Dritter Teil
Schutzmöglichkeiten für Modelle maschinellen Lernens im Urheberrecht

Nachdem geklärt wurde, welche Voraussetzungen erfüllt sein müssen, um ML-Modelle rechtmäßig zu trainieren, stellt sich die Frage, wie die trainierten Modelle rechtlich geschützt sein könnten. Schließlich fließen viel Erfahrung, Arbeitszeit, Geld und technische Ressourcen in die Entwicklung effizienter und korrekt arbeitender Machine Learning-Modelle.[202]

In der Regel sind die trainierten Modelle das Ergebnis eines unter Umständen langwierigen Prozesses des Trainingsdatensammelns, -auswertens und -aufbereitens, der Modelleinstellung und -optimierung, des Modelltrainings, der Ergebnisauswertung und ggf. iterativen Überarbeitung. Daher sprechen sowohl die Wertschätzung der Arbeit der Entwickler maschineller Lernmodelle bzw. Systeme als auch der Innovationsanreiz für einen solchen Schutz.

Problematisch in der Diskussion um den Schutz von ML-Modellen scheint, dass es bisher im juristischen Diskurs noch nicht greifbar genug war, was ein trainiertes Modell ausmacht.

Schon bei der Betrachtung des Entstehungsprozesses eines trainierten Modells wird klar, dass hier eine nicht unerhebliche menschliche Leistung erforderlich ist, um zum gewünschten Ergebnis zu kommen. Es ist mitnichten lediglich ein mathematischer Rechenvorgang, der das trainierte Modell erzeugt, sondern es ist Erfahrung in Bezug auf die Auswahl von Trainingsdaten sowie insbesondere die Strukturierung des Modells erforderlich.

Wirtschaftlich gesehen, ist das trainierte Modell ein hochwertiges Werkzeug, das einerseits im Produktiveinsatz bestehen muss, andererseits aber – ungeschützt – einfach kopiert und ausgebeutet werden kann.

Der urheberrechtliche Schutz des „Gehirns von KI“ – der Modelle maschinellen Lernens – wurde bereits in einigen Veröffentlichungen diskutiert

202 Eine Übersicht der Entwicklung weltweiter Investitionen in KI kann z. B. dem AI Index Report 2019 entnommen werden: *Perrault* et al., The AI Index 2019 Annual Report, S. 94 ff.; vgl. außerdem *Graf*, Multitalent für Sprache (Spektrum.de vom 11.08.2020).

bzw. zumindest angerissen,[203] jedoch bisher an keiner der Autorin bekannten Stelle ausreichend tiefgehend bzw. zutreffend mit Blick auf die tatsächlich verwendete Technologie durchleuchtet. Dessen nimmt sich dieser Teil der Arbeit an. Dazu werden die relevanten Bausteine von ML-Systemen identifiziert (s. § 6) und daraufhin auf ihre Schutzfähigkeit kategorisch untersucht (s. § 7).

Auch außerhalb der Vorschriften des Urheberrechtsgesetzes wird ein Schutz für ML-Modelle diskutiert. So wird etwa ein Schutz als Geschäftsgeheimnis im Sinne der GeschGeh-RL bzw. des GeschGehG angedacht,[204] und auch das Wettbewerbsrecht könnte Anknüpfungspunkte[205] bieten.

Vorgeschlagen wird zudem die Einführung eines Leistungsschutzrechts für computergenerierte Erzeugnisse (mit dem Ziel, die errechneten „Trai-

203 *Ehinger/Stiemerling*, CR 12 2018, 761 ff.; *Hauck/Cevc*, ZGE 11 2019, 135 ff.; *Papastefanou*, CR 4 2019, 209 ff.; *Linke*, GRUR Junge Wissenschaft 2019, 29 ff.; *Nebel/Stiemerling*, CR 1 2016, 61 ff.; *Hartmann/Prinz*, WRP 12 2018, 1431 ff.; *Gomille*, JZ Nr. 20 2019, 969, 970; *Spindler*, IIC 2019, 1049, 1050; *Linke/Petrlik*, GRUR Int. 2020, 39 ff.; *Grätz*, Künstliche Intelligenz im Urheberrecht; *Iglesias Portela/Shamuilia/Anderberg*, Intellectual Property and Artificial Intelligence: A Literature Review: EUR 30017 EN, S. 9;.

204 *Ehinger/Stiemerling*, CR 12 2018, 761, 769 erwägen einen Geschäftsgeheimnisschutz für „Trainingsergebnisse" im Rahmen der GeschGeh-RL, ebenso sieht *Gomille*, JZ Nr. 20 2019, 969, 970 einen Schutz gem. §§ 2 ff. GeschGehG einschlägig; vgl. außerdem ausführlich *Hauck/Cevc*, ZGE 11 2019, 135, 163 f.; *Söbbing*, Fundamentale Rechtsfragen zur künstlichen Intelligenz. (AI Law), S. 14; einen Schutz für Modelle erwägend *Apel/Kaulartz*, RDi Nr.1 2020, 24, 29; insbesondere für „durch ein deutlich höheres Maß an Autonomie bestimmter" neuronaler Netzwerke auf GeschGeh-Schutz verweisend Schricker/Loewenheim–*Loewenheim/Leistner*, Urheberrecht, § 2 Rn. 41a, allerdings ist davon auszugehen, dass die Autoren nicht den Schutz neuronaler Netze an sich meinen, sondern Erzeugnisse – für diese wäre allerdings insbesondere im Zusammenhang der Erzeugung „künstlerischer" Werke fraglich, wie hier ein GeschGeh-Schutz anzubringen wäre: Denn die Komplexität der neuronalen Netze ändert nach hiesiger Ansicht nichts an der Zurechenbarkeit ihrer Erschaffung (also der Erschaffung der Netze) zum Entwickler.

205 *Hauck/Cevc*, ZGE 11 2019, 135, 166 f. nennen u. a. § 4 Nr. 3 UWG als Auffangtatbestand im Rahmen des lauterkeitsrechtlichen Nachahmungsschutzes.

ningsergebnisse“ zu schützen).[206] Naheliegend und bereits in der Praxis angekommen ist zudem der Schutz im Patentrecht.[207]

Diese Arbeit fokussiert jedoch auf diejenigen Schutzmöglichkeiten, die das UrhG bietet.

206 *Ehinger/Stiemerling*, CR 12 2018, 761, 769.

207 Ausführlich dazu *Hauck/Cevc*, ZGE 11 2019, 135 ff., die einen anwendungsbezogenen Patentschutz für möglich halten; gegen einen patentrechtlichen Schutz: *Gomille*, JZ Nr. 20 2019, 969, 970; *Apel/Kaulartz*, RDi Nr.1 2020, 24, 29, die ein Problem insbesondere in Bezug auf die erforderliche Technizität sehen; auf europäischer Ebene steigt die Zahl der (begehrten) Patentanmeldungen, vgl. *Iglesias Portela/Shamuilia/Anderberg*, Intellectual Property and Artificial Intelligence: A Literature Review: EUR 30017 EN, S. 6; das Europäische Patentamt hat Richtlinien zur Patentierung von KI herausgegeben, vgl. z. B. https://www.epo.org/law-practice/legal-texts/html/guidelines2018/e/g_ii_3_3_1.htm (Stand: 22.02.2021) und *Craglia* et al., Artificial intelligence: A European perspective, S. 66.

§ 6 Technische Bestandsaufnahme

Ein großer Teil der divergierenden Aussagen über den urheberrechtlichen Schutz von ML-Modellen könnte daher rühren, dass keine klare Vorstellung davon existiert, was eigentlich geschützt werden soll, oder zumindest nicht klar kommuniziert ist, was als Schutzgegenstand angenommen wird. Im Folgenden werden daher die künstlichen neuronalen Netze und Random Forest-Systeme – stellvertretend für eine große Zahl verschiedener ML-Modelle – analytisch in ihre Bestandteile zerlegt, mit dem Ziel, klare Schutzgegenstände zu identifizieren.

Um ML-Modelle einer im UrhG bestehenden Werkkategorie zuzuordnen, ist zunächst erforderlich, die Modelle näher zu betrachten, um anhand der identifizierten einzelnen Bestandteile eine Kategorisierung vornehmen zu können. Im Folgenden werden dazu die im Zeitpunkt der Entstehung der Arbeit überwiegend verwendeten Frameworks bzw. Technologien analysiert, Gemeinsamkeiten herausgearbeitet und im nächsten Abschnitt auf ihre urheberrechtliche Schutzfähigkeit hin untersucht.

A. Grundlegende Begriffe

Dieses Kapitel wird in die Strukturen und technischen Abläufe der Entstehung von ML-Modellen eintauchen. Dabei werden einige, in der Softwareentwicklung übliche, grundlegende Begriffe verwendet, die dem juristischen Leser möglicherweise nicht – oder nicht in diesem Kontext – geläufig sind und daher zum besseren Verständnis nachfolgend eingeführt werden.

