: Datenstruktur der Auswertung

5.5 Zusammenfassung

In diesem Kapitel wurde auf die implementierten Optimierungsalgorithmen und künstlichen neuronalen Netzen eingegangen. Dazu wurde zuerst der Systemaufbau, Programmiersprache und alle dazu gehörigen Python-Pakete und ihre jeweilige Funktion erläutert. Nachfolgend wurde auf die Hardware, die zur Verfügung stand, eingegangen. Danach wurden die verwendeten Datensätze: Mnist Fashion, Mnist Digits und der Cifar10 Datensatz erklärt. Bei allen drei handelt es sich um Bilddatensätze, welche mit 28x28 Pixel und 32x32x3 Pixel, relativ kleine Bilddaten enthalten. Anschließend wird erläutert, welche 5 Schritte für den Genetischen Algorithmus implementiert wurden, sowie die Methoden, die zur Evaluierung des GA eingesetzt werden. Ebenso wird die implementierte Zufallssuche erläutert welche zum Vergleich des GA dient. Beide Algorithmen benutzen die gleichen Datensätze, als auch die gleichen künstlichen neuronalen Netze. Als KNNs werden ein Fully Conneted Netz mit zwei Variationen optimiert und ein Convolutional Neuronal Network verwendet. Zum Abspeichern der Netze werden Json und Excel Files benutzt. Aus diesen Files werden dann die Auswertungen bzw. das Dichte-Diagramm bestimmt.

Kapitel 6

Ergebnisse und Auswertung

In diesem Kapitel wird auf die Ergebnisse der Evaluation eingegangen. Zudem wird auf die Benchmarks der CPU und GPU Versionen eingegangen. Anschließend werden die Ergebnisse der Hyperparameter Optimierung von Fully-Connected-Networks und Convolution-Neuronal-Networks ausgewertet und besprochen. Abschließend wird auf automatischen Dichte-Diagramme eingegangen, dazu wird eine Optimierung beispielhaft ausgewählt und erklärt.

6.1 Benchmarks eines Trainingsvorganges auf der CPU und GPU

Zur Berechnung der Trainingsvorgänge wurden eine GPU und eine CPU optimierte Variante implementiert. Generell ist die Berechnung von künstlichen neuronalen Netzen mit der GPU wesentlich schneller als mit der CPU. Dies ist aber nicht immer der Fall, weshalb Tests durchgeführt wurden um sie nach ihrer Performance einzustufen. Um eine verbesserte Vergleichbarkeit zu erhalten wird kein Multiprocessing verwendet, da dieses auf der GPU nicht bzw. nur bedingt möglich ist. In der Tabelle 6.1 sind die erreichten Benchmarks abgebildet. Die Benchmarks wurden für das kleine Fully-Connected-Network(FCN) 5.1 und das Convolutional-Neural-Network(CNN) 5.3 erstellt. Für das FCN war die Berechnung mit der CPU deutlich schneller als die Berechnungen auf der GPU. Beim CNN zeigte sich das genaue Gegenteil. Die GPU schnitt, mit 270 Sekunden, deutlich besser gegenüber der CPU, mit 470 Sekunden, ab. Dies ist auf die Anzahl der trainierbaren Parameter zurückzuführen. Somit konnte nachgewiesen werden, dass das

Training auf der CPU für kleine Fully-Connected-Network effektiver ist, zudem kann auf der der CPU noch Multiprocessing verwendet werden. Dies kann das Training zusätzlich beschleunigen. Im Gegensatz dazu stehen die größeren Netze, wie das Convolutional-Neuronal-Network, dieses trainiert deutlich schneller auf der GPU. Dies ist auf die Größere der Daten und trainierbaren Parametern zurück zuführen. Da die Trainingsdaten des CNNs dreimal so groß 5.2 sind. Zudem besitzt das CNN 10 mal so viele trainierbaren Paramter wie das FCN 5.4. Aus diesem Test hat sich ergeben, dass die kleinen Netze auf der CPU mit Multiprocessing und die großen Netze auf der GPU trainiert werden.

Tabelle 6.1: Benchmarks für den Trainingsvorgang der Netze

Time to Train 10 Epochs	GPU	CPU
FCN	160 sec	27 sec
CNN	260 sec	470 sec

6.2 Ergebnisse der Evaluation

Für jede durchgeführte Optimierung wird eine Tabelle erstellt, in welcher die Evaluationsergebnisse festgehalten werden. Diese Ergebnisse beinhalten die folgenden Bewertungsmetriken: Klassifizierungsgenauigkeit, Precision, Recall und F1-Score und werden folgenderweise definiert:

- **Klassifizierungsgenauigkeit** (eng. accuracy) wird wie in Gleichung 6.1 definiert.

$$Accuracy = \frac{True\ Positive + True\ Negativ}{True\ Positive + True\ Negative + False\ Positive + False\ Negative} \tag{6.1}$$

- **Precision** wird wie in Gleichung 6.2 definiert.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{6.2}$$

- **Recall** wird wie in Gleichung 6.3 definiert.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{6.3}$$

- **F1 Score** wird wie in Gleichung 6.4 definiert.

$$F1\text{-}Score = 2\frac{Recall \cdot Precision}{Recall + Precision} \tag{6.4}$$

Diese Bewertungsmetriken werden für das beste künstliche neuronale Netz jedes Optimierungsalgortihmus berechnet. Anhand diesen Metriken werden die Algorithmen anschließend verglichen. Diese vier Metriken werden später in der gleichen Formation, wie in Tabelle 6.2, dargestellt. Die Ergebnisse werden in vier Abschnitte gegliedert: Ergebnis der Hyperparameter Optimierung eines Fully-Connected-Network, Ergebnis der Hyperparamet Optimierung eines Convolutional Neural Network, Ergebnis der Hyperparamter Optimierung mit verkleinertem Datensatz und Ergebnis des Versuchs zur Modell-Architektur Optimierung.

Tabelle 6.2: Aufbau der Evaluationsergebnisse

Größe des KNN	Größe des trainierten Netzes	
Algorithmus	accuarcy recall score	precision score f1 score

6.2.1 Ergebnis der Hyperparameter Optimierung eines Fully-Connected-Network

Die Ergebnisse der Optimierung der Hyperparameter eines Fully-Connected Netzes für 50 Iterationen ist in Tabelle 6.3 zusehen und für 250 Iterationen, in Tabelle 6.4. In diesen Tabellen werden die Metriken für zwei unterschiedlich große Fully-Connected Netze berechnet. Beide Netze wurde auf den Mnist Digits Datensatz trainiert und getestet. Darüber hinaus werden die Hyperparameter, wie in Kapitel 5.4, beschreiben für 50 und 250 Iterationen optimiert. Die Ergebnisse der optimierten Netze sind sich sehr ähnlich. Zudem ist der Nachkommaberreich vernachlässigbar, somit bringt der Genetische Algorithmus keine effektive Verbesserung der Ergebnisse gegenüber der Zufallssuche. Auch bei höheren Iterationen ist keine Verbesserung der Ergebnisse mithilfe des GA zuerkennen. Die Größe der KNN macht keine Unterschiede auf das Klassifizierungsergebnis. Darüber hinaus sind keine Unterschiede in den Bewertungsmetriken zu erkennen.

Tabelle 6.3: Ergebnisse der Algorithmen auf dem Mnist Digits Datensatz für 50 Iterationen

Größe des KNN	klein		groß	
Genetischer Algorithmus	98,13%	98,20%	97,94%	97,86%
	98,19%	98,19%	97,84%	97,85%
Zufallssuche	97,87%	97,82%	97,99%	98,23%
	97,80%	97,81%	98,22%	98,23%

Tabelle 6.4: Ergebnisse der Algorithmen auf dem Mnist Digits Datensatz für 250 Iterationen

Größe des KNN	klein		groß	
Genetischer Algorithmus	98,04%	97,98%	98,08%	98,00%
	97,98%	97,98%	97,94%	97,96%
Zufallssuche	98,03%	98,24%	98,18%	98,14%
	98,23%	98,23%	98,11%	98,11%

6.2.2 Ergebnis der Hyperparameter Optimierung eines Convolutional Neural Network

Die Ergebnisse der Optimierung der Hyperparameter eines Covolutional Neural Network sind in Tabelle 6.5 zusehen. Es haben sich ähnliche Ergebnisse für das CNN wie für das FCN ergeben. Es konnten hier kleine Verbesserungen im Nachkommabereich erbracht werden. Diese geringfügigen Unterschiede können aber vernachlässigt werden, da sie sich im Durchschnitt unter 1% befinden. Somit zeigte sich bei dieser Optimierung keine Verbesserung der Klassifikationsergebnisse durch den Genetischen Algorithmus, im Vergleich zur Zufallssuche.

Tabelle 6.5: Ergebnisse der Algorithmen auf dem Cifa10 Datensatz

Iterations	50		250	
Genetischer Algorithmus	78,74%	78,12%	79,34%	78,49%
	78,18%	77,95%	78,66%	78,49%
Zufallssuche	79,51%	79,36%	82,87%	81,58%
	79,09%	78,88%	82,87%	81,54%

6.2.3 Ergebnis der Hyperparameter Optimierung mit verkleinertem Datensatz

Die Evaluationen der Optimierung mit verkleinertem Datensatz wird für das Fully-Connected Netz und das Convolutional Neuronal Network durchgeführt. Diese sind in Tabelle 6.6 und 6.8 zusehen. Die Optimierungen entsprechen exakt den gleichen Abläufen der zuvor durchgeführten Optimierungen nur mit dem Unterschied, dass für diese Optimierung der Datensatz um 90% verkleinert wurde.

Bei der Optimierung der Hyperparameter des Fully-Connected Netzes (6.6) gibt es Unterschiede bei den Ergebnissen des GA im Vergleich zur Zufallssuche. Dies ist vor allem in der Berechnung von 250 Iterationen bei der kleinen Modell Architektur zusehen. Hier ist die Klassifikationsgenauigkeit besser als ein Prozent. Des Weiteren konnte im Durchschnitt ein halbes Prozent bessere Ergebnisse mit dem Genetischen Algorithmus im Vergleich zu der Zufallssuche erreicht werden. Dennoch sind die Verbesserungen nur sehr gering.

Tabelle 6.6: Ergebnisse der Algorithmen auf dem verkleinerten Mnist Digits Datensatz für 50 Iterationen

Größe des KNN	klein		groß	
Genetischer Algorithmus	95,58%	94,47%	95,92%	94,92%
	94,46%	94,45%	94,86%	94,88%
Zufallssuche	94,42%	94,44%	94,92%	95,33%
	94,28%	94,32%	94,81%	94,82%

Tabelle 6.7: Ergebnisse der Algorithmen auf dem verkleinerten Mnist Digits Datensatz für 250 Iterationen

Größe des KNN	klein		groß	
Genetischer Algorithmus	95,67%	94,82%	95,83%	94,65%
	94,81%	94,80%	94,59%	94,61%
Zufallssuche	94,85%	94,41%	95,25%	94,43%
	94,22%	94,24%	94,42%	94,38%

Bei der Optimierung der Hyperparameter des Convolutional Neural Networks (6.8) gibt es Unterschiede bei den Ergebnissen des GA im Vergleich zur Zufallssuche. Bei der Optimierung mit Hilfe des GA konnte keine Verbesserung erbracht werden. Die Ergebnisse der Klassifikation sind sogar bis zu 4% schlechter.

Tabelle 6.8: Ergebnisse der Algorithmen auf dem verkleinerten Cifar10 Datensatz

Iterations	50		250	
GA	58,7%	56,01%	57,8%	57,39%
	56,28%	55,97%	54,78%	56,02%
ZS	61,6%	59,58%	60,7%	62,59%
	59,65%	59,17%	60,73%	60,87%

6.2.4 Ergebnis des Versuchs zur Modell-Architektur Optimierung

Bei der Modell-Architektur Optimierung zeigte sich nach einigen Versuchen bei beiden Algorithmen eine Tendenz zu einer höheren Neuronenanzahl. Denn je mehr Neuronen desto mehr Informationen können vom KNN gelernt werden und führen dementsprechend zu besseren Ergebnissen. Somit wurde die Anzahl an Neuronen pro Schicht meist Richtung Maximum (maximale Anzahl Neuronen pro Schicht) optimiert. Um dies zu verhindern müsste eine spezielle Fitnessfunktion geschrieben werden, in welcher die gesamte Anzahl von Neuronen als negativen Einfluss mit eingebracht wird. Da die Optimierung der Modell-Architektur nicht zum eigentlichen Teil der Arbeit gehört, wurde nach einigen Test die Versuche zur Modell-Architektur Optimierung eingestellt. Ideen zur Weiterentwicklung der Modell-Architektur Optimierung werden im Ausblick 7.2 angeführt.

6.3 Ergebnis Visualisierung - Dichtediagramm

Das Dichte-Diagramm dient als zusätzliche Auswertung des Genetischen Algorithmus. Es kann nur auf die Ergebnisse des Genetischen Algorithmus angewendet werden, da dieser genügend Individuen liefert die einen hohen Fitnesswert besitzen. Um eine Aussage zu den einzelnen Hyperparameter zu treffen, werden die Hyperparameter der letzte Generation in einem Diagramm dargestellt. Im Dichte-Diagramm werden die Hyperparameter einzelner Individuen über die Fitness aufgetragen. Hierfür werden schlechte Individuen, die einen Fitnesswert schlechter als 20% des besten Individuums haben, heraus gefiltert. Somit werden nur Individuen mit hoher Qualität dargestellt. Durch das Auftragen der Hyperparameter, bezogen auf ihre Fitness, wird durch die visuelle Auswertung klar, welche Werte sich für einen speziellen Hyperparameter eignen. Dieses Dichte-Diagramm hat eine begrenzte Aussagekraft, da es sich um einen mehrdimensionalen Suchraum handelt, in dem sich alle Hyperparameter gegeneinander beeinflussen. Dies kann in den Dichte-Diagrammen nicht berücksichtigt werden. Trotzdem wird durch das Dichte-Diagramm ein Rahmen für jeden Hyperparameter sichtbar. Im nachfolgenden Kapitel werden die Diagramm für jeden optimierten Hyperparameter aufgezeigt und besprochen. Diese Diagramme werden aus den Ergebnissen der Hyperparameter Optimierung mit dem Fully-Connected Netz mit 250 Iterationen mit ganzem Datensatz erstellt. Es werden Diagramme zu allen Optimierungsvorgängen erstellt. Diese werden aber aus Platzgründen hier nicht aufgezeigt und sind im Anhang zu finden.