I. Frameworks und Bibliotheken

Frameworks und Bibliotheken[208] sind zu verstehen als programmiersprachenspezifische Baukästen, die in ein bestehendes Programmierprojekt eingebunden werden können, und dann eine Vielzahl an Funktionalitäten bereitstellen.[209] Auf diese Funktionalitäten kann der Entwickler über vordefinierte Schnittstellen bzw. Befehle zugreifen, diese sind in der Regel in sogenannten Application Programming Interfaces (APIs) dokumentiert.

II. API

Die Abkürzung „API“ steht für *Application Programming Interface* und ist die Bezeichnung für eine Schnittstelle zu einer Software-Anwendung, die üblicherweise bereitgestellt wird, um die Kompatibilität von Programmen zu ermöglichen, etwa durch Datenaustausch oder Programmerweiterungen.[210] Häufig wird damit (unpräzise) auch eine Dokumentation aller in einem Framework oder einer Bibliothek vorhandenen Funktionen und Klassen bezeichnet. Die „Schnittstelle“ besteht selbst aus Programmteilen, die zum Beispiel über das Internet von anderen Programmen angesteuert werden können, oder innerhalb des ausgeführten Programms, wenn die „Fremdanwendung“ bzw. Bibliothek dort eingebunden wurde.

III. Objekte und Funktionen

Im Rahmen dieser Arbeit werden die Begriffe „Objekt“ und „Funktion“ häufig verwendet, weshalb an dieser Stelle ein Gefühl für die Bedeutung derselben im Softwarekontext vermittelt werden soll. Grundsätzlich ist zu

---

208 Die Begriffe „Framework“ und „Bibliothek“ werden in dem Kontext nicht trennscharf verwendet – so beschreibt etwa der Autor von *Keras* sein Projekt eher als eine „Schnittstelle“ als ein „Framework“ (vgl. https://github.com/keras-team/keras/issues/5050#issuecomment-272945570 (Stand: 22.02.2021)), während Wikipedia *Keras* als „Bibliothek“ bezeichnet (vgl. https://de.wikipedia.org/wiki/Keras (Stand: 22.02.2021)) und auf der projekteigenen Website ist die Rede vom „most used deep learning framework“(vgl. https://keras.io/ (Stand: 22.02.2021)). Vorliegend wird deshalb auch das Begriffspaar Framework/Bibliothek verwendet.

209 Vgl. *Gamma* et al., Design Patterns, S. 26.

210 Vgl. auch *Fischer/Hofer*, Lexikon der Informatik, S. 45.

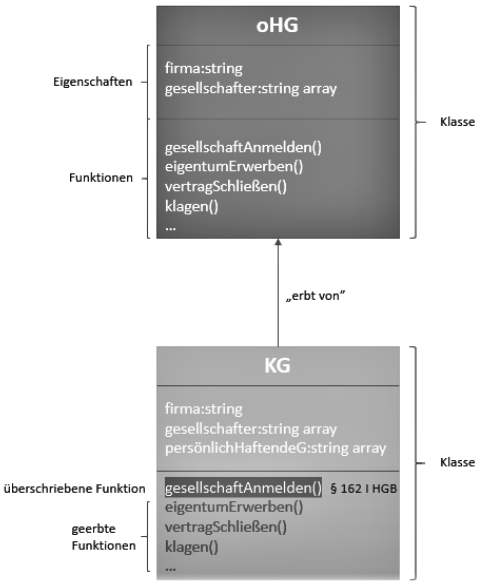

Abbildung 6.1: „Objektorientierung" im HGB, eigene Darstellung.

unterscheiden zwischen prozeduraler und objektorientierter Programmierung (OOP). Im ML-Kontext kommt in der Regel OOP zum Einsatz. Diese lässt sich für Juristen anhand einem übertragenden Beispiel aus dem Handelsgesetzbuch erklären, vgl. dazu Abbildung 6.1[211]:

In § 105 HGB wird die offene Handelsgesellschaft (oHG) definiert. Dabei werden ihr „Eigenschaften" zugewiesen: Ihr Zweck ist auf den Betrieb eines Handelsgewerbes unter gemeinschaftlicher Firma gerichtet. Außerdem ist bei keinem der Gesellschafter die Haftung gegenüber den Gesellschaftsgläubigern beschränkt. Zudem hat die oHG Fähigkeiten („Funktionen"): Sie kann gem. § 124 Abs. 1 HGB unter ihrer Firma Rechte erwerben und Verbindlichkeiten eingehen. Sie kann Eigentum und andere dingliche Rechte an Grundstücken erwerben, vor Gericht klagen und verklagt werden. Das Gesetz definiert also ein abstraktes Konstrukt, das im Rahmen objektorientierter Programmierung als „Klasse" bezeichnet würde.

211 „String" bezeichnet den Typ der Eigenschaft „Firma" und steht für einen Wert in Form einer Zeichenkette bzw. „Text". Ein „Array" bezeichnet eine Sammlung von Einzelwerten. () weisen auf eine Funktion hin.

Eine andere „Klasse“ im HGB ist die Kommanditgesellschaft (KG). Diese wird in § 161 Abs. 1 HGB definiert, sie hat ebenfalls die „Eigenschaft“, dass ihr Zweck auf den Betrieb eines Handelsgewerbes unter gemeinschaftlicher Firma gerichtet ist. Außerdem ist bei einigen Gesellschaftern die Haftung gegenüber den Gesellschaftsgläubigern auf den Betrag einer bestimmten Vermögenseinlage beschränkt, während bei einem anderen Teil der Gesellschafter eine Beschränkung der Haftung nicht stattfindet. In § 161 Abs. 2 HGB wird außerdem bestimmt, dass die Vorschriften, die für die oHG gelten, auch für die KG Anwendung finden, wenn nicht in dem Abschnitt über die KG etwas anderes bestimmt ist. In der objektorientierten Programmierung ließe sich sagen: „Die Klasse KG erbt [“Eigenschaften„ und “Funktionen„] von der Klasse oHG“, das gilt, solange in der Klasse KG nicht „Eigenschaften“ und „Funktionen“ überschrieben werden („überschrieben“ wird eine Eigenschaft, wenn ihr ein anderer Wert zugewiesen wird – wenn also der Gesetzgeber, um beim Beispiel zu bleiben, für die KG speziellere Vorschriften festlegt, die für die oHG nicht gelten). Wie in der Gesetzgebung werden auch in der OOP Maßnahmen zur Vereinfachung dergestalt eingesetzt, dass soweit möglich Redundanzen vermieden und Gemeinsamkeiten zentral geregelt werden[212] (vgl. etwa auch den allgemeinen Teil gegenüber dem besonderen Teil einiger Gesetze).

„Klassen“ sind in der OOP-Welt wie abstrakte Schablonen, „Objekte“ hingegen sind konkrete Instanzen einer „Klasse“.[213] Beide bündeln „Funktionen“ und „Eigenschaften“.[214] Ein Unternehmen Müller, Meier, Schmidt oHG wäre ein Objekt (mit ausgefüllten Eigenschaften) der „Klasse“ oHG gem. § 105 HGB. Eine „Klasse“ gibt es innerhalb eines Projektes bzw. einer Bibliothek jeweils nur einmal, es kann aber unzählig viele „Objekte“ (Instanzen) dieser „Klasse“ geben.

„Funktionen“ (in Abbildung 6.1 durch runde Klammern gekennzeichnet, teilweise auch „Methoden“ oder „Operationen“ genannt[215]) kann eine Vielzahl von Programmbefehlen zugeordnet werden. So könnte die Funktion der oHG `vertragSchließen()` etwa die Vertragsparteien zu identifizieren haben, die essentialia negotii dokumentieren, eine Anweisung an eine Zahlungsstelle ausgeben etc.

212 Vgl. *Gamma* et al., Design Patterns, S. 15.
213 Vgl. Dies., Design Patterns, S. 14; „Instanz“ meint in der Softwareentwicklung einen konkreten Gegenstand vom Typ der „Klasse“, so wäre die „Karl Friedrich oHG“ eine Instanz vom Typ „oHG“.
214 Vgl. Dies., Design Patterns, S. 11.
215 Vgl. Dies., Design Patterns, S. 11.

B. Grundbausteine für ML-Modelle: Frameworks, Bibliotheken, APIs

Es ist heute nicht erforderlich, ein ML-Modell von Grund auf neu zu entwickeln. Den Programmierern stehen zahlreiche Frameworks und Bibliotheken zur Verfügung, die sie für die Umsetzung ihrer Konzepte nutzen können. Beispiele für solche Machine Learning-Frameworks sind für die Programmiersprache *Python TensorFlow*,[216] *Keras*,[217] *PyTorch*[218] und *Scikit-Learn*,[219] für die Sprache *R* insbesondere das Paket *randomForests*[220]. Diese Arbeit wird sich in den folgenden Kapiteln zur Analyse der ML-Modelle stets auf diese vier Frameworks und Bibliotheken und das *R*-Paket beziehen, um praxistaugliche Ergebnisse zu erzielen.

Mit diesen Frameworks können unterschiedlichste Arten von ML-Modellen realisiert werden – künstliche neuronale Netze sind nur eine davon. Insbesondere können auch mit den *Python*-Frameworks Random Forest-Modelle erzeugt werden. Die Ausführungen mit Bezug zu den Frameworks sind in der Regel für alle Modelltypen gültig, die mit diesen Frameworks und Bibliotheken umgesetzt werden können. Es wird daher einheitlich von ML-Modellen gesprochen. Eine Abgrenzung erfolgt lediglich zu den Random Forests in *R*, da sich hier strukturelle Unterschiede zu den Modellen in *Python* ergeben.

In Bezug auf die Frage nach dem Schutzgegenstand sind die Frameworks und Bibliotheken jedoch der falsche Anknüpfungspunkt. Sie mögen zwar ausschnittsweise in dem Programm enthalten sein, das letztendlich in der Lage ist, die Aufgaben zu lösen, die dem ML-Modell gestellt wurden, jedoch sind sie – für sich genommen – eher mit der Palette des Malers vergleichbar, der für die Umsetzung seines Werkes daraus erst noch Farben, Intensität, Anordnung und Motiv bestimmen muss. Wenngleich der Schutz der Frameworks und Bibliotheken selbst auch diskutiert werden könnte[221] ist dies jedoch nicht Ziel dieser Arbeit, und wird daher nicht thematisiert.

---

216 *Abadi* et al., TensorFlow: A system for large-scale machine learning, 265 ff..
217 *Chollet* et al., Keras.
218 *Paszke* et al., Automatic differentiation in PyTorch.
219 *Buitinck* et al., API design for machine learning software: experiences from the scikit-learn project.
220 *Liaw/Wiener*, R News 2 Nr. 3 2002.
221 Vgl. z. B. Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 23 zum Schutz von *Interfaces*; Schricker/Loewenheim–*Spindler*, Urheberrecht, § 69a Rn. 11 zu Programmbibliotheken.

C. *Quellcode*

Sowohl in der Vorbereitung der Trainingsdaten als Input für ML-Modelle, als auch im Training und dem Produktiveinsatz von KI spielt Quellcode – in Abgrenzung zu Parameterwerten und Hyperparametern[222] – eine wesentliche Rolle. „Quellcode" wird der vom Menschen geschriebene, menschenlesbare Programmcode genannt, der dann von speziellen Programmen „verstanden" werden und in Anweisungen an den Computer übersetzt werden kann.[223] Er ist erforderlich, um einem ML-Modell Gestalt zu verleihen, Daten einzulesen, Daten zu bearbeiten bzw. vorzubereiten, vorgefertigte Modelle einzubinden, diese mit Daten zu versehen und den Trainingsvorgang anzustoßen. Er ist quasi das „Drehbuch" für eine Vorhersage eines Wertes oder die Erzeugung eines Bildes mittels eines ML-Modells. Er gibt die Akteure vor, ihre Gestalt, wann sie die Bühne betreten und mit wem sie interagieren. Ist ein ML-Modell trainiert, kann es mithilfe von Quellcode in die gewünschte Umgebung (etwa eine Webseite oder eine umfangreiche Analysesoftware) eingebunden und ausgeführt werden.

D. *Trainiertes Modell in* Python

Im Umgang mit ML-Modellen, die mit den beschrieben *Python*-Frameworks bzw. Bibliotheken erzeugt werden, ist das Ziel der Entwicklung immer ein fertiges, trainiertes ML-Modell, das schließlich zur Bilderkennung oder Bildgenerierung, Regression oder einer anderen zu Trainingsbeginn definierten Aufgabe produktiv eingesetzt werden soll. Ein trainiertes Modell stellt mitunter den Kern eines neuen Produktes dar, mit dem ein Unternehmen in den Markt einsteigen möchte. Es ist daher verständlich, dass ein großes Interesse daran besteht, das Ergebnis exklusiv verwerten und andere von der Nutzung ausschließen zu können. Fraglich erscheint jedoch, ob das Urheberrecht dies leisten kann. Wenngleich die Zahl der Konflikte bisher noch überschaubar sein mag (ein Großteil der Entwicklung von ML-Modellen bewegt sich im Open Source-Sektor, zudem wird auch noch sehr viel an der Technologie geforscht, sodass viele Entwickler bereit sind, ihre Ergebnisse freigiebig mit anderen Forschern zu teilen) so ist es doch absehbar, dass in der Zukunft durchaus Regelungsbedarf bestehen wird, insbesondere wenn Entwickler be-

222 Vgl. zur Begriffsklärung oben § 2 B.III.6..
223 Vgl. *Fischer/Hofer*, Lexikon der Informatik, S. 721.

ginnen, ihre Ergebnisse nicht mehr unter sehr liberale Open Source-Lizenzen zu stellen (bisher kommen vor allem die MIT- und die BSD-Lizenz zum Einsatz).[224] Aber ist eine Lizenzierung überhaupt möglich? Besteht geistiges Eigentum an einem fertig trainierten Modell? Oder können andere Rechte in Anspruch genommen werden?

Um diese Frage zu beantworten, gilt es zunächst, den Begriff „trainiertes ML-Modell" genau zu umreißen. Hier wird als „ML-Modell" jede Form von Modellen[225] erfasst, die unter Einsatz der genannten Frameworks und Bibliotheken mit *Python* erzeugt werden können. Eine Beschränkung auf künstliche neuronale Netze erfolgt nicht, weil die Ausführungen zwar auch, aber nicht ausschließlich auf künstliche neuronale Netze zutreffen. Was ein trainiertes ML-Modell technisch ausmacht, wird folgend in § 6 D.II. erläutert.

In der Literatur wird hinsichtlich eines trainierten Modells bisher häufig entweder nur auf die „Trainingsergebnisse" abgestellt[226] oder implizit unter dem Begriff „trainiertes künstliches neuronales Netz" nur die Parameter bzw. Gewichtungsinformationen berücksichtigt[227]. Übersehen wird dabei jedoch, dass es für die Erzielung tauglicher Ergebnisse auch und gerade auf die Hyperparameter ankommt[228] und folglich diese möglicherweise in den Begriff des trainierten Modells einzubeziehen sind.

I. Vorab: Einsatz eines trainierten Modells

In der Praxis wird beim Einsatz eines ML-Modells für die aufgabengemäße Verwendung üblicherweise wie folgt vorgegangen, vgl. dazu auch den Teil

224 Vgl. MIT-Lizenz: https://spdx.org/licenses/MIT.html#licenseText (Stand: 22.02.2021); BSD-Lizenz: https://spdx.org/licenses/BSD-3-Clause (Stand: 10.02.2021); „liberal" bedeutet in diesem Kontext, dass die Lizenzen es nicht untersagen, den Quellcode in Projekte bzw. Produkte einzubinden, die den Quellcode nicht für jedermann verfügbar offenlegen, es wird dann auch von „Non-Copyleft-Lizenzen" gesprochen, vgl. https://ifross.org/?q=welche-lizenztypen-gibt-es-bei-open-source-software-und-unterscheiden-sie-sich (Stand: 22.02.2021).

225 Vgl. zum Begriff des Modells schon § 2 B.I.4..

226 *Ehinger/Stiemerling*, CR 12 2018, 761, 766 Rn. 45; *Hauck/Cevc*, ZGE 11 2019, S. 161 ff..

227 *Hartmann/Prinz*, WRP 12 2018, 1431, 1437 Rn. 62.

228 Dies wird teilweise erkannt und als „Architektur" oder „Topologie" bezeichnet, jedoch stets separat von den Parametern gesehen und nicht als ein Ganzes betrachtet, vgl. z. B.Dies., WRP 12 2018, 1431, 1434.

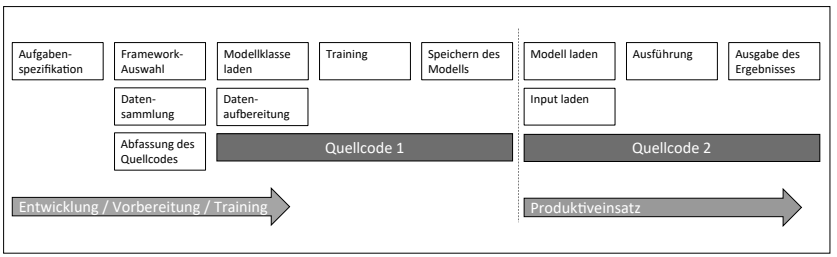

Abbildung 6.2: Entstehung eines ML-Modells, vereinfachte, eigene Darstellung.

„Produktiveinsatz“ in der Abbildung 6.2[229], nachdem das ML-Modell trainiert und gespeichert wurde:[230]

Es wird ein Programm (z. B. in der Programmiersprache *Python*) entwickelt, das – mindestens – erforderliche Bibliotheken lädt – vgl. in Abbildung 6.2 *Quellcode 2*.