6.3.1 Diagramm Lernrate

In Abbildung 6.1 ist die Lernrate in Verbindung mit der Klassifizierungsgenauigkeit aufgetragen. Es zeigte sich eine Lernrate im Bereich von 0,005 bis 0,1 als zielführend. Zudem ist die Lernrate im Bereich von 0,005 bis 0,05 bei den Individuuen der letzen Generation häufig vertreten, wodurch dies eine hohe Garantie für eine hochwertiges Individuum widerspiegelt. Da es aber einige Individuen eine Lernraten von 0.1 besitzen, die eine hohe Fitness aufweisen ist es nicht möglich einen besten Wert für die Lernrate festzulegen.

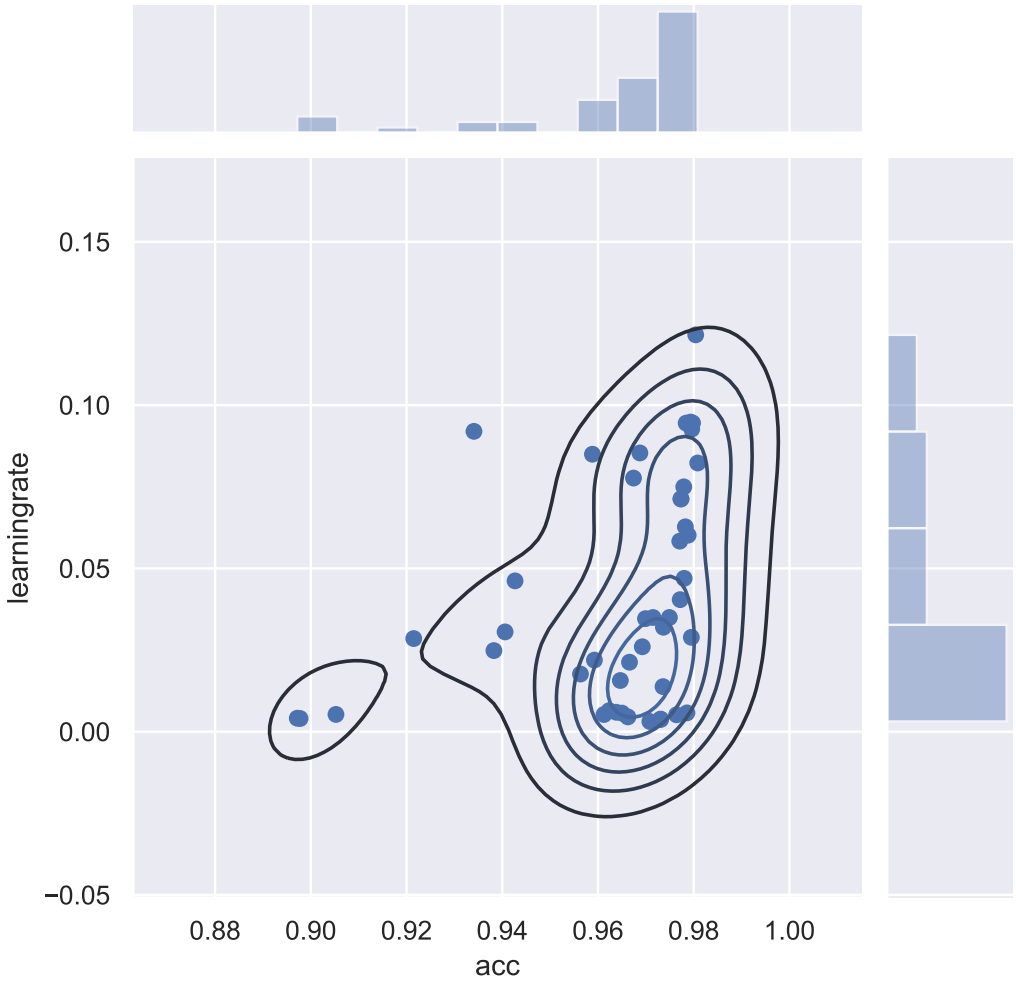

Abbildung 6.1: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

6.3.2 Diagramm Dropout

In Abbildung 6.2 ist der Dropout in Verbindung mit der Klassifizierungsgenauigkeit aufgetragen. Es zeigte sich ein Dropout im Bereich von 0,1 bis 0,4 als zielführend. Zudem ist der Dropout im Bereich von 0,2 in den Individuen der letzten Generation häufig vertreten, wodurch dies eine hohe Garantie für eine hochwertiges Individuum widerspiegelt. Im Diagramm ist auch gut zusehen, dass eine Dropoutrate über 0,5 sich kontraproduktiv für die Klassifzierungsgenauigkeit auswirkt. Dies ist über die geringe Klassifzierungsgenauigkeit(acc) für den Dropout über 0,5 sichtbar.

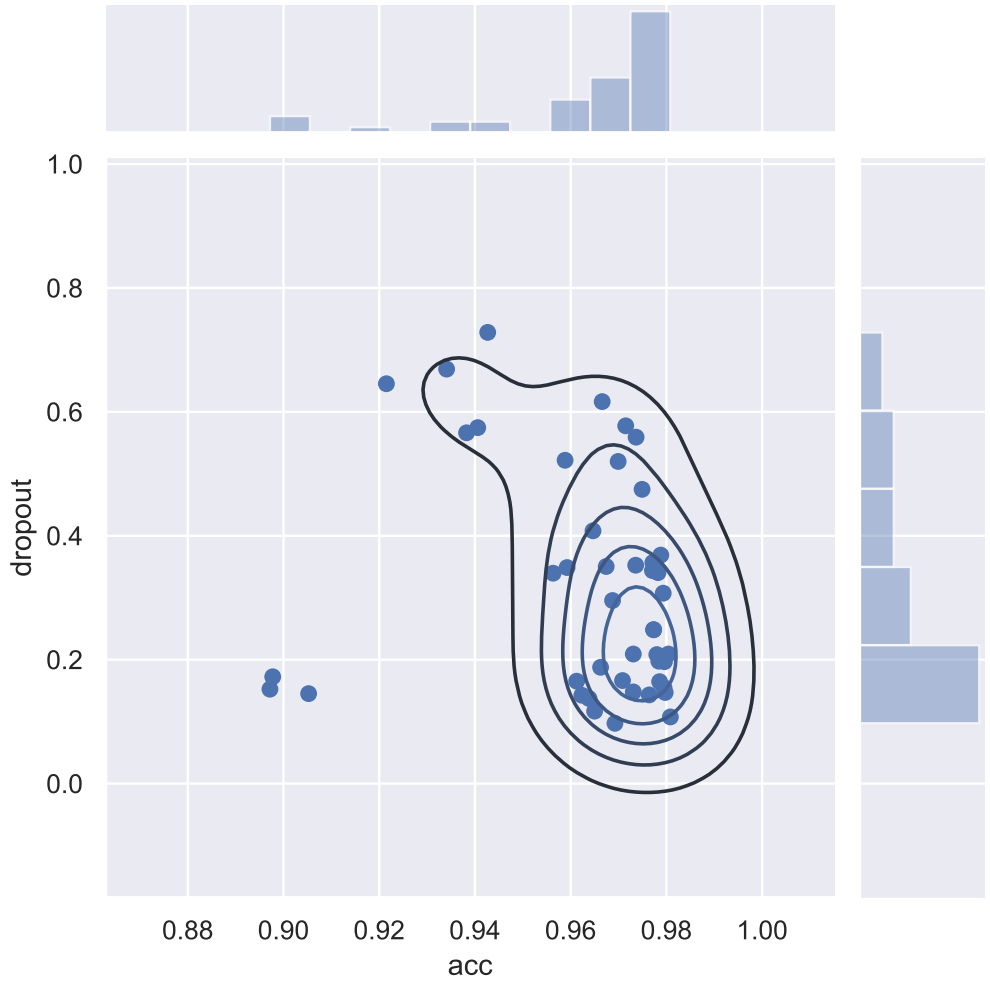

Abbildung 6.2: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

6.3.3 Diagramm Batchsize

In Abbildung 6.3 ist die Batchsize in Verbindung mit der Klassifizierungsgenauigkeit aufgetragen. Es zeigte sich eine Batchsize im Bereich von 30 bis 50 als nützlich. Zudem ist die Batchsize im Bereich von 40 in den Individuen der letzten Generation häufig vertreten, wodurch dies eine hohe Garantie für ein hochwertiges Individuum widerspiegelt.

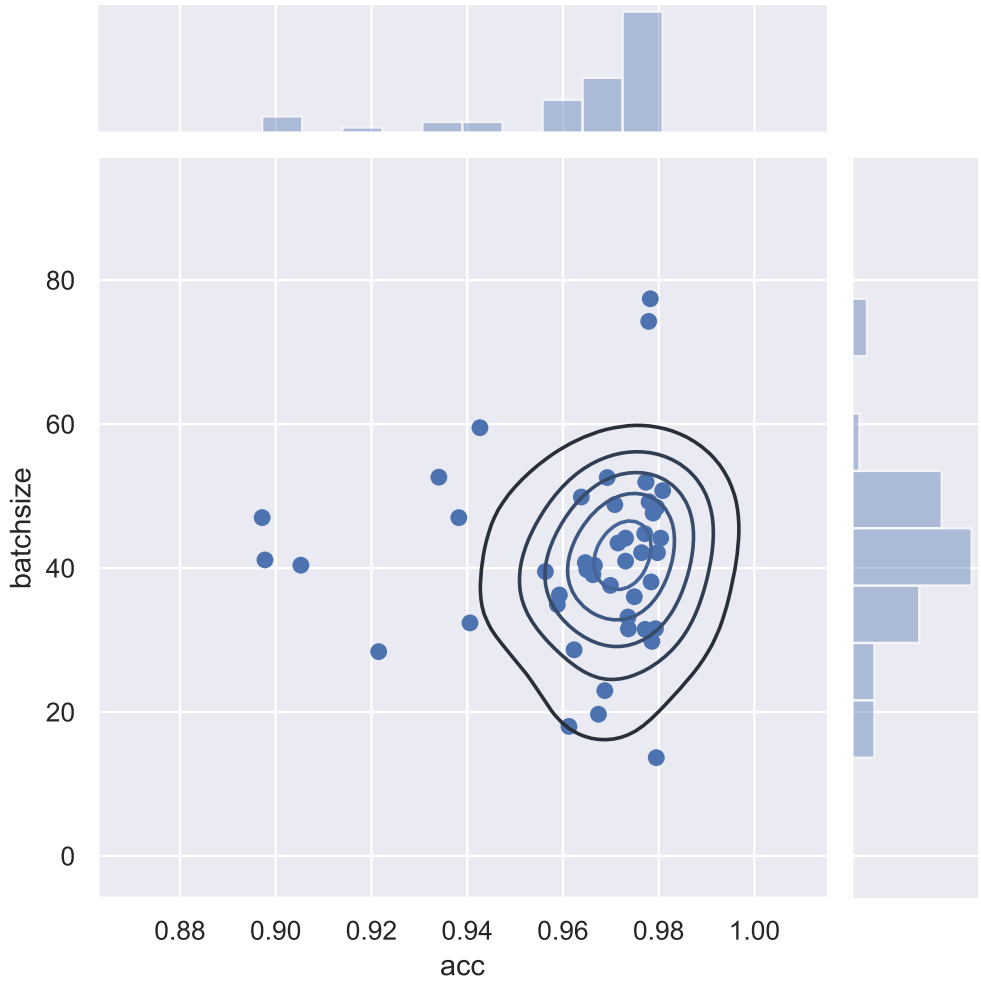

Abbildung 6.3: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

6.3.4 Diagramm Epochen

In Abbildung 6.4 ist die Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit aufgetragen. Es zeigte sich, dass eine Epochenanzahl im Bereich von 20 bis 80 als zielführend. Zudem sind die Epochen im Bereich von 40 in den Individuen der letzten Generation häufig vertreten, wodurch dies eine hohe Garantie für eine hochwertiges Individuum widerspiegelt. In diesem Diagramm ist gut zusehen das ein Individuum mit höherer Epochenanzahl meist gute Fitnesswerte liefert, wobei eine geringere Epochenanzahl ab einem bestimmten Wert, hier 20, eine geringere Fitness aufweist.