Zudem werden – je nach dem welche Elemente des ML-Modells gespeichert vorliegen – entweder die gesamte Konfiguration des Modells, also die Hyperparameter und die Parameter, oder nur einzelne Informationen oder Informationspakete (abhängig von dem verwendeten Framework) geladen („Modell laden“). Alternativ, wenn das gesamte zuvor trainierte ML-Modell-Objekt im Binärformat gespeichert wurde, wird entsprechend das Objekt geladen („Modell laden“). Zudem werden die Daten geladen, für die eine Vorhersage durchgeführt werden soll („Input laden“).

Anschließend wird die Vorhersagefunktion (z. B. `predict()`) aufgerufen, um das konkrete Problem oder die konkrete Aufgabe (etwa die Objekterkennung in einem Bild) zu lösen („Ausführung“). Erst nach der Ausführung dieses Codes wird die Lösung der Aufgabe oder des Problems ausgegeben.

229 Abbildung 6.2 soll nur dazu dienen, wesentliche Schritte in der Modellentstehung visuell nachzuverfolgen. Vernachlässigt werden dabei u. a. Prozessschritte iterativer Überarbeitung und Anpassung sowie Test- und Evaluierungsphasen.

230 Vgl. z. B. für ein Minimalbeispiel in *Scikit-Learn*, das den hier dargestellten Ablauf in Code umsetzt: https://scikit-learn.org/dev/modules/model_persistence.html#python-specific-serialization (Stand: 22.02.2021).

II. Begriff des trainierten Modells

Für die konkrete Problemlösung kommt es also maßgeblich darauf an, was am Ende der Ausführung von *Quellcode 1* abgespeichert wurde, denn das Abgespeicherte wird dann in *Quellcode 2* geladen („Modell laden"). Andernfalls müsste das Modell für jeden Einsatz vollständig neu trainiert werden. Je nach verwendeter Programmiersprache und Framework stehen dem Entwickler zum Abspeichern des Modells und seiner Bestandteile unterschiedliche Möglichkeiten zur Verfügung. Die Technologie wird ständig weiterentwickelt, eine Betrachtung der Möglichkeiten kann folglich nur eine Momentaufnahme darstellen.

Dennoch ist es der urheberrechtlichen Bewertung zuträglich, sich mit den bisherigen Möglichkeiten auseinanderzusetzen, um ein Grundverständnis für die relevanten Vorgänge zu entwickeln. Es folgt daher eine Übersicht der Speichervarianten jeweils in *TensorFlow*, *Keras*, *PyTorch* und *Scikit-Learn*. Identifiziert werden soll dabei, in welcher Form die Ergebnisse gespeichert – und dementsprechend auch wieder geladen – werden, um anschließend den passenden urheberrechtlichen Schutz ermitteln zu können. Es werden nur Lösungen basierend auf der Programmiersprache *Python* thematisiert, Ziel ist es aber, am Ende dieses Kapitels auch auf andere Programmiersprachen übertragbare Grundsätze zu entwickeln.

1. „Trainiertes Modell" in *TensorFlow*

TensorFlow ist eine „End-to-End Open Source Machine Learning Plattform".[231] Es handelt sich um eine Software-Bibliothek, die seit November 2015[232] von *Google* entwickelt wird und unter der Open Source-Lizenz Apache License 2.0 für die Entwicklung von Machine Learning-Projekten zur Verfügung steht. Die Plattform basiert auf der Programmiersprache *Python* und ermöglicht es, mittels vorgefertigter Klassen und Funktionen, ML-Projekte zügig umzusetzen.

TensorFlow bietet verschiedene Möglichkeiten, die durch das Training erarbeiteten Fortschritte des Modells zu sichern. Zum einen speichert *TensorFlow* während des Trainings immer wieder sogenannte Checkpoint-Dateien. Das

231 https://www.tensorflow.org (Stand: 22.02.2021).
232 Geht hervor aus der Versionsgeschichte auf GitHub, https://github.com/tensorflow/tensorflow/commits/master/RELEASE.md (Stand: 22.02.2021).

sind Dateien im für Menschen nicht lesbaren Binärformat, die ein Verzeichnis berechneter Tensoren,[233] also Parameter bzw. Gewichtungsinformationen, beinhalten.[234]

Mit jedem Trainingsvorgang wird die aktuellste Checkpoint-Datei geladen und das Modell damit initialisiert, sodass eine Fortsetzung des Trainings nach einer Unterbrechung möglich ist und nicht von vorne begonnen werden muss.[235]

Ist das Training abgeschlossen, kann das Modell zum Beispiel mittels der Funktion `tf.saved_model.save()` exportiert und im Anschluss mittels `tf.saved_model.load()` wieder eingelesen werden.[236] Das hat zur Folge, dass eine sogenannte *ProtocolBuffer*-Datei (saved_model.pb) und zwei Verzeichnisse namens „assets“ und „variables“ erzeugt werden.[237] Das Verzeichnis „variables“ enthält einen Training-Checkpoint, „assets“ enthält zusätzliche Informationen (zum Beispiel Textdateien für Vokabeltabellen, falls textverarbeitende Modelle entwickelt werden).[238] *TensorFlow* stellt außerdem (noch, dies ist eine Funktion aus der *TensorFlow* Version 1) die Funktionalität `Saver.save()` bereit, mit der bei entsprechender Einstellung vier Dateien erzeugt werden können – eine enthält dann die Parameter (.data), eine zweite weitere Informationen zum Checkpoint-Index (.index), eine dritte eine Liste aller zu speichernden Checkpoints (.pb) und eine die Graph-Struktur (.meta).[239]

Fraglich ist also, welche dieser bereitgestellten Funktionalitäten dem Schutzgegenstand „trainiertes ML-Modell“ im Sinne dieser Arbeit entspricht. Die Checkpoints enthalten keine Informationen über die Struktur des Modells, sind also nicht dazu einsetzbar, das Modell ohne anderweitige Informationen zu laden. Dazu sind hingegen grundsätzlich die Ergebnisse von `Saver.save()` und `tf.saved_model.save()` geeignet. Diese Arbeit stellt daher im weiteren Verlauf auf die Ergebnisse der Aufrufe von `tf.saved_model.save()` sowie `Saver.save()` ab, wenn es um ein „trainiertes Modell“ in *TensorFlow* geht.

233 Zum Begriff vgl. Fußnote 121; eine Einführung in Tensoren gibt auch *TensorFlow*: https://www.tensorflow.org/guide/tensor (Stand: 22.02.2021).
234 https://www.tensorflow.org/guide/checkpoint (Stand: 22.02.2021).
235 Initialisierung: Setzen von Anfangswerten bei Start eines Programmes, vgl. auch https://www.dwds.de/wb/Initialisierung (Stand: 22.02.2021).
236 Vgl. https://www.tensorflow.org/guide/saved_model (Stand: 22.02.2021).
237 Vgl. https://www.tensorflow.org/guide/saved_model (Stand: 22.02.2021).
238 Vgl. https://www.tensorflow.org/guide/saved_model (Stand: 22.02.2021).
239 Vgl. https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/Saver (Stand: 22.02.2021).

2. „Trainiertes Modell“ in *Keras*

Auf der Projekt-Website wird *Keras* beschrieben als „The Python Deep Learning Library“, bzw. als „High-Level Neural Network API“, die auf *TensorFlow* aufsetzt.[240] *Keras* stellt diverse Möglichkeiten bereit, ein trainiertes Modell zu exportieren bzw. zu speichern.[241]

- `model.save(<Dateipfad>)` Erzeugt eine Datei im binären HDF5-Format (*.h5), die die Architektur des Modells beinhaltet sowie die berechneten Gewichte, die Trainingskonfiguration (etwa die Verlust- und Optimierungsfunktion)[242] sowie den Zustand des Optimierers[243] im Zeitpunkt des Abspeicherns.[244] Mithilfe dieser Datei kann das Training dort fortgesetzt werden, wo es unterbrochen wurde. Alternativ kann auch das SavedModel-Format von *TensorFlow* verwendet werden.
- `model.to_json() / model.to_yaml()` Speichert nur die Architektur des Modells in einem menschenlesbaren Format (nicht binär, sondern strukturierte Textdaten entweder im JSON- oder YAML-Format).[245]
- `model.save_weights()` Erzeugt eine .h5-Datei, die nur die Gewichtsinformationen enthält.[246] Diese können dann in ein Modell geladen werden. Mithilfe der Layer-Namen können Inhalte – Gewichte – selektiert werden.
- `keras.callbacks.ModelCheckpoint(...)` Damit kann das Modell z. B. nach jeder Epoche gespeichert werden – die Funktion ruft dann entweder model.save_weights() oder model.save() auf.[247]

---

240 Vgl. https://keras.io (Stand: 22.02.2021).

241 Vgl. *TensorFlow-Keras*-Dokumentation, https://www.tensorflow.org/guide/keras/save_and_serialize?hl=en (Stand: 22.02.2021), bzw. *Keras*-API-Dokumentation, https://keras.io/api/models/model_saving_apis/ (Stand: 22.02.2021).

242 Funktionen, die errechnen, wie weit das Modell vom gewünschten Zielwert – vorgegeben z. B. durch die Labels der Trainingsdaten – entfernt ist, vgl. auch § 2 B.III.7..