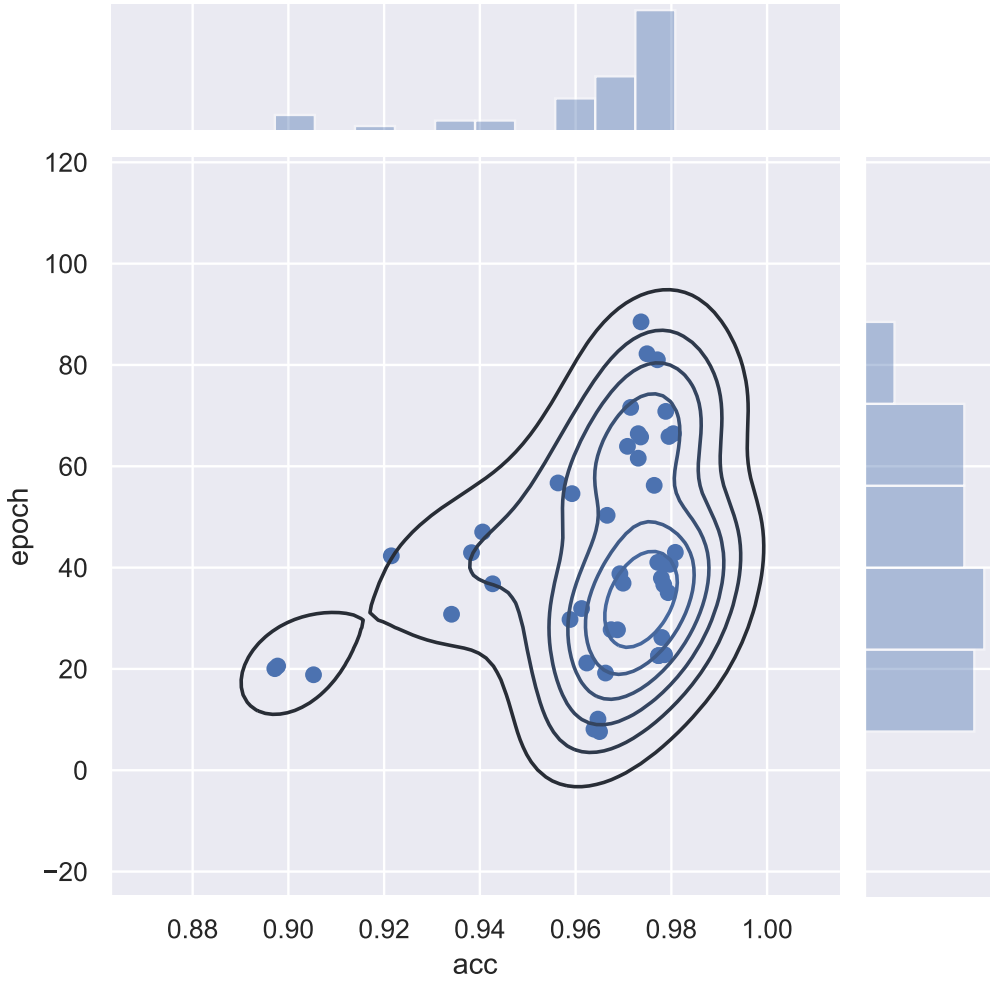

Abbildung 6.4: Dichte-Diagramm die Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

6.3.5 Diagramm Optimierer

In Abbildung 6.5 sind die Optimierer in Verbindung mit der Klassifizierungsgenauigkeit aufgetragen. Es zeigte sich, dass der Optimierer 1 = SGD und 3 = Adagrad am häufigsten mit gutem Fintesswert auftreten. Es ist gut zusehen, das einzelne Optimierer wie 2 = RMSporp oder 7 = ftrl gar nicht auftreten, dementsprechend schneiden diese Optimierer schlecht ab. Somit erhalten wir durch das Dichte-Diagramm, dass SGD und Adagrad die favorisierten Optimierer sind.

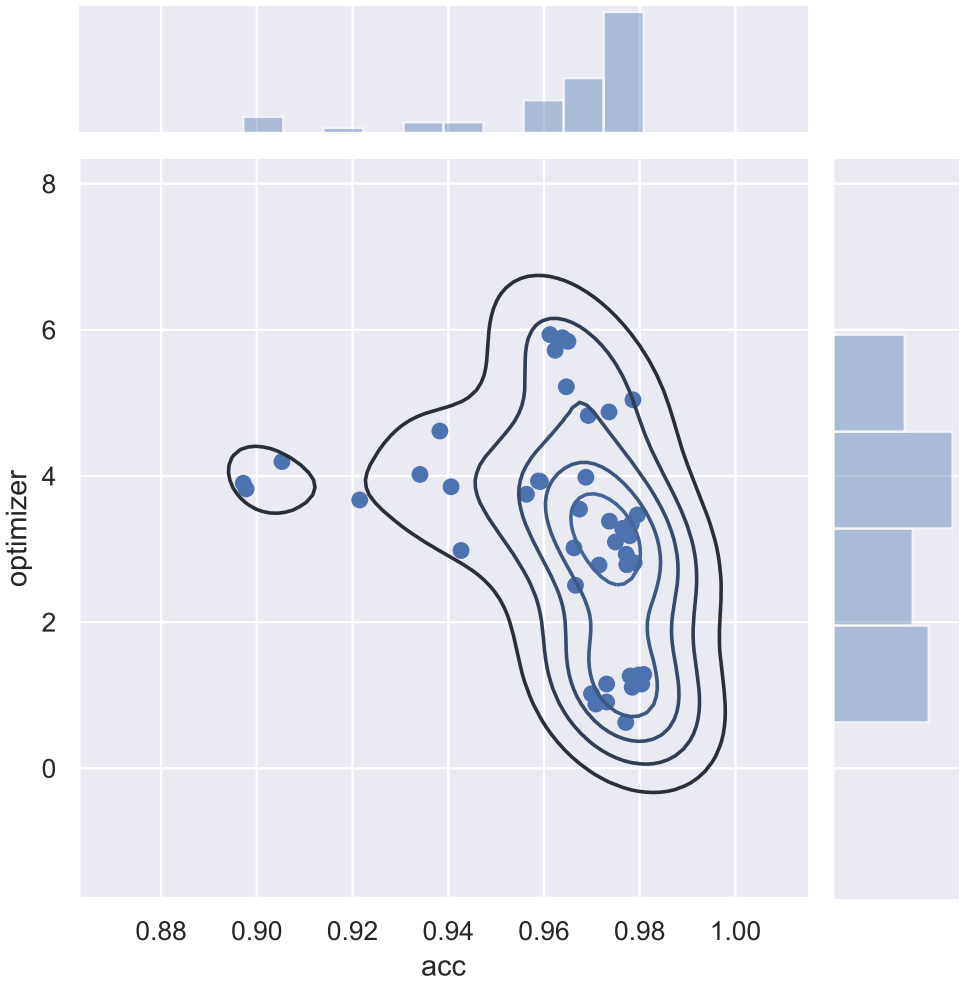

Abbildung 6.5: Dichte-Diagramm der Optimierer in Verbindung mit der Klassifizierungsgenauigkeit(acc)

Jede ganze Zahl des Optimizer steht für einen eigenen Optimierer. Es gilt: 0=adam, 1=SGD, 2=RMSprop, 3=Adagrad, 4=adadelta, 5=adammax, 6=nadam, 7=ftrl.

6.4 Diskussion der Ergebnisse

Bei den Optimierungen der Hyperparameter haben sich nicht die erwarteten Ergebnisse gezeigt. Bei der Optimierung der Hyperparameter der Fully Connected Netze zeigten sich nur sehr geringe Unterschiede in den Ergebnissen. Diese Unterschiede liegen im Nachkommabereich und können vernachlässigt werden. Somit konnte bei der Optimierung des Fully Connected keiner der Algorithmen sich als besser herausstellen. Bei der Hyperparameter Optimierung ergaben sich ähnliche Ergebnisse. Bei 250 Iterationen war die Zufallssuche sogar bis zu 4% besser als der GA. Bei der Hyperparameteroptimierung mit kleinem Datensatz waren eine Verbesserung von bis zu 1% beim Fully Conntected Netz mit Hilfe des GA möglich, dennoch ist diese Verbesserung irrelevant klein. Somit konnten in allen drei Optimierungsversuchen keine relevante Verbesserung der Klassifikationsgenauigkeit mit Hilfe des Genetischen Algorithmus gegenüber der Zufallssuche gezeigt werden. Dabei äußerte sich eine Änderung der Berechnungsdauer sowie der Modell-Architektur nicht in einer Änderung der Endergebnisse. Die geringe Verbesserung liegt nicht daran, dass der GA ungeeignet ist. Denn der GA konnte im Vergleich zur Zufallssuche häufiger Individuen finden, welche einen guten Fitnesswert besitzen. Dennoch gab es kein Individuum das verbesserte Ergebnisse lieferte. Somit konnte die Zufallssuche mit ihren wenigen gefundenen Individuen die gleiche Ergebnisse erreichen wie der GA. Dies kann durch einen geringen Einfluss der Hyperparameter auf das Endergebnis begründet werden. Kleine Veränderungen der Hyperparameter haben keine messbaren Änderungen der Endergebnisse ergeben. Solange die Hyperparameter in einem bestimmten Rahmen liegen, werden gute Klassifizierungsergebnisse erzielt. Dieser Rahmen für die Hyperparameter kann mit Hilfe des GA sichtbar gemacht werden. Durch die Population im GA kann gesagt werden in welchem Rahmen sich die Hyperparameter befinden müssen um gute Ergebnisse zu liefern. Diese Information konnte aus der Zufallssuche nicht entnommen werden, da die meisten Individuen der Zufallssuche keinen guten Fitnesswert besitzen. Dieser Rahmen zeigt sich in der Auswertung über die Visualisierung 6.3, somit erreicht der Genetische Algorithmus gegenüber der Zufallssuche keine großen Verbesserungen. Doch durch seine Zusatzinformationen kann ein Mehrwert mit gleicher Berechnungszeit gewonnen werden.

6.5 Zusammenfassung

In diesem Kapitel wurde auf die Ergebnisse und deren Auswertung eingegangen. Dazu wurden zuerst die Benchmarks des Trainingsvorgangs erklärt. Es zeigte sich, dass große KNNs deutlich schneller auf der GPU trainieren. Hingegen trainieren kleine KNNs auf der CPU deutlich schneller. Anschließend werden die Ergebnisse des Evaluierungsteils besprochen. Hier zeigten sich nicht die erwarteten Ergebnisse. Der Genetische Algorithmus konnte nur in wenigen Fällen eine Verbesserung der Klassifkationsergebnisse, über das Optimieren der Hyperparameter, erreichen. Zudem war die Verbesserungen vernachlässigbar klein. Denn die Zufallssuche konnte wenige Individuen mit guten Hyperparametern finden. Diese Individuen waren ausreichend gut, dass eine Optimierung mit Hilfe des GA keine Verbesserung gegenüber diesen Individuen erbrachte. Dennoch hat der Genetische Algorithmus einen Vorteil. Er kann über seine letzte Population eine zusätzliche Aussage treffen. Diese beinhaltet in welchen Bereichen sich die einzelnen Hyperparameter befinden, die gute Ergebnisse liefern. Diese Bereiche wurden mithilfe des Dichte-Diagramms sichtbar gemacht.

Kapitel 7

Fazit und Ausblick

7.1 Fazit

Mit Hinblick auf die zu Beginn der Arbeit definierten Ziele, kann ein abschließendes positives Fazit gezogen werden. Zur Umsetzung dieser Ziele wurde ein automatisierter Traininigs- und Auswerte- Prozess für künstliche neuronale Netze erstellt. Anschließend wurde ein Genetischer Algorithmus zur Optimierung von Hyperparametern künstlicher neuronaler Netze konzipiert. Dazu wurden Versuche durchgeführt, in welchen die Methoden des Genetischen Algorithmus verglichen wurden. Um diese Versuche durchzuführen, wurden alle State-of-the-Art Methoden implementiert. Die zielführendsten Methoden wurden anschließend in der Evaluierung verwendet. Des Weiteren wurde die Zufallssuche als Algorithmus zum Vergleichen der Ergebnisse ausgewählt und implementiert. Abschließend wurden diese beiden Algorithmen evaluiert. In den Ergebnissen zeigte der Genetische Algorithmus nicht die erhofften Verbesserungen in der Klassifikationsgenauigkeit gegenüber der Zufallssuche. Obwohl der Genetische Algorithmus wesentlich mehr Individuen findet die eine hohe Qualität aufweisen. Bei den Individuen des GA konnte sich jedoch kein Individuum deutlich von den anderen absetzen bzw. eine Verbesserung der Klassifikationsgenauigkeit erbringen. Dies liegt daran, dass kleine Veränderungen der Hyperparameter keine messbaren Unterschiede auf DIE Endergebnisse haben. Dennoch hat sich in der Evaluierung gezeigt, dass sich die Hyperparameter in einem bestimmten Intervall befinden müssen, sonst liefert das künstliche neuronale Netz keine brauchbaren Ergebnisse. Dieser bestimme Bereich kann über das Dichte-Diagramm des Genetischen Algorithmus sichtbar gemacht werden. Dieser Bereich ist für jedes Netz bzw. Modell-Architektur individuell und kann sogar von den Trainingsdaten abhängen. Der Geneti-

sche Algorithmus, ist in Bezug auf das Finden der besten Hyperparameter, vergleichbar mit der Zufallssuche. Zusätzlich hat der GA den Vorteil, dass er Zusatzinformationen besitzt. Somit kann er ein bestimmtes Intervall angeben, in welchem sich die Hyperparameter befinden müssen, um gute Ergebnisse zu erreichen. Zusammengefasst kann gesagt werden, dass bei einem kleinen Suchraum oder wenn nur wenig Rechenressourcen zur Verfügung stehen, eine Zufallssuche ausreichend ist. Doch wenn der Suchraum für die Hyperparameter groß ist oder ein leistungsstarker Rechner zur Verfügung steht, hat der Genetische Algorithmus durchaus seine Vorteile.

7.2 Ausblick

Die Optimierung von Hyperparametern mit Hilfe eines Genetischen Algorithmus hat nur bedingt Verbesserungen erbracht. Weswegen eine Optimierung der Hyperparameter mit Hilfe eines anderen Optimierungsalgorithmus denkbar wäre. Hierfür ist eine Optimierung mit dem Bayesian Optimierer oder dem HyperOPt (Distributed Asynchronous Hyperparameter Optimization) denkbar.

Neben der Optimierung von Hyperparametern könnte in einer weiteren Arbeit auf die Optimierung der Modell-Architektur eingegangen werden. Zu diesem Thema wurden in dieser Arbeit bereits erste Versuche durchgeführt. Die Ergebnisse des Versuches 6.2.4 zeigten bei der Optimierung eine Tendenz zu größeren Modell-Architekturen. Um dieser Tendenz entgegenzuwirken müsste eine geeignete Fitnessfunktion entwickelt werden. Hierfür könnte beispielsweise die Gesamtanzahl an Parametern als negativer Faktor mit einbezogen werden. Generell ist ein Finden spezieller Netze über das Erstellen einer speziellen Fitnessfunktion möglich. Eine spezielle Fintessfunktion ist nicht nur für die trainierbaren Parameter denkbar. Sie könnte auch zur Findung von Netzen mit hoher Interferenz genutzt werden.

Zudem könnten noch andere Parameter als Gene umgesetzt werden. In dieser Arbeit ging es nur um die gängigen Hyperparameter 2.1.5. Diese Hyperparameter haben aber nur einen geringen Einfluss auf das Klassifizierungsergebnis. Es wäre durchaus denkbar eine Optimierung mit anderen oder mehreren Parametern durchzuführen. Somit kann der Genetische Algortihmus seine Vorteile im großen Suchraum ausspielen.

Literaturverzeichnis

[ACT19] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. *CoRR*, abs/1902.01724, 2019.

[Bak85] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[BNR06] Nicola Beume, Boris Naujoks, and Günter Rudolph. Mehrkriterielle optimierung durch evolutionäre algorithmen mit s-metrik-selektion. *Universität Dortmund, Chair for Algorithm Engineering*, 2006.

[Cal08] Luisa Caldas. Generation of energy-efficient architecture solutions applying gene_arch: An evolution-based generative design system. *Advanced Engineering Informatics*, 22(1):59–70, 2008.

[Cho18] Francois Chollet. *Deep Learning mit Python und Keras - Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek.* MITP-Verlags GmbH & Co. KG, Heidelberg, 1. aufl. edition, 2018.

[ES+03] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[Fra17] Chollet Francois. Deep learning with python, 2017.

[Gad18] Ahmed Fawzy Gad. *Practical Computer Vision Applications Using Deep Learning with CNNs - With Detailed Examples in Python Using TensorFlow and Kivy.* Apress, New York, 1. aufl. edition, 2018.

[GLL11] Matthias Gerdts, Frank Lempio, and Frank Lempio. *Mathematische Optimierungsverfahren des Operations Research* -. Walter de Gruyter, Berlin, 1. aufl. edition, 2011.

[hC19] Yu hsin Chen. How evolutionary selection can train more capable self-driving cars, 2019.

[HGLL06] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. Automated antenna design with evolutionary algorithms. *American Institute of Aeronautics and Astronautics*, 2006.

[JDO+17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[KJ11] Sina Keshavarz and Reza Javidan. Software quality control based on genetic algorithm. *International Journal of Computer Theory and Engineering*, pages 579–584, 01 2011.

[Kra17] Oliver Kramer. *Genetic algorithm essentials*, volume 679. Springer, 2017.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SHM+16] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[sie19] siemens. Generative design, 2019.

[SKK12] Karolina Stanislawska, Krzysztof Krawiec, and Zbigniew W. Kundzewicz. Modeling global temperature changes with genetic programming. *Comput. Math. Appl.*, 64(12):3717–3728, December 2012.

[SKV15] Karolina Stanislawska, Krzysztof Krawiec, and Timo Vihma. Genetic programming for estimation of heat flux between the atmosphere and sea ice in polar regions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1279–1286, New York, NY, USA, 2015. ACM.

[SM02] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[SPM15] Anupriya Shukla, Hari Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. *2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, ABLAZE 2015*, 02 2015.

[SSF+16] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial Neural Networks - A Practical Course*. Springer, Berlin, Heidelberg, 1st ed. 2017 edition, 2016.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[US15] Dr. Anantkumar Umbarkar and P Sheth. Crossover operators in genetic algorithms: A review. *ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1)*, 6, 10 2015.

[Wei15] Karsten Weicker. *Evolutionäre Algorithmen*. Springer-Verlag, 2015.

Kapitel 8

Anhang

8.1 Dichte-Diagramme zu den durchgeführten Optimierungen mit dem gesamten Datensatz

8.1.1 50 Iterationen des Genetischen Algorithmus des kleinen Fully-Connected Netzes

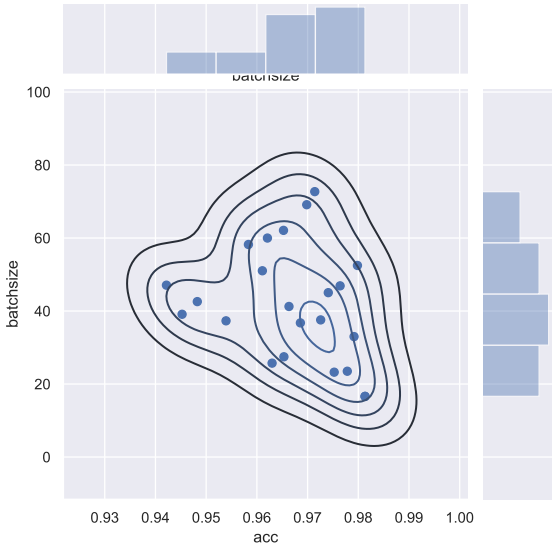

Abbildung 8.1: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

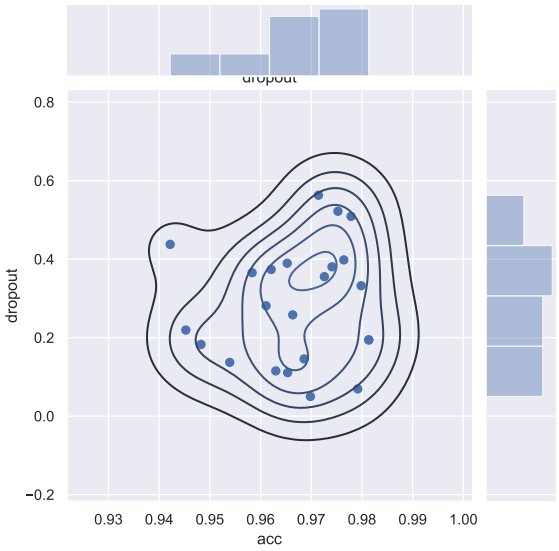

Abbildung 8.2: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

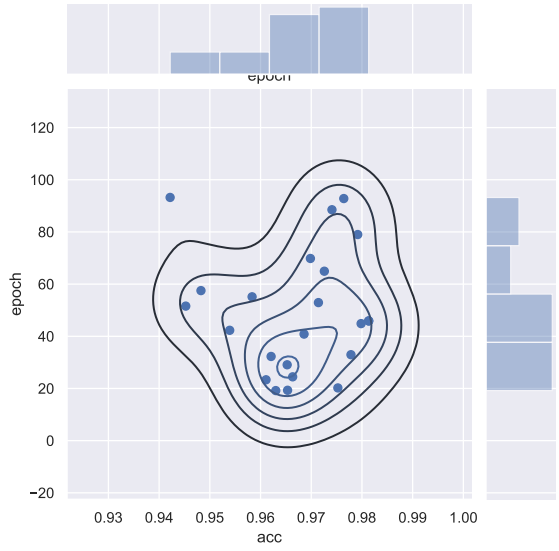

Abbildung 8.3: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

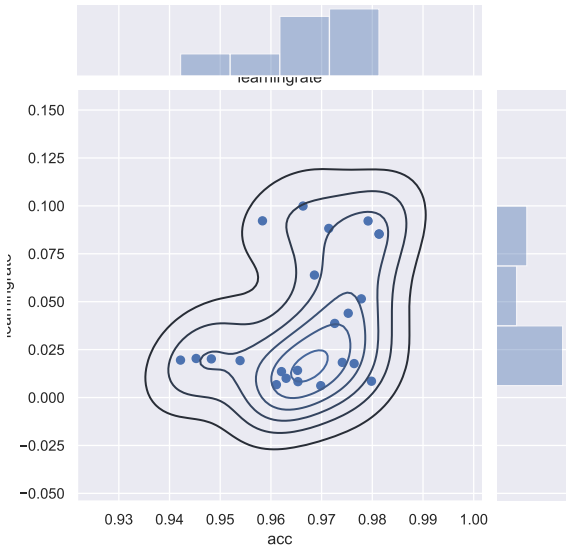

Abbildung 8.4: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

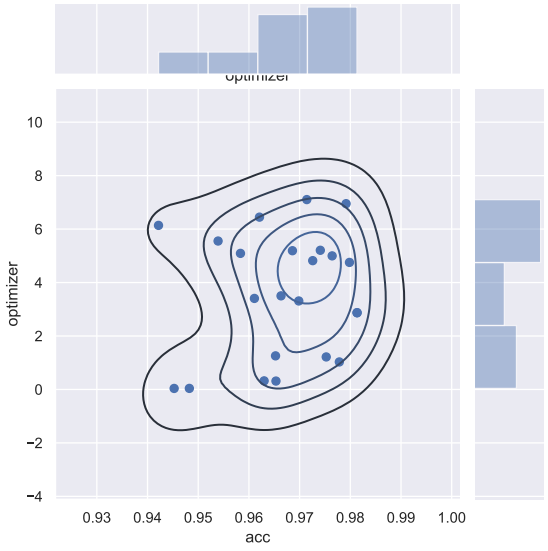

Abbildung 8.5: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.1.2 50 Iterationen des Genetischen Algorithmus des großen Fully-Connected Netzes

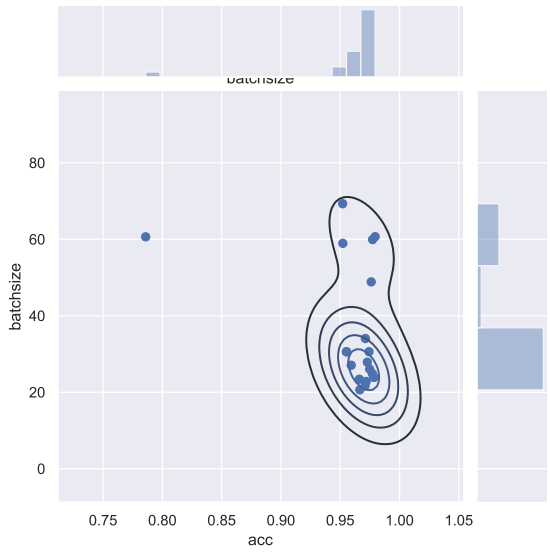

Abbildung 8.6: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

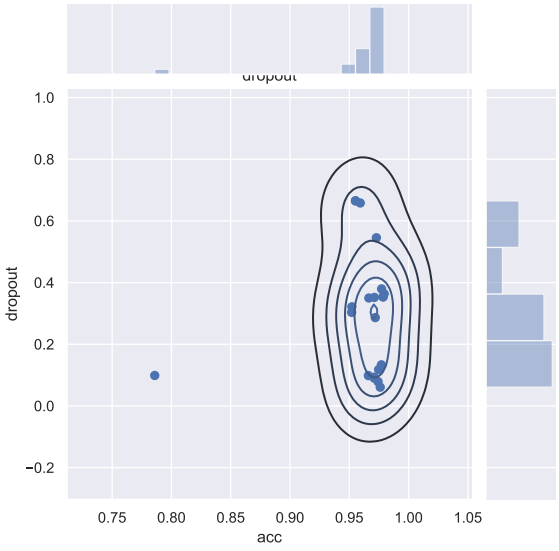

Abbildung 8.7: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

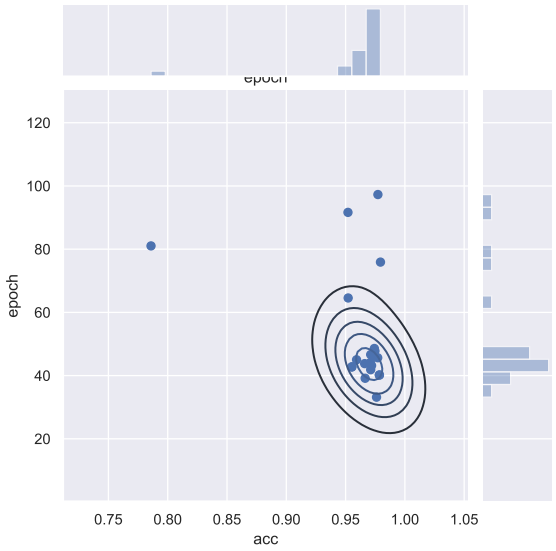

Abbildung 8.8: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

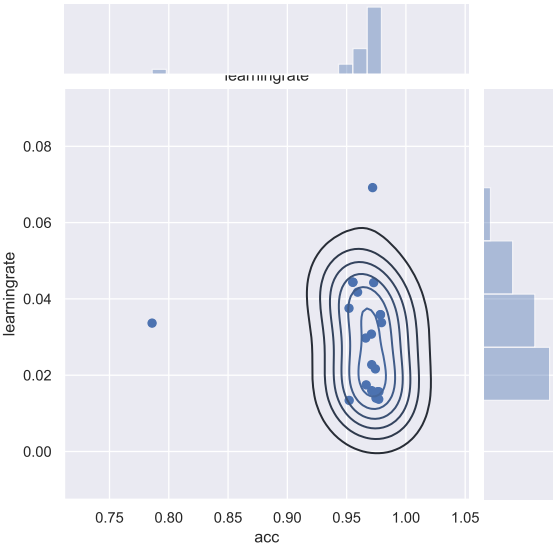

Abbildung 8.9: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

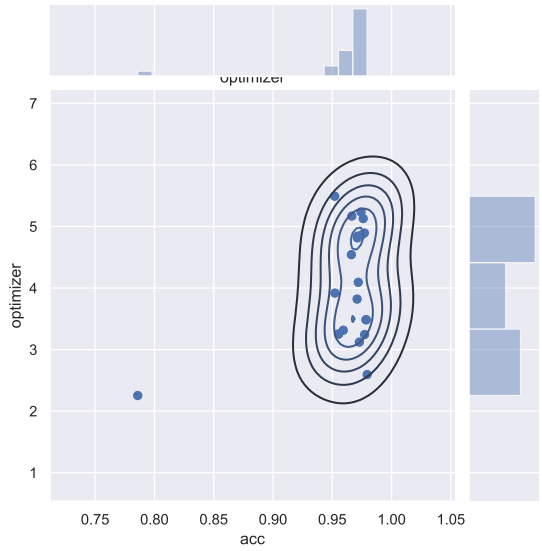

Abbildung 8.10: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.1.3 250 Iterationen des Genetischen Algorithmus des kleinen Fully-Connected Netzes