243 Der „Optimierer“ ist eine Funktionalität, die zwischen Verlustfunktion und Modell eine Rückkopplung herstellt und die Parameter des Modells entsprechend anpasst.

244 Vgl. *Keras*-Dokumentation, https://keras.io/api/models/model_saving_apis/#save-method (Stand: 22.02.2021).

245 Vgl. *Keras*-Dokumentation, https://keras.io/api/models/model_saving_apis/#tojson-method (Stand: 22.02.2021).

246 Vgl. https://keras.io/api/models/model_saving_apis/#saveweights-method (Stand: 22.02.2021).

247 Vgl. *TensorFlow-Keras*-Dokumentation, https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint (Stand: 22.02.2021).

– `keras.utils.plot_model(model, ...)` Konvertiert ein *Keras*-Modell in ein Format, das eine grafische Ausgabe der Architektur bzw. der Hyperparameter des Modells erzeugt.[248]

Dem „trainierten Modell“ im Sinne dieser Arbeit entspricht in *Keras* das Ergebnis von `model.save(<Dateipfad>)`, da hier alle Informationen, die zur Ausführung des Modells erforderlich sind, abgelegt werden.

3. „Trainiertes Modell“ in *PyTorch*

Auf der *PyTorch*-Website wird *PyTorch* beschrieben als ein „Open Source Machine Learning Framework, das den Weg vom Research Prototyping zum Production Deployment beschleunigt“,[249] also von der Entwicklung bis zum Einsatz des fertig trainierten Modells in der Zielumgebung verwendet werden kann.

Auch in *PyTorch* gibt es unterschiedliche Möglichkeiten, Trainingsergebnisse bzw. ML-Modelle zu speichern:

– `torch.save(the_model.state_dict(), PATH)` speichert und lädt nur die Modellparameter.[250]
– `torch.save(the_model, PATH)` speichert und lädt das gesamte Modell als ein Objekt, dabei wird ein Tool verwendet, das sich „Pickle“ nennt.[251]

248 Vgl. *TensorFlow-Keras*-Dokumentation, https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model (Stand: 22.02.2021).
249 https://pytorch.org/ (Stand: 22.02.2021)
250 *PyTorch*-Dokumentation, https://pytorch.org/tutorials/beginner/saving_loading_models.html (Stand: 22.02.2021).
251 Das Pickle-Modul in *Python* implementiert binäre Protokolle für die Serialisierung und Deserialisierung von *Python*-Objektstrukturen. „Pickling“ ist der Vorgang durch den eine *Python*-Objekthierarchie in einen Byte-Stream umgewandelt wird, und „unpickling“ ist der umgekehrte Vorgang, bei dem ein Byte-Stream zurück in eine Objekthierarchie gewandelt wird. Der Dateiinhalt ist nicht „menschenlesbar“, sondern nur für das Laden in ein Programm vorgesehen; https://docs.python.org/3/library/pickle.html (Stand: 22.02.2021).

– `torch.nn.Module.load_state_dict()` Lädt die Parameter eines Modells in Form eines Dictionaries[252] unter Verwendung eines deserialisierten (also „unpickled“) `state_dict`-Objektes.[253]

Ein `state_dict` ist ein *Python* Dictionary-Objekt, das die Beziehungen jeder Schicht des Modells zu ihrem Parameter-Tensor enthält.

Optimiererobjekte haben ebenfalls ein `state_dict`, das Informationen über den Zustand des Optimierers enthält, sowie die verwendeten Hyperparameter.[254]

Das Ergebnis eines Aufrufs von `torch.save(the_model, PATH)` entspricht dem, was hier als „trainiertes Modell“ verstanden wird, denn die anderen beiden Funktionen speichern im Gegensatz dazu nur einzelne Bestandteile des Modells und sind im Ergebnis nicht ausreichend, um das Modell ohne weitere Informationen über die Architektur wiederherzustellen.

4. „Trainiertes Modell“ in *Scikit-Learn*

Scikit-Learn stellt simple und effiziente Tools für Data Mining und Datenanalyse zur Verfügung, die frei zugänglich und unter der Open Source BSD-Lizenz verfügbar sind.[255] Bezüglich der Speicherung eines Modells verweist die Dokumentation von *Scikit-Learn* zum einen auf die Pickle-Funktionalität von *Python*, und erwähnt zum anderen die Möglichkeit, das erstellte und trainierte Modell in andere Formate zu exportieren.[256] Aus der exportierten Form kann das Modell nicht weiter trainiert, sondern „nur noch“ für Vorhersagen – also den Produktiveinsatz – verwendet werden.[257]

---

252 Ein Dictionary-Objekt ist ein Objekt, das wie ein Nachschlagewerk funktioniert: Unter Schlüsselwörtern (keys) sind Informationen abrufbar

253 *PyTorch*-Dokumentation, https://pytorch.org/tutorials/beginner/saving_loading_models.html#save-load-state-dict-recommended (Stand: 22.02.2021).

254 Output der `state_dict`-Datei, insb. des Optimizer state dict einsehbar auf der Website https://pytorch.org/tutorials/beginner/saving_loading_models.html, zuletzt abgerufen am 22.02.2021.

255 Vgl. *Scikit-Learn*-Dokumentation, https://scikit-learn.org/stable/index.html (Stand: 22.02.2021).

256 Vgl. *Scikit-Learn*-Dokumentation, https://scikit-learn.org/stable/modules/model_persistence.html (Stand: 22.02.2021).

257 Vgl. *Scikit-Learn*-Dokumentation, https://scikit-learn.org/stable/modules/model_persistence.html (Stand: 22.02.2021).

Die Autoren von *Scikit-Learn* beschreiben den Prozess, ein mit zukünftigen Versionen des Frameworks zuverlässig wiederherstellbares Modell zu speichern: Es ist nicht nur das Modell zu speichern, sondern auch die Metadaten (hier: die Trainingsdaten, der *Python*-Quellcode mit dem das Modell erstellt wurde, die *Scikit-Learn*-Version und eventuelle Abhängigkeiten zu anderen Tools, sowie der Kreuzvalidierungswert der mit den Trainingsdaten erzielt wurde, damit geprüft werden kann, ob das Modell erfolgreich wiederhergestellt wurde).[258]

Das Speichern eines „trainierten Modells" im Sinne dieser Arbeit erfolgt also wie schon in *PyTorch* mithilfe des Pickle-Moduls, eine *Scikit-Learn*-spezifische Methode muss dafür nicht zum Einsatz kommen, sondern es werden die „Bordmittel" von *Python* eingesetzt.

III. Zusammenfassung und Definition

Ziel des Trainings eines ML-Modells ist es, ein Softwaregebilde zu schaffen, das reproduzierbar und zuverlässig für den gleichen Input die gleichen Vorhersagen bzw. Ergebnisse (Output) liefert. Dies kann nur erreicht werden, indem die während des Trainings noch manipulierbare Struktur des Modells am Ende des Trainings eingefroren und wiederabrufbar gemacht wird. Dazu unerlässlich sind

- Hyperparameter,
- Parameter, und
- Quellcode.

Die Hyperparameter sind erforderlich, weil sie das „Gerüst" des Modells beschreiben. Die Parameter sind erforderlich, weil sie das Gerüst des Modells mit Werten füllen. Der passende Quellcode führt alles zusammen.

Ein trainiertes ML-Modell liegt also vor, wenn eine Dateistruktur existiert, die von einem Quellcode oder Script verwendet wird, wobei das daraus entstehende Modellobjekt ohne weitere Zwischenschritte zur bei Entwick-

258 Vgl. *Scikit-Learn*-Dokumentation, https://scikit-learn.org/stable/modules/model_persistence.html (Stand: 22.02.2021).

Technologie: / Funktion speichert:	Keras	PyTorch	TensorFlow	Scikit-learn
Hyperparameter	model.to_json() / model.to_yaml()	z.B. torch.save({,optimizer_state_dict': optimizer.state_dict()}, PATH) oder optimizer.state_dict() als JSON speichern	export_meta_graph() – die Hyperparameter können als Collection aus dem importierten Objekt abgerufen werden Dateiendung also .meta	Evtl. mit Pickle leeres Objekt speichern
Parameter	model.save_weights() → .h5-Datei	torch.save(the_model.state_dict(), PATH) speichert und lädt nur die Parameter	Saver.save() → *.data-Datei	Speichern der Parameter mit JSON
Alles	model.save(filepath) → .h5-Datei **Inhalt:** - Architektur d. Modells - Gewichte des Modells - Trainingskonfig. (Loss, Optimizer) - Zustand Optimierer keras.utils.plot_model(...)	torch.save(the_model, PATH) speichert und lädt das gesamte Modell als ein serialisiertes Objekt (Pickle)	SavedModel / SavedModelBuilder → speichert alles in einer .pb-Datei und dazugehörigen Ordnern (Parameter separat) Saver.save() erzeugt 4 Dateien: - *.meta (Graph-Struktur) - *.data (Parameter) - *.index - Checkpoint-Datei	joblib.dump(...) / pickle.dump(...)