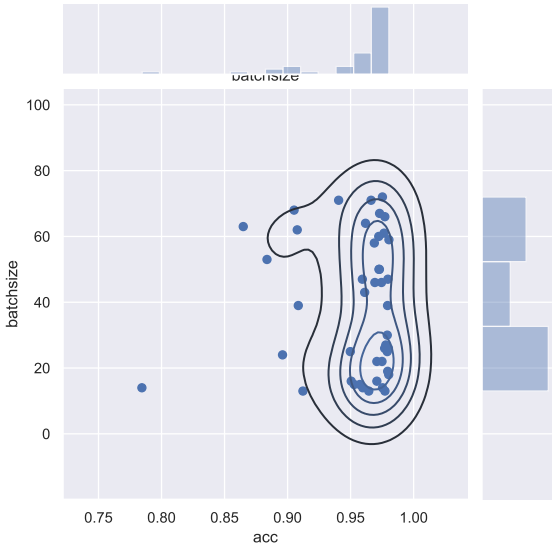

Abbildung 8.11: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

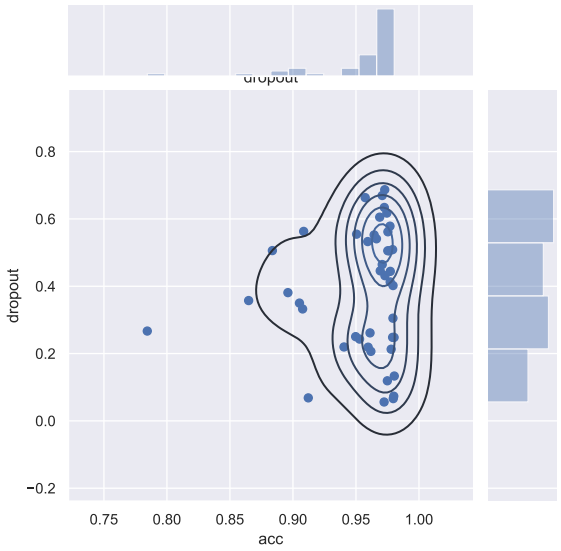

Abbildung 8.12: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

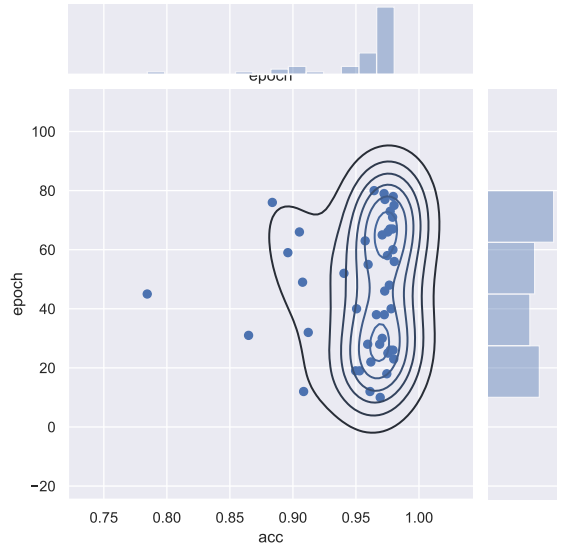

Abbildung 8.13: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

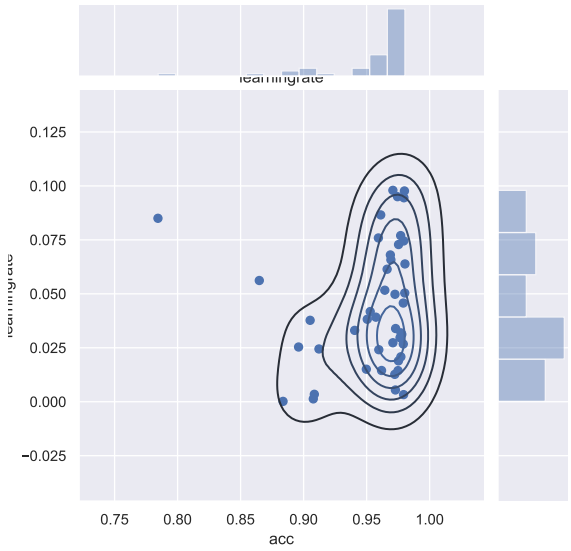

Abbildung 8.14: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

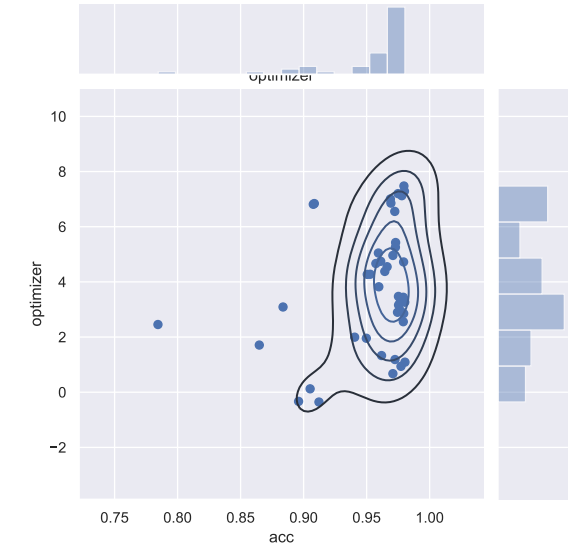

Abbildung 8.15: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.1.4 250 Iterationen des Genetischen Algorithmus des großen Fully-Connected Netzes

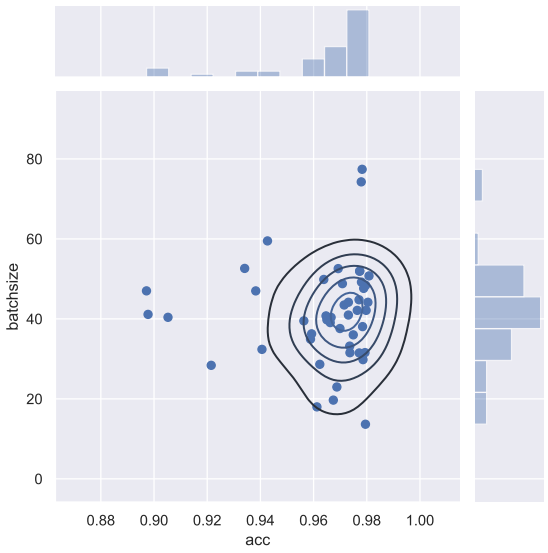

Abbildung 8.16: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

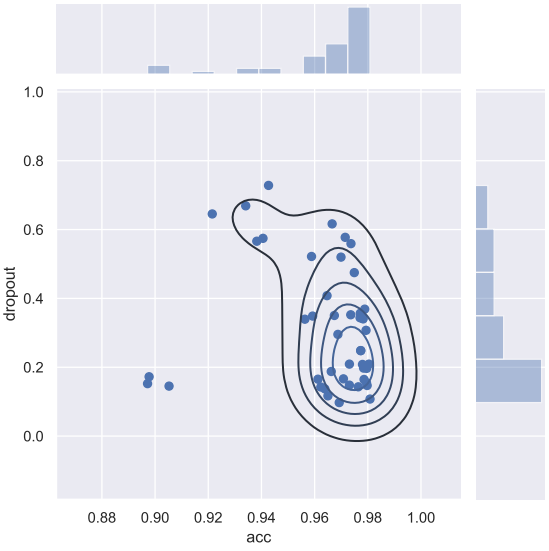

Abbildung 8.17: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

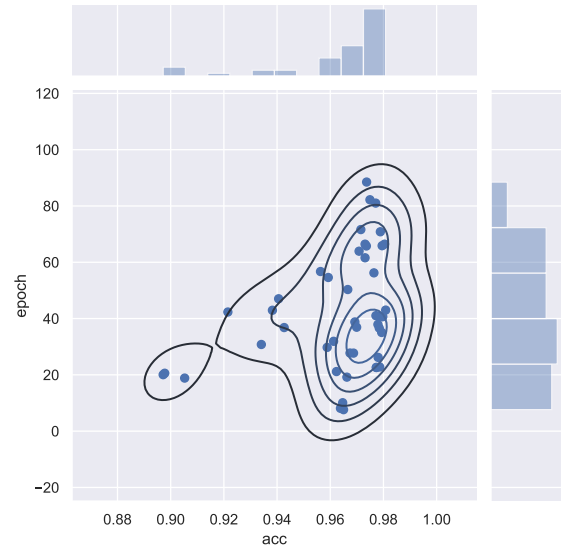

Abbildung 8.18: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

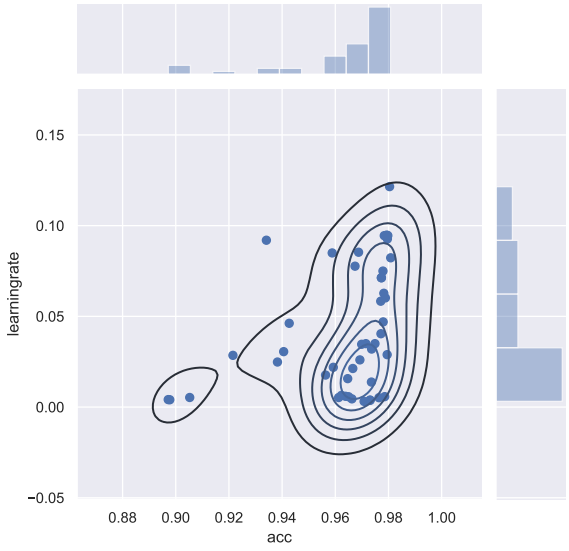

Abbildung 8.19: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

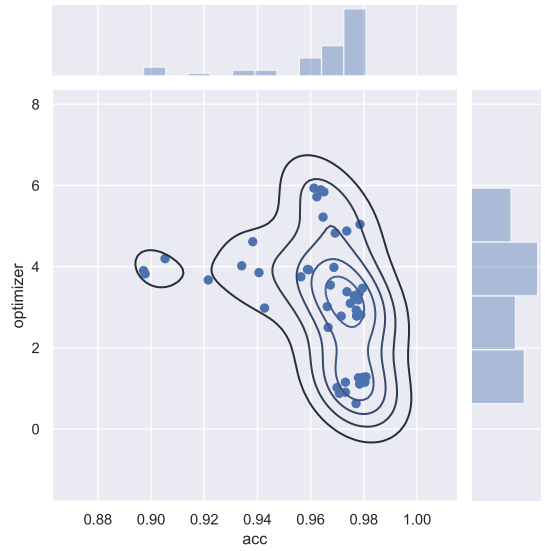

Abbildung 8.20: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.1.5 50 Iterationen des Genetischen Algorithmus des Convolutional Neural Network

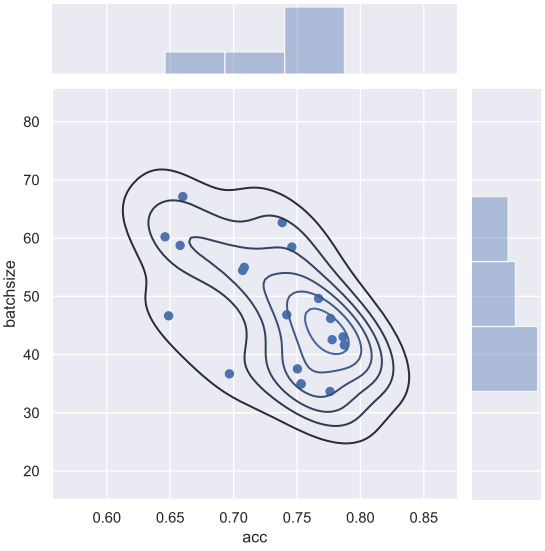

Abbildung 8.21: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

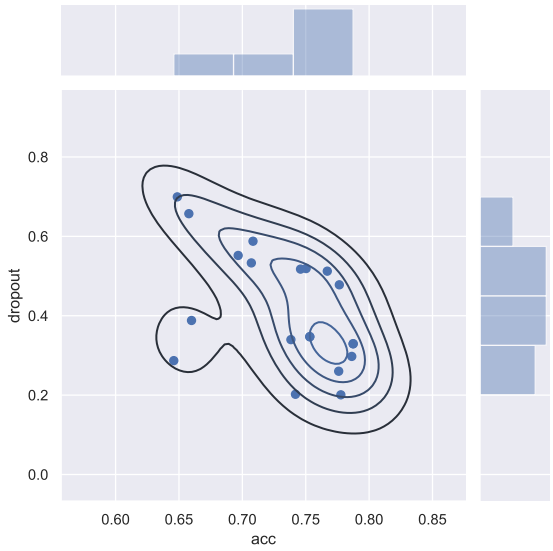

Abbildung 8.22: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

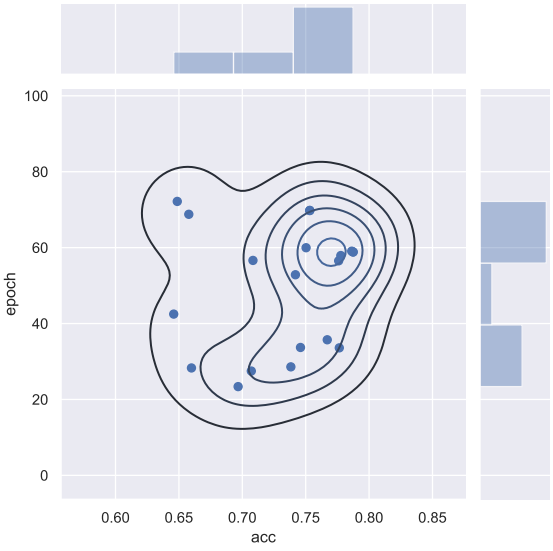

Abbildung 8.23: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

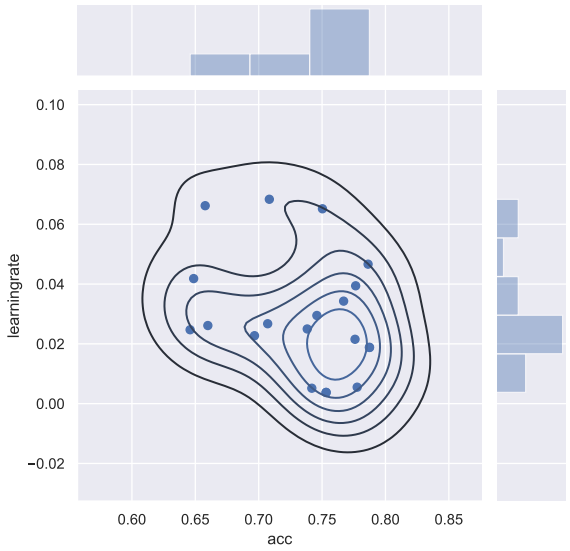

Abbildung 8.24: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

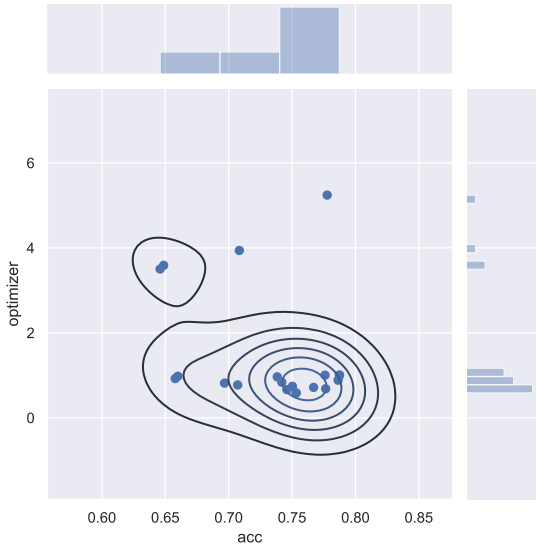

Abbildung 8.25: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.1.6 250 Iterationen des Genetischen Algorithmus des Convolutional Neural Network

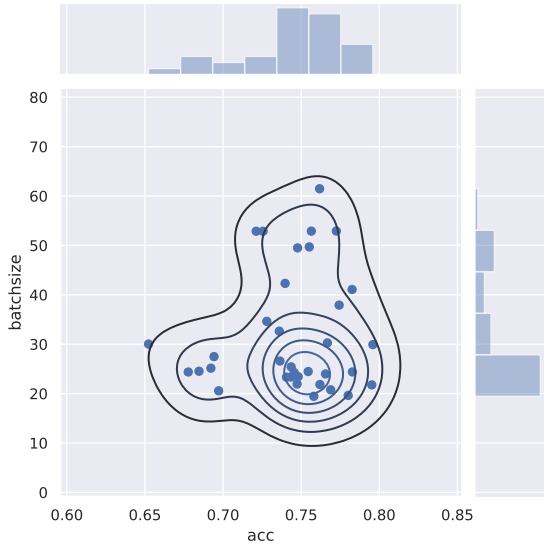

Abbildung 8.26: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

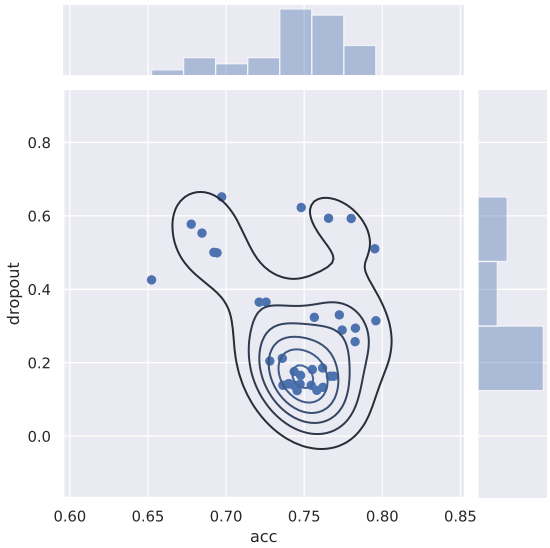

Abbildung 8.27: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

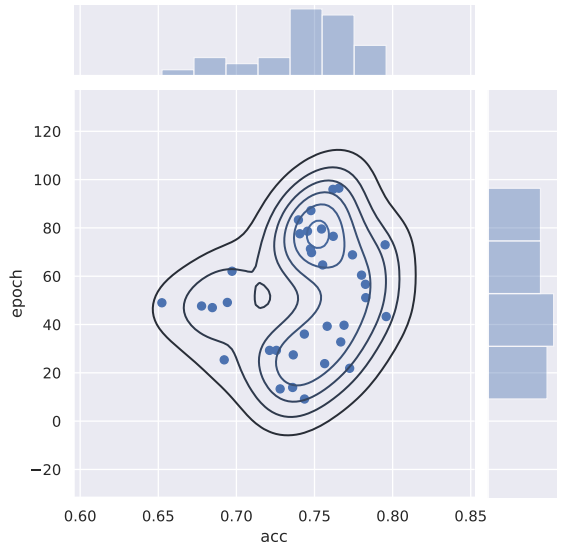

Abbildung 8.28: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

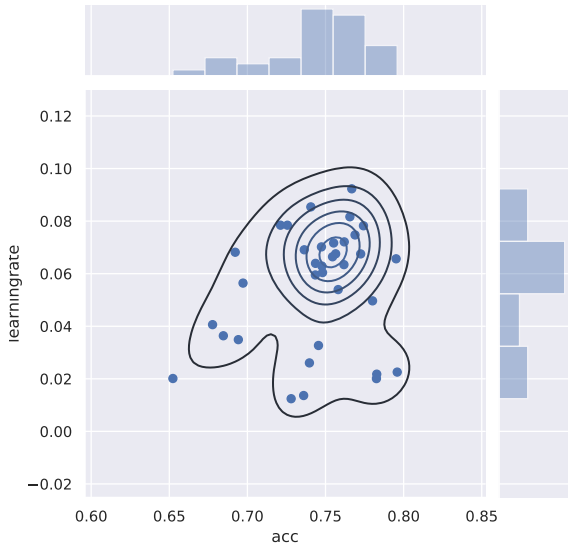

Abbildung 8.29: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

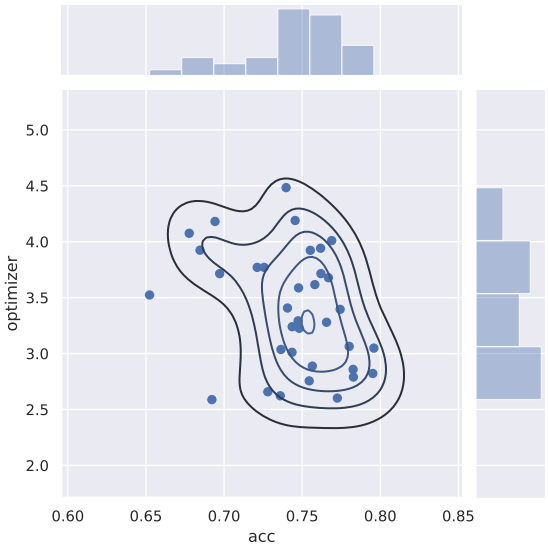

Abbildung 8.30: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2 Dichte-Diagramme zu den durchgeführten Optimierungen mit dem verkleinerten Datensatz

8.2.1 50 Iterationen des Genetischen Algorithmus des kleinen Fully-Connected Netzes

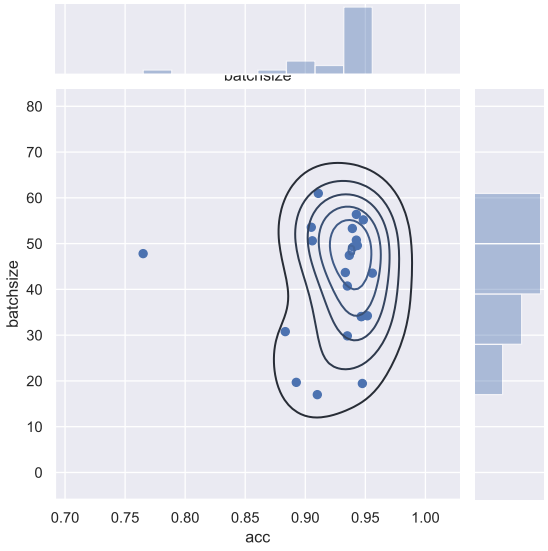

Abbildung 8.31: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

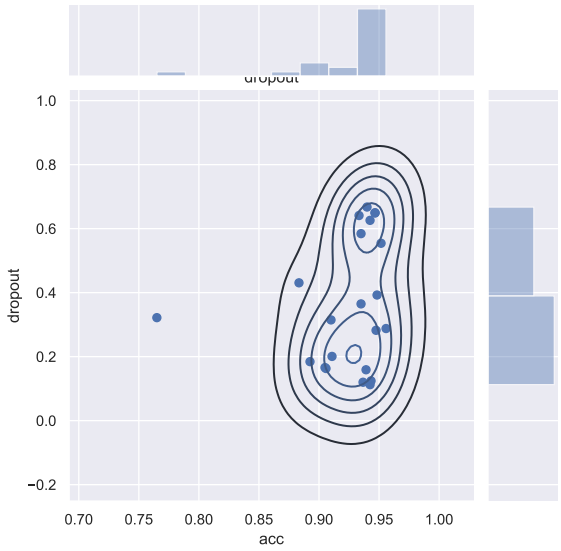

Abbildung 8.32: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

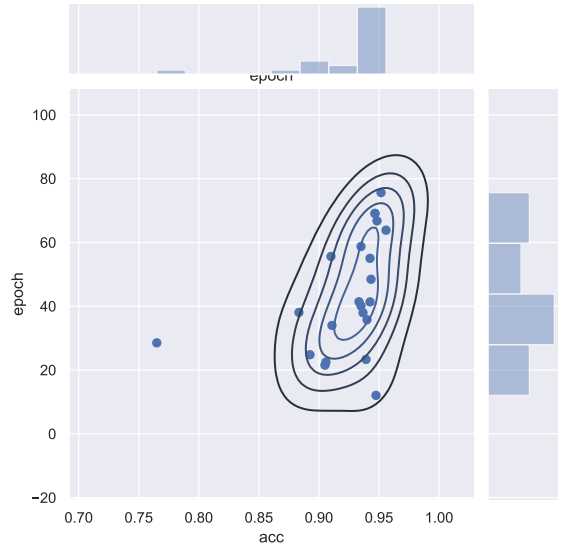

Abbildung 8.33: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

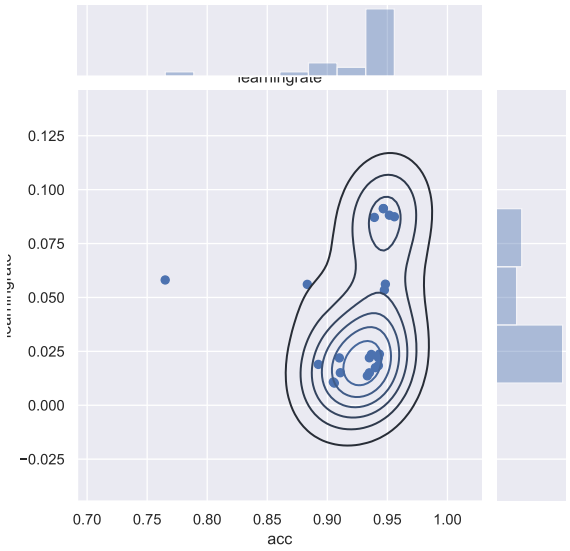

Abbildung 8.34: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

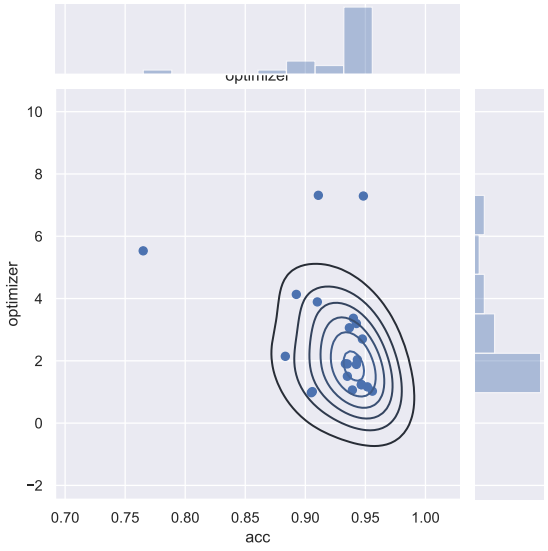

Abbildung 8.35: Dichte-Diagramm der Optimierer in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2.2 50 Iterationen des Genetischen Algorithmus des großen Fully-Connected Netzes