Abbildung 6.3: Überblick über Speichervarianten in den untersuchten Frameworks, eigene Darstellung.

lung des Modells angestrebten Problemlösung eingesetzt werden kann.[259] Grundsätzlich ist also immer ein aufrufender Quellcode bzw. ein aufrufendes Skript erforderlich, sowie eine oder mehrere Dateien, die Informationen über das Modell enthalten. Der Grad zwischen Quellcode bzw. Skript und Datei, auf dem die Informationen verteilt sind, ist je nach verwendetem Framework unterschiedlich. Für einen abschließenden Überblick vgl. Abbildung 6.3.

E. Trainierter Random Forest in R

Da sich für ein *Random Forest*-Modell, das nicht in *Python*, sondern in *R* konzipiert und trainiert wurde,[260] Unterschiede im Vergleich zur Implementierung in *Python* ergeben, erfolgt hier eine differenzierte Darstellung.

259 Theoretisch wäre es auch denkbar, ein trainiertes Modell zu erzeugen, das auch ohne Ablage in Dateien im Arbeitsspeicher eines Rechners mit endloser Runtime für immer existiert. Ein solches, doch eher fernliegendes, Konstrukt soll jedoch hier von der Definition nicht erfasst sein.

260 Die Verwendung der Programmiersprache *R* scheint für Random Forests nahezu so verbreitet wie *Python*, und soll deshalb hier auch Berücksichtigung finden.

Ein trainiertes Random Forest-Modell enthält eine Vielzahl von ungleichen Entscheidungsbäumen.[261] Jeder Entscheidungsbaum besteht aus einer Vielzahl an Knoten, die jeweils ein Feature hinsichtlich eines Schwellwertes überprüfen. „Trainiert" werden die Schwellwerte, anhand derer entschieden wird, ob ein Datensatz zum linken oder rechten Tochterknoten weitergeleitet wird. Im Ergebnis entsteht eine als Tabelle abrufbare Datensammlung, die – in der Implementierung des Pakets „randomForest" in der Programmiersprache *R* – für jeden konstruierten Baum in sechs Spalten die Baumstruktur enthält[262].

Folgende sechs Spalten entstehen dabei:

- Linker Tochterknoten
- Rechter Tochterknoten
- Gewählte Split-Variable bzw. Feature
- Split-Point bzw. Schwellenwert
- Status: handelt es sich um einen Endknoten?
- Vorhersage (nur für Endknoten relevant)

Diese Spalten enthalten alle Informationen, die zur Durchführung der Vorhersagen für neue Daten erforderlich sind. Für den Produktiveinsatz eines Random Forest in *R* ist folglich der Zugriff auf die so entstandene Tabelle ausreichend, diese stellt für den weiteren Verlauf der Prüfung der Schutzmöglichkeiten von mit *R* entwickelten Random Forest-Modellen den Schutzgegenstand dar.

F. Trainierte Parameter

Ein zentrales Element von ML-Modellen sind die Werte, die vielfach „Gewichte", „Gewichtungsinformationen" oder „Trainingsergebnisse" genannt werden. In den einschlägigen Frameworkbeschreibungen ist hingegen ne-

261 Für eine ausführlichere Erklärung vgl. § 2 B.II.1..
262 *Liaw/Wiener*, R News 2 Nr. 3 2002, S. 4.

ben „Weights“[263] auch von Parametern,[264] bzw. trainierbaren Parametern[265] die Rede. Diese werden stets kontrastiert mit „Hyperparametern“,[266] oder auch „System-“ oder „Metaparametern“.[267] Es handelt sich um Werte, die sich in einem Modell-Optimierungsprozess („Training“) nicht (automatisch) verändern.[268]

Trainierte Parameter, die aus dem Training eines Modells mit sorgfältig ausgewählten Trainingsdaten und einer entsprechenden Architektur hervorgehen, füllen das Modell mit Inhalt. Ohne die richtigen Parameter kann ein Modell nicht die richtigen Vorhersagen treffen. Es liegt daher nahe, auch die Sammlung der Parameterwerte als Schutzgegenstand ins Auge zu fassen.

G. Hyperparameter

Als Schutzgegenstand diskutiert werden auch die Hyperparameter, auch „Topologie des Netzes“[269] genannt. Wie bereits erläutert,[270] hat die korrekte Auswahl der Anzahl zu verwendender Schichten, der Menge Neuronen je Schicht, verwendeter Funktionen und Rückkopplungen innerhalb der Struktur einen wesentlichen Einfluss darauf, wie gut das KNN gestellte Aufgaben lösen kann.[271] Hyperparameter sind außerdem nicht nur für KNNs relevant, sondern für verschiedenste ML-Modelle – auch die Struktur von *Random Forests* wird unter anderem die maximale Baumtiefe, die Anzahl ausgewerteter Features etc. durch Hyperparameter vorgegeben. Eine sorgfältige Auswahl der passendsten Hyperparameter verleiht einem Modell die Struktur, die es braucht, um taugliche Ergebnisse zu produzieren. Die Hyperparame-

263 Weights: *Keras*, https://keras.io/api/models/model_saving_apis/ (Stand: 22.02.2021), *TensorFlow*, https://www.tensorflow.org/guide/saved_model?hl=en (Stand: 22.02.2021).

264 *TensorFlow*, https://www.tensorflow.org/guide/checkpoint?hl=en (Stand: 22.02.2021), *PyTorch* https://pytorch.org/tutorials/beginner/saving_loading_models.html (Stand: 22.02.2021).

265 *PyTorch*, https://pytorch.org/tutorials/beginner/saving_loading_models.html#what-is-a-state-dict (Stand: 22.02.2021).

266 Vgl. z. B. *Osinga*, Deep Learning Cookbook, S. 21; *Nielsen*, Neural Networks and Deep Learning, Kap. 1.

267 *Ertel*, Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung, S. 304.

268 Vgl. zur Begriffsklärung auch schon oben § 2 B.III.6..

269 *Hartmann/Prinz*, WRP 12 2018, 1431, 1434.

270 S. oben § 2 B.II.2..

271 Vgl. auch *Nielsen*, Neural Networks and Deep Learning, Kapitel „Implementing our network to classify digits“.

ter werden mitunter immer weiter angepasst, sodass sie das Ergebnis eines langwierigen Entwicklungsprozesses darstellen können. Dementsprechend könnten auch die vom Entwickler gewählten Hyperparameter urheberrechtlichen Schutzes bedürfen, und sind im weiteren Verlauf als eigenständiger Schutzgegenstand aufzufassen.

H. Zusammenfassung

Ziel dieses Kapitels war es, Klarheit zu schaffen in Bezug auf die Bestandteile von ML-Modellen, um unmissverständliche Schutzgegenstände zu identifizieren, die im nächsten Kapitel auf ihre Schutzfähigkeit untersucht werden können. Dafür wurden zunächst die Grundbausteine für ML-Modelle identifiziert, für die sich keine (urheberrechtlichen) Besonderheiten gegenüber den Grundbausteinen anderer Software ergeben.

Sodann wurde der Begriff des „trainierten Modells“ untersucht, und was darunter je nach eingesetzter Technologie zu verstehen ist. Aus den Erkenntnissen wurde eine die Gemeinsamkeiten der untersuchten Technologien hervorhebende allgemeine Definition für den Begriff des „trainierten Modells“ aufgestellt: Als „trainiertes Modell“ versteht diese Arbeit eine Dateistruktur (also permanent gespeicherte Informationen), die von einem Quellcode (*Quellcode 2* in Abbildung 6.2) verwendet wird, wobei das daraus entstehende Modellobjekt ohne weitere Zwischenschritte zur bei Entwicklung des Modells angestrebten Problemlösung eingesetzt werden kann. Neben dem Objekt „trainiertes Modell“ wurden zudem die einzelnen Bestandteile Quellcode, Hyperparameter und Parameter der Modelle als mögliche unabhängige Schutzgegenstände identifiziert. Für in der Sprache *R* entwickelte Random Forests bzw. Entscheidungsbaummodelle ist wesentlicher Schutzgegenstand die durch das Training ermittelte bzw. ermittelbare Tabelle.

§ 7 Schutzmöglichkeiten im Urheberrecht und den verwandten Schutzrechten

A. *Einleitung*

I. Forschungsstand

Bisher wird Schutz für ML-Modelle insbesondere unter dem Dach des Computerprogrammschutzes diskutiert,[272] teilweise wird dies auch strikt abgelehnt[273] oder es erfolgt eine differenzierte[274] Betrachtung. Andere diskutieren zwar eine Einordnung als Datenbank (insbesondere im Lichte des Leistungsschutzrechtes), kommen hier jedoch entweder zu keinem einen Schutz bejahenden Ergebnis,[275] oder wagen den Versuch einer Subsumtion erst gar nicht[276]. Wieder andere ziehen sogar den Tonträgerschutz als Möglichkeit heran, Modellen doch noch zu einem Schutz zu verhelfen, wenngleich unklar ist, wie dies in der Praxis auszusehen hätte.[277]

II. Hier gewählter Lösungsansatz

An dieser Stelle erfolgt daher unter Berücksichtigung der zuvor dargestellten technischen Gegebenheiten ein Perspektivwechsel, der Klarheit hinsichtlich der Schutzmöglichkeiten schaffen soll. Berücksichtigt wird neben den im

272 *Iglesias Portela/Shamuilia/Anderberg*, Intellectual Property and Artificial Intelligence: A Literature Review: EUR 30017 EN, S. 9; *Linke*, GRUR Junge Wissenschaft 2019, S. 47.