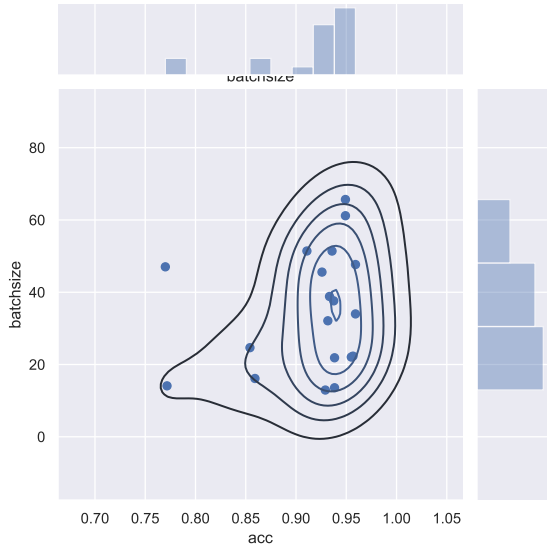

Abbildung 8.36: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

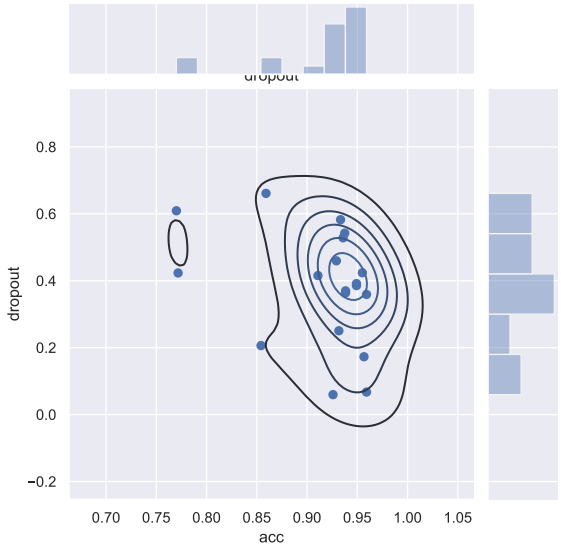

Abbildung 8.37: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

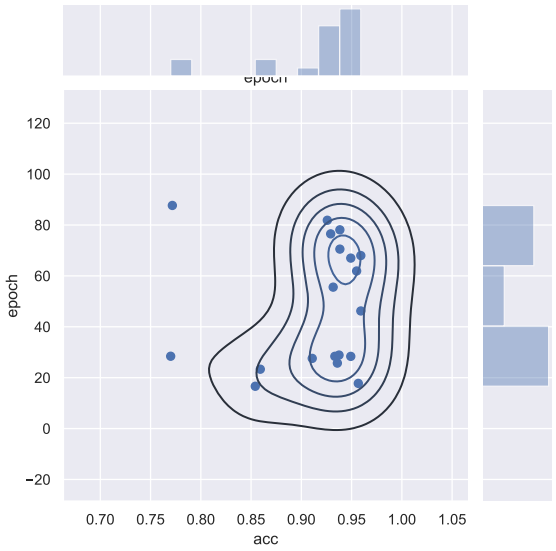

Abbildung 8.38: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

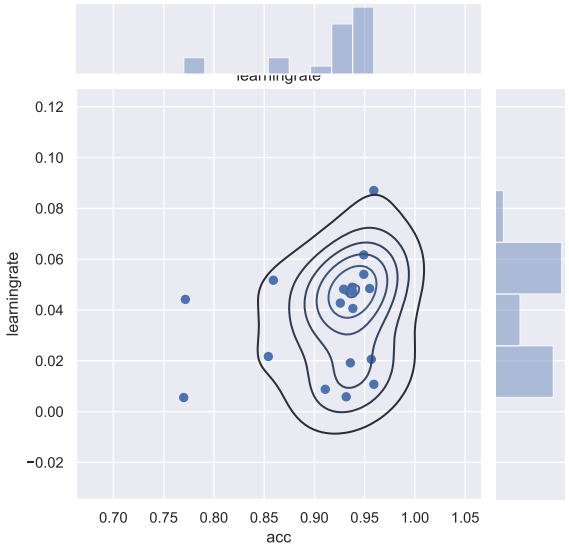

Abbildung 8.39: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

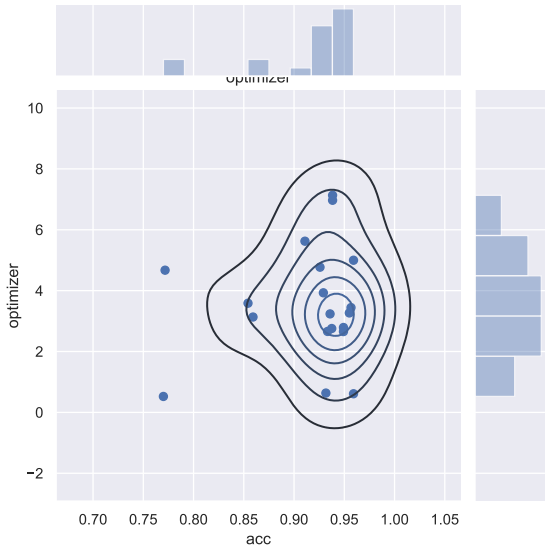

Abbildung 8.40: Dichte-Diagramm der Optimierer in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2.3 250 Iterationen des Genetischen Algorithmus des kleinen Fully-Connected Netzes

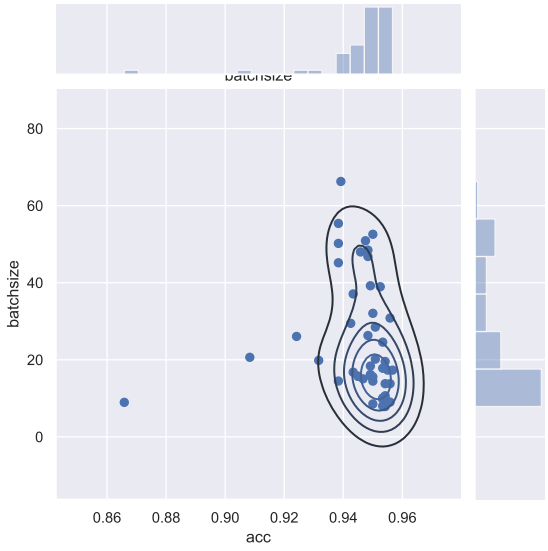

Abbildung 8.41: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

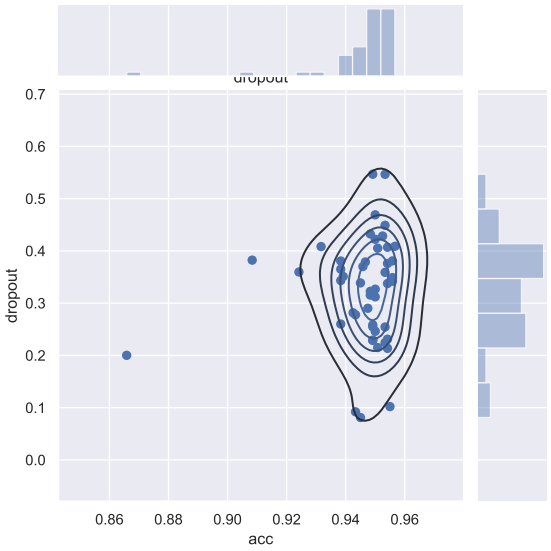

Abbildung 8.42: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

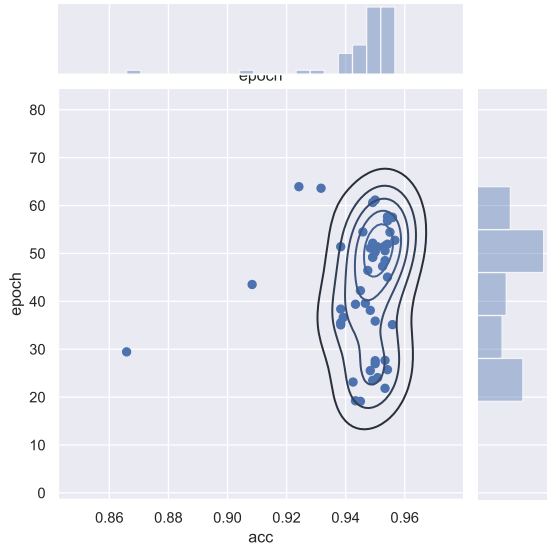

Abbildung 8.43: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

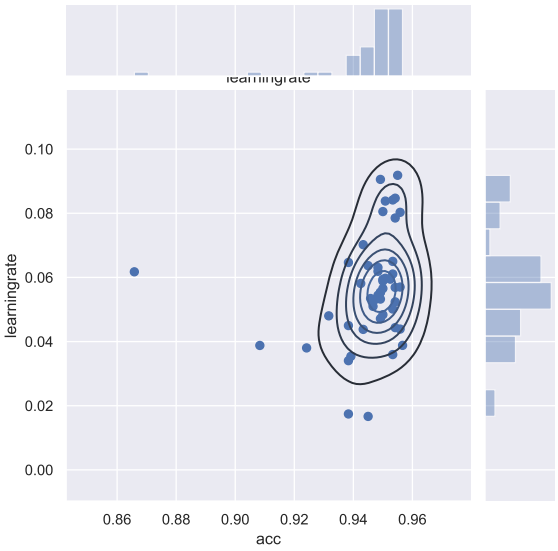

Abbildung 8.44: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

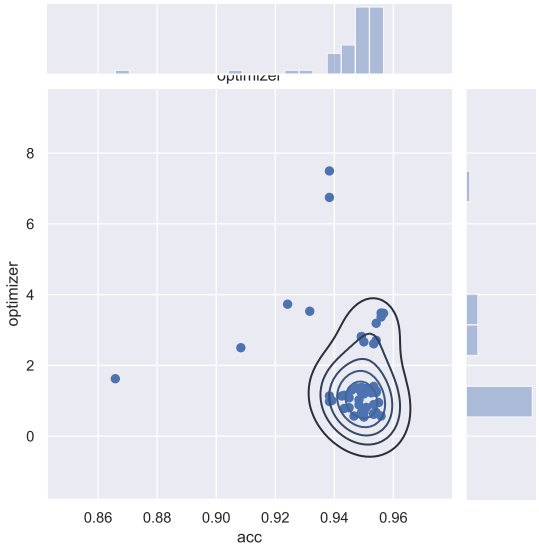

Abbildung 8.45: Dichte-Diagramm der Optimierer in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2.4 250 Iterationen des Genetischen Algorithmus des großen Fully-Connected Netzes

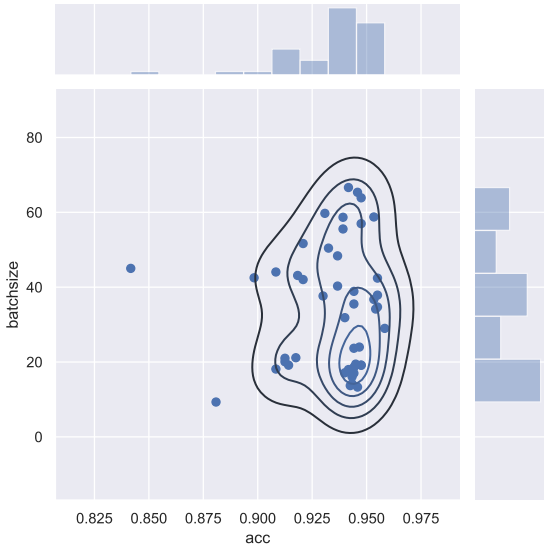

Abbildung 8.46: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

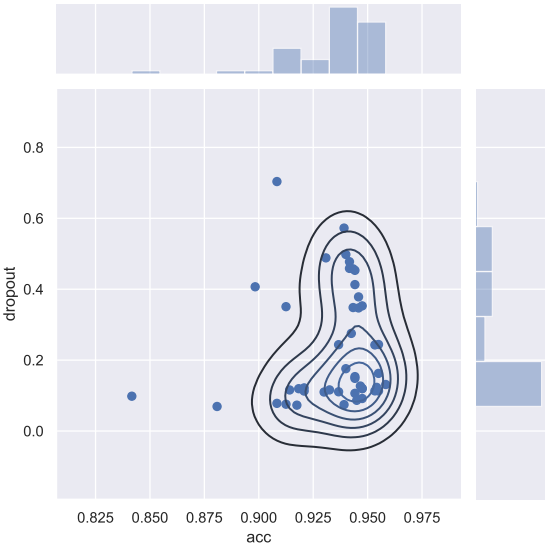

Abbildung 8.47: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

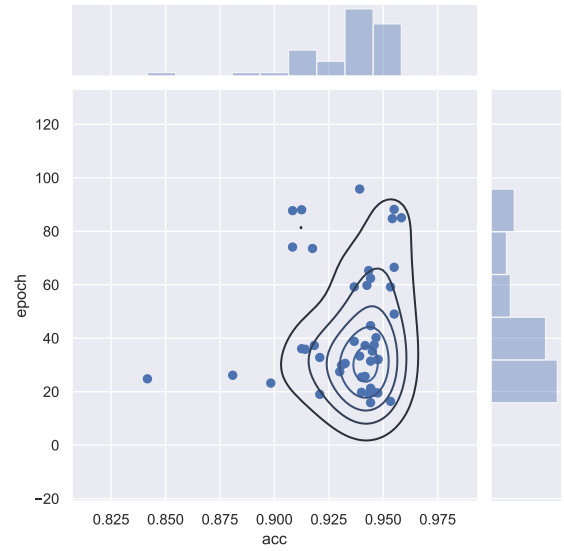

Abbildung 8.48: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

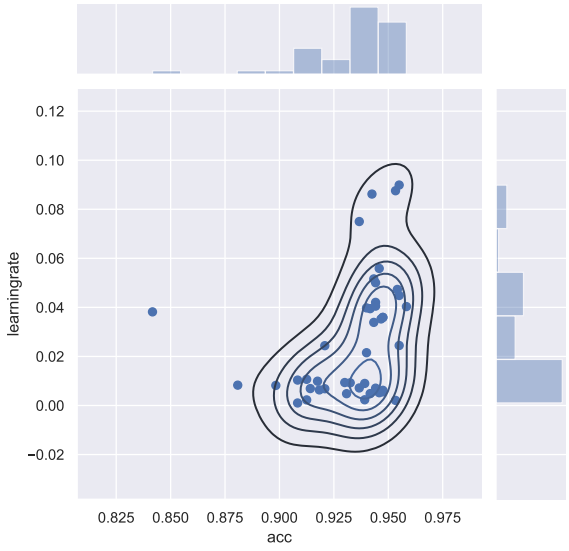

Abbildung 8.49: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

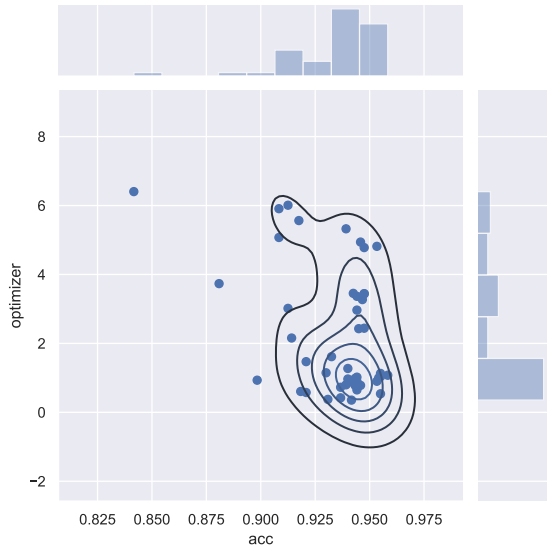

Abbildung 8.50: Dichte-Diagramm der Optimierer in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2.5 50 Iterationen des Genetischen Algorithmus des Convolutional Neural Network

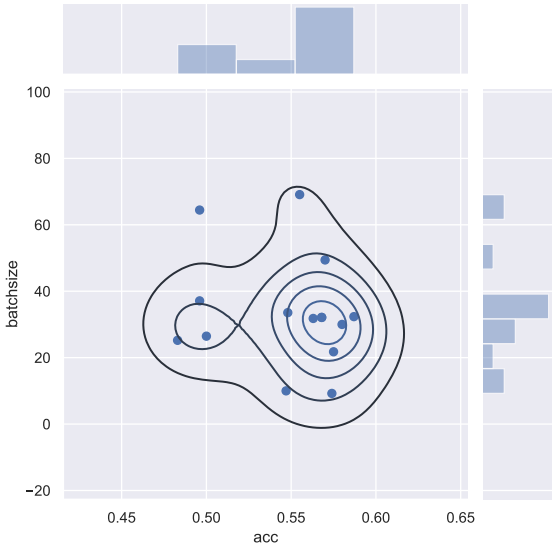

Abbildung 8.51: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

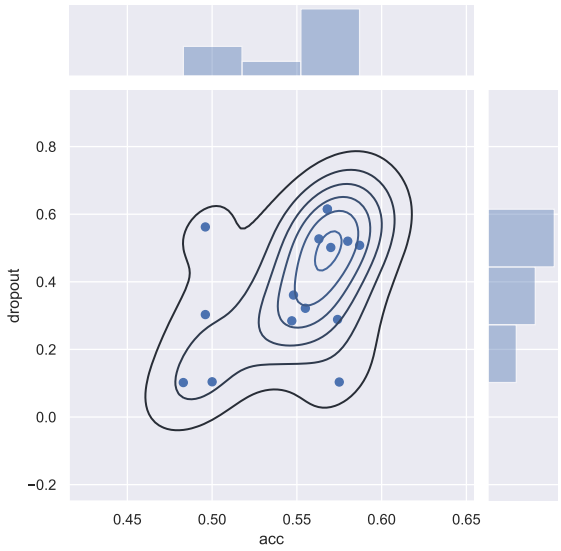

Abbildung 8.52: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

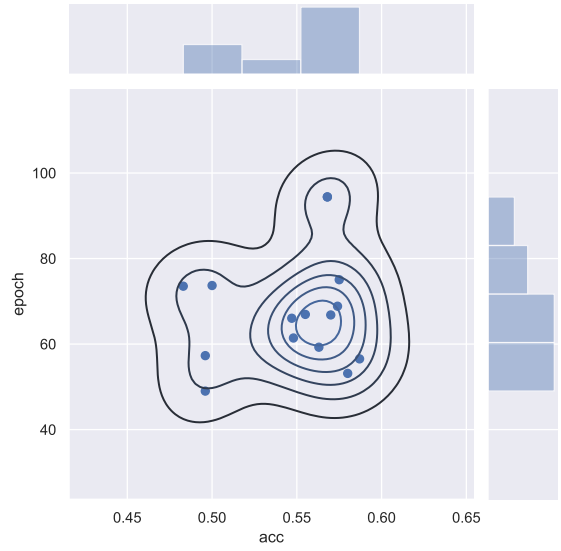

Abbildung 8.53: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

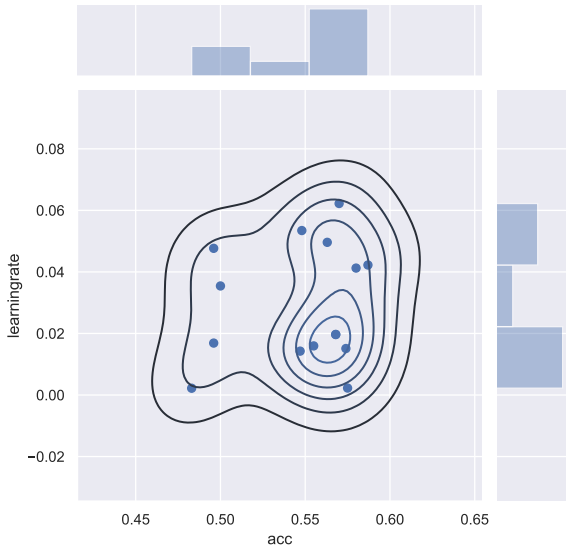

Abbildung 8.54: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

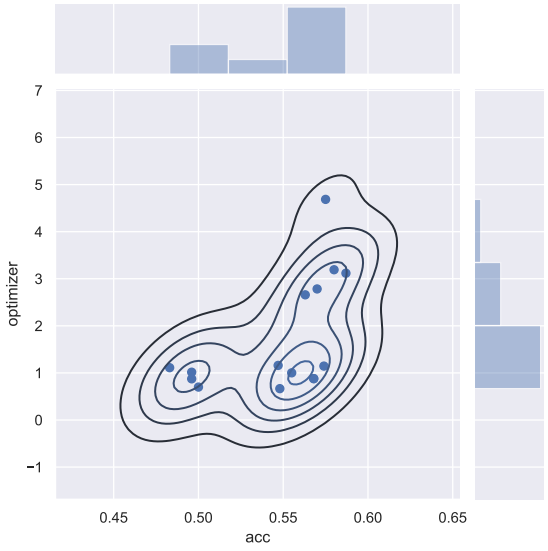

Abbildung 8.55: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

8.2.6 250 Iterationen des Genetischen Algorithmus des Convolutional Neural Network

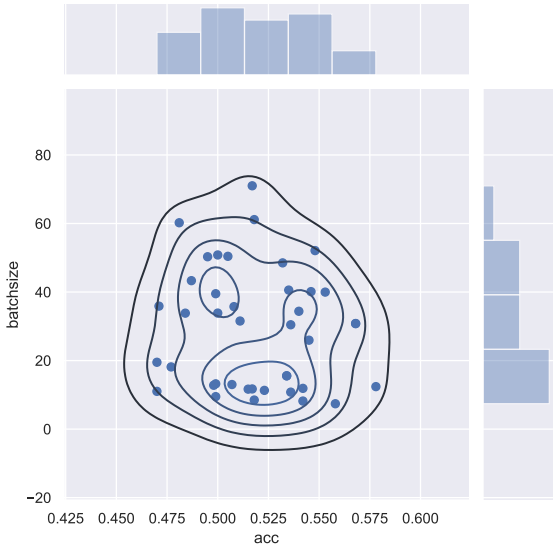

Abbildung 8.56: Dichte-Diagramm der Batchsize in Verbindung mit der Klassifizierungsgenauigkeit(acc)

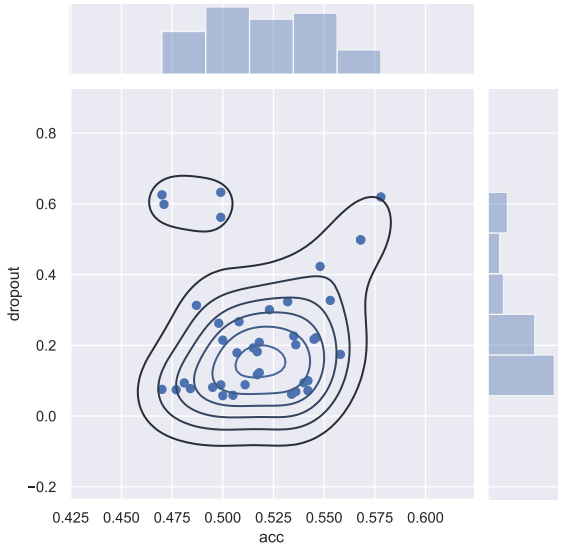

Abbildung 8.57: Dichte-Diagramm des Dropouts in Verbindung mit der Klassifizierungsgenauigkeit(acc)

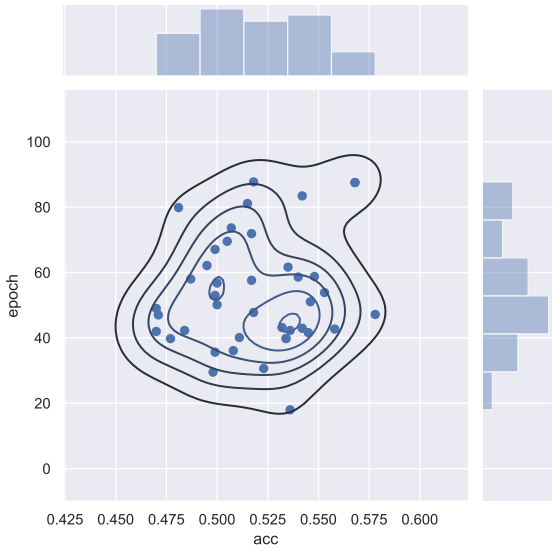

Abbildung 8.58: Dichte-Diagramm der Epochenanzahl in Verbindung mit der Klassifizierungsgenauigkeit(acc)

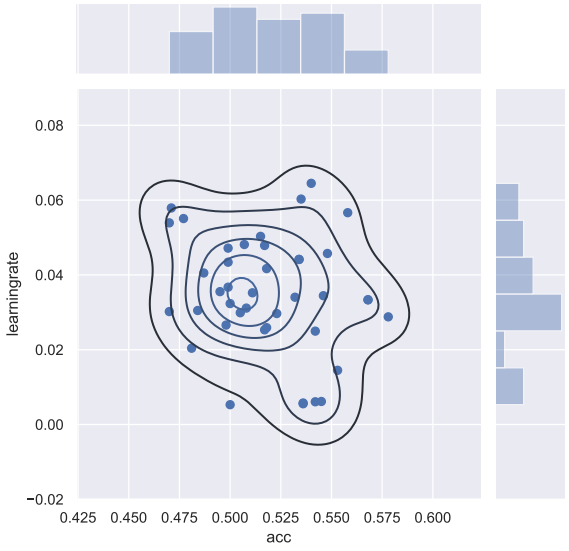

Abbildung 8.59: Dichte-Diagramm der Lernrate in Verbindung mit der Klassifizierungsgenauigkeit(acc)

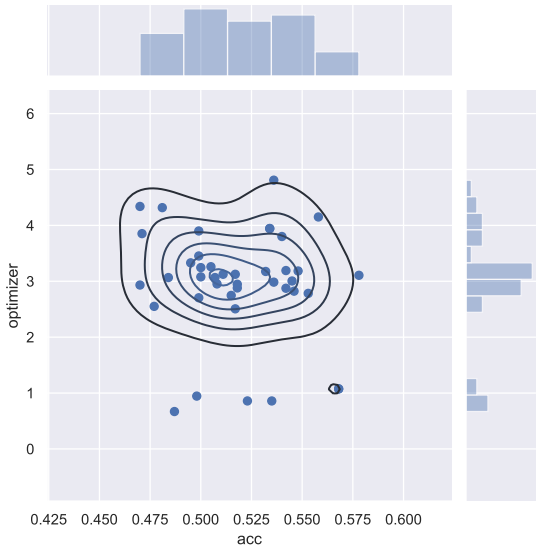

Abbildung 8.60: Dichte-Diagramm des Optimierers in Verbindung mit der Klassifizierungsgenauigkeit(acc)

Abbildungsverzeichnis