273 Sehr verallgemeinernd z. B. Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 21; KI auf Algorithmen reduzierend *Hauck/Cevc*, ZGE 11 2019, 135, 159; nur sehr eingeschränkt den Datenbankschutz für möglich haltend *Apel/Kaulartz*, RDi Nr.1 2020, 24, 28 f..

274 *Gomille*, JZ Nr. 20 2019, 969, 970; *Ehinger/Stiemerling*, CR 12 2018, 761, 765 Rn. 41; *Hartmann/Prinz*, WRP 12 2018, 1431, 1436; BT-Drs. 19/23700 S. 69 (Bericht der Enquete-Kommission KI).

275 *Hauck/Cevc*, ZGE 11 2019, 135, 161 f.; *Hartmann/Prinz*, WRP 12 2018, 1431, 1437; Loewenheim–*Leistner/Zurth*, Handbuch Urheberrecht, § 49 Rn. 146; *Haberstumpf*, GRUR 2003, 14, 19.

276 *Kaulartz/Braegelmann*, Rechtshandbuch Artificial Intelligence, Kap. 7.1 Rn. 24 ff..

277 Nägele/Apel in Dies., Rechtshandbuch Artificial Intelligence, Kap. 7.1 Rn. 50.

Urheberrecht infrage kommenden Vorschriften über den Schutz als Datenbankwerk (§ 4 Abs. 2 UrhG), als Computerprogramm (§ 69a UrhG) und als sonstiges Werk (§ 2 UrhG) auch ein Schutz durch das Datenbankherstellerrecht (§§ 87a ff. UrhG). Ermittelt wird ein Schutz für die folgenden zuvor erläuterten Schutzgegenstände: trainiertes ML-Modell in *Python* (insb. trainiertes KNN), trainierter Random Forest in *R*, trainierte Parameter, Hyperparameter, sowie das untrainierte Modell (in Kombination aus Hyperparametern und Parametern). Dabei wird jeder Schutzgegenstand für sich genommen anhand der infrage kommenden Vorschriften geprüft, und zwar das trainierte Modell in *Python* in § 7 B., der trainierte Random Forest in § 7 C., die trainierten Parameter in § 7 D., die Hyperparameter in § 7 E. und das untrainierte Modell in § 7 F..

B. Trainiertes ML-Modell in Python *(insb. trainiertes KNN)*

Aus den vorangegangenen Erläuterungen[278] geht hervor, dass der Entwickler eines ML-Modells entscheiden kann, was am Ende des Trainingsvorgangs in einer Datei abgelegt wird. In einem nächsten Schritt gilt es nun, herauszufinden, ob die so abgelegten Modellbestandteile einem urheberrechtlichen Schutz zugänglich sind.

I. Schutz als Datenbankwerk gem. § 4 Abs. 2 UrhG

Um einen urheberrechtlichen Schutz im Sinne von § 4 Abs. 2 UrhG zu genießen, müsste eine Datenbank folgende Voraussetzungen erfüllen: Erforderlich ist ein Datenbankwerk in Form einer Sammlung voneinander unabhängiger Elemente, die systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind. Ferner muss die Sammlung aufgrund der Auswahl oder Anordnung eine persönliche geistige Schöpfung darstellen.

278 S. oben § 6 D..

1. Datenbankwerk

Gemäß der Legaldefinition in § 4 Abs. 2 UrhG ist ein Datenbankwerk ein Sammelwerk, dessen Elemente systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind. Sammelwerk wiederum ist eine Sammlung von Werken, Daten oder anderen unabhängigen Elementen, die aufgrund der Auswahl oder Anordnung der Elemente eine persönliche geistige Schöpfung darstellen (§ 4 Abs. 1 UrhG). Zunächst ist also zu klären, ob ein trainiertes Netz die strukturellen Anforderungen an ein Datenbankwerk grundsätzlich erfüllt. Die Betrachtung hinsichtlich der Werkqualität des trainierten Netzes erfolgt in einem zweiten Schritt.

a) Sammlung

Ein trainiertes KNN müsste folglich eine Sammlung von Elementen darstellen. Elemente können gem. § 4 Abs. 1 UrhG Werke im Sinne von § 2 Abs. 2 UrhG, aber auch Daten und andere unabhängige Elemente sein.[279] Eine Sammlung ist eine Zusammenstellung mehrerer solcher Elemente. Den Elementen einer Sammlung muss kein Werkcharakter anhaften, es genügt, dass sie lediglich informationellen Wert aufweisen.[280]

Analog zum Vorgehen bei den eingangs präsentierten vier Frameworks wird erneut eine Untersuchung hinsichtlich der für die Sammlung infrage kommenden Elemente vorgenommen.

aa) *TensorFlow*

In *TensorFlow* könnte zum einen auf die durch jeden Aufruf von `Saver.save()` erzeugten vier Dateien abgestellt werden.[281] Es wird mithin nicht darauf rekurriert, dass – bzw. ob – ein einzelner Parameter bzw. eine einzelne „Gewichtungsinformation“ als Elemente anzusehen sind, sondern ob die genannten Dateien insgesamt Elemente einer Sammlung darstellen.

279 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 9; Wandtke/Bullinger–*Marquardt*, PK UrhR, § 4 Rn. 4.

280 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 9; Wandtke/Bullinger–*Marquardt*, PK UrhR, § 4 Rn. 4.

281 Vgl. Ausführungen in § 6 D.II.1..

In *TensorFlow* gibt es darüber hinaus die Möglichkeit, mit der `SavedModel`-Variante alle Informationen gesammelt in einer Protocol Buffer-Datei zu speichern (*.pb / *.pbtxt).[282] In beiden Fällen wären die Elemente jeweils zumindest die Graph-Struktur (also Hyperparameter) und die Parameter (Gewichtungsinformationen), für Variante 1 kommen noch der Checkpoint-Index sowie eine Liste der erstellten Checkpoints hinzu, die Sammlungen bestünden entweder in der durch `Saver.save()` erzeugten Zusammenstellung oder aber der Protocol Buffer-Datei.

bb) *Keras*

Der Sammlung entspricht in *Keras* die mit `model.save(Dateipfad)` erstellte .h5-Datei. In der Datei liegen sämtliche Informationen vor, die dazu benötigt werden, das trainierte Netz wiederherzustellen. Elemente der Sammlung sind die Architektur des Modells (die dazugehörigen Hyperparameter), die Parameter (die „Gewichte“), die Trainingskonfiguration sowie der Zustand des Optimierers.[283] Auch hier wird nicht auf die einzelnen Gewichtsinformationen als Elemente abgestellt, sondern auf deren Sammlung in der .h5-Datei.

cc) *PyTorch*

PyTorch stellt mit `torch.save(...)` eine Funktion bereit, mit der das ganze KNN-Modell in serialisierter Form in einer Datei abgelegt wird. Dabei wird das *Python*-Tool „Pickle“[284] eingesetzt. Dies hat allerdings auch zur Folge, dass etwa die Modellklasse[285] nicht „gepicklet“ wird, sondern nur eine Referenz zu der Datei, in der die Klasse definiert ist. Diese darf nicht verändert werden, wenn das Modell erfolgreich wieder geladen werden soll. Die Elemente entsprechen auch hier wieder den verschiedenen Hyperparametern und Parametern, und der Untersuchungsgegenstand „Sammlung“ ist

282 S. oben § 6 D.II.1..
283 Vgl. § 6 D.II.2..
284 Name abgeleitet aus dem Englischen *to pickle* – konservieren / einlegen, vgl. „pickle“, Merriam-Webster.com, 2019, https://www.merriam-webster.com, zuletzt abgerufen am 22.02.2021.
285 Also die abstrakte Definition der Modellart – vgl. zur Erklärung des Begriffs der „Klasse“ § 6 A.III..

die durch Ausführung der Funktion `torch.save(...)` bewirkte Zusammenstellung dieser Elemente.

dd) *Scikit-Learn*

Unter Zuhilfenahme von `joblib.dump(...)` oder `pickle.dump(...)` kann das gesamte mit *Scikit-Learn* erstellte Modell gespeichert werden. Das Modell – in *Scikit-Learn* repräsentiert als ein einzelnes Objekt – enthält in den ihm zugewiesenen Eigenschaften die Hyperparameter und Parameter, mithin die Elemente des trainierten KNN als Datensammlung.

ee) Zusammenfassung und Subsumtion

Entgegen der andernorts vorgenommenen Einschränkung auf die Gewichtungsinformationen eines KNN[286] werden diese hier explizit nur als ein Element der Sammlung verstanden. Die Sammlung „trainiertes Modell" besteht vielmehr aus den folgenden Elementen:

- Architektur des Modells (repräsentiert durch eine Kombination von Hyperparametern – Anzahl Schichten, Anzahl Neuronen je Schicht etc.),
- sonstige Hyperparameter (etwa die Aktivierungsfunktionen)
- sowie Parameter (als ein Element).

Fraglich erscheint, ob auch der Quellcode oder ein Skript zur Initialisierung des Netzes als Bestandteil der Sammlung eingeordnet werden müsste. Dieser ist jedoch erforderlich, um auf die Daten überhaupt zugreifen zu können. Ihn als Bestandteil der Sammlung zu verstehen, wäre mithin widersprüchlich, und ist gem. § 4 Abs. 2 S. 2 UrhG außerdem ausgeschlossen. Der Quellcode wird hier folglich aus der Begriffsdefinition des trainierten Netzes herausgenommen. Dennoch kann dieser freilich unabhängig davon einem Computerprogrammschutz gem. § 69a UrhG zugänglich sein.[287]

Funktionen, die lediglich zum Training des KNN erforderlich sind, gehören ebenfalls nicht zur Sammlung „trainiertes Modell". Nur dann, wenn das KNN zur Weiterentwicklung bereitgestellt werden sollte, könnte es sinnvoll sein,

286 Vgl. etwa *Ehinger/Stiemerling*, CR 12 2018, 768 f..
287 Vgl. dazu aa).

auch diese Hyperparameter in die Sammlung aufzunehmen. Gleiches gilt dann entsprechend auch für den Quellcode.

b) Unabhängige Elemente

Fraglich erscheint, ob den einzelnen Elementen der Sammlung ein von den anderen Elementen unabhängiger Informationsgehalt zukommen muss. Dass sie selbst nicht auch Werke sein müssen, geht aus § 4 Abs. 1 S. 1 UrhG hervor, der explizit von „Werken", „Daten" und „anderen unabhängigen Elementen" spricht.

Nach dem Wortlaut des § 4 Abs. 1 S. 1 UrhG müssen sämtliche Elemente jedenfalls „unabhängig" sein, wobei zu klären ist, welche Qualität diese Unabhängigkeit aufweisen muss.

Grundsätzlich sind die Elemente einer Sammlung unabhängig, wenn sie sich trennen lassen, ohne dass der Wert ihres informativen, literarischen, künstlerischen, musikalischen oder sonstigen Inhalts dadurch beeinträchtigt wird. [288] Dieses Kriterium soll unter anderem verhindern, dass etwa Musikstücke als Sammelwerk ihrer Töne, oder ein Buch als Sammelwerk der Buchstaben geschützt würde.[289] In diesen Fällen gewinnen die einzelnen Bestandteile ihren Sinn erst aus dem Kontext mit den anderen Elementen: Ein einzelner Ton etwa ergibt für den Hörer noch keinen „Sinn", im Gegensatz zum Abspielen einer Phrase oder des gesamten Stücks (ggf. wäre zu untersuchen, ob sich diese Argumentation auch hält, wenn ein umfassendes Musikwerk – etwa eine Sinfonie – mit ihren einzelnen Passagen betrachtet wird, sodass die Bestandteile nicht einzelne Töne, sondern etwa musikalische Motive wären – auch Kombinationen können zumindest nach europäischer Rechtsprechung Elemente darstellen [290]). Die Elemente dürfen also nicht erst aus der Gesamtschau, sondern müssen auch alleinstehend Sinn ergeben,[291] jedoch kann der Sinn auch durch ein „formales Anordnungsprinzip erzeugt"[292]

288 EuGH GRUR 2005, 254, 255, Rn. 29 – *Fixtures-Fußballspielpläne II*; Schricker/Loewenheim–*Leistner*, Urheberrecht, § 4 Rn. 18.

289 Nordemann/Fromm–*Czychowski*, UrhR, § 4 Rn. 24; Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 10.

290 EuGH GRUR 2005, 254, 255, Rn. 35 – *Fixtures-Fußballspielpläne II*; EuGH MMR 2016, 51, 52, Rn. 20 f. – *Verlag Esterbauer*.

291 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 10; Schricker/Loewenheim–*Leistner*, Urheberrecht, § 4 Rn. 18.

292 Nordemann/Fromm–*Czychowski*, UrhR, § 4 Rn. 28.

werden. Als Beispiel wird etwa eine Sammlung von Postleitzahlen herangezogen:[293] eine einzelne fünfstellige Zahlenfolge, wie etwa 70839, ergibt für sich genommen wenig Sinn. Erst das Wissen darüber, dass es sich um eine Postleitzahl handelt, eröffnet Informationen darüber, dass etwa bestimmte Straßen diesem Postleitzahlenbereich zugeordnet sein könnten.

Bezüglich des Informationsgehaltes eines Elements ist darüber hinaus nicht maßgeblich, welcher Informationsgehalt dem Element nach der Zweckbestimmung der Sammlung zukäme, sondern jeder denkbare Informationswert.[294]

Das Abrufen einzelner, wie oben definierter Elemente (abgestellt wird nicht nur auf die trainierten Parameter, sondern auch auf die Hyperparameter, vgl. ee)) aus einem KNN liefert Informationen über die verwendeten Aktivierungsfunktionen, die Anzahl der Schichten und alle weiteren Informationen, die in der Sammlung „trainiertes Modell" enthalten sein können. Jedes einzelne Element trifft dabei eine Aussage über das betreffende Modell. Diese Elemente haben also jeweils einen eigenen Aussage- bzw. Informationsgehalt und sind mithin unabhängig.[295]

Fraglich könnte sein, ob die Elemente einer Datenbank gleichartig sein müssen: Die Elemente der Sammlung „trainiertes Modell" können nicht nur Zahlenwerte sein („Anzahl Schichten", „Anzahl Neuronen" etc.) sondern auch Funktionsnamen (Art der Aktivierungsfunktion, Kostenfunktion etc.) und auch Zusammenstellungen vieler Zahlenwerte in einem Element (Parameter). Denkbar wäre zu argumentieren, es handele sich bei allen Informationen zu einem einzigen trainierten KNN um lediglich *einen* Datensatz, und nur bei einer Zusammenstellung mehrerer trainierter Modelle könne von einer Sammlung gesprochen werden. Dem ist zu entgegnen, dass zwar in einer Briefmarkensammlung bestimmte Briefmarken gesammelt, und in einer Zeitschriftendatenbank bestimmte Zeitschriften erfasst sind, dass aber Sammlungen sich auch mit einem bestimmten Thema befassen können, das alle Elemente verbindet, so unterschiedlich diese im Einzelnen auch sein mögen, wie es etwa in Ausstellungen[296] üblicherweise der Fall ist. Ferner kann auch schon ein einzelnes topografisches Datenblatt eine Datenbank

293 Nordemann/Fromm–*Czychowski*, UrhR, § 4 Rn. 26.
294 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 18.
295 Die Unabhängigkeit „der Elemente" ablehnend *Apel/Kaulartz*, RDi Nr.1 2020, 24, 29, ohne jedoch die Elemente tiefergehend zu analysieren.
296 Vgl. z. B. LG München I ZUM-RD 2003, 492, 498 f. – *Jemen-Ausstellung*.

darstellen.[297] Die Verschiedenheit der Art der Elemente steht demzufolge hier einem Schutz nicht entgegen.

Die Sammlung „trainiertes Modell" besteht also aus unabhängigen Elementen im Sinne von § 4 Abs. 1 S. 1 UrhG.[298]

c) Systematische oder methodische Anordnung

Gem. § 4 Abs. 2 S. 1 UrhG müssten die Elemente nicht nur unabhängig, sondern auch systematisch oder methodisch angeordnet sein. Erforderlich ist also, dass der Anordnung ein gewisses System zugrunde liegt, das auch in einem Ordnungsschema oder einer Klassifizierung bestehen kann, oder die Anordnung müsste – methodisch – auf einer ordnenden Handlungsanweisung oder einem Plan basieren.[299]

Wenn die Elemente in einer für das erneute Einlesen vorgesehenen Datei gesammelt vorliegen, ist bereits evident, dass eine Struktur vorhanden sein muss: Sonst wäre ein Abrufen der Daten nicht möglich. Die Strukturierung innerhalb der Dateien ergibt sich entweder aus Indizes oder aus Schlüsselwörtern, mit denen die einzelnen Elemente adressiert werden können. Wenn mehrere Dateien vorliegen, kann sich die Anordnung auch aus der Ordner- bzw. Verzeichnisstruktur ergeben.

---

297 LG München I GRUR 2006, 225, 227 – *Topografische Kartenblätter*; Wandtke/Bullinger–*Hermes*, PK UrhR, § 87a Rn. 32.

298 Randbemerkung: auch wenn eine einzelne Gewichtungsinformation (die stets in einer Kombination aus die dazugehörigen Neuronen identifizierenden Indizes ausgegeben wird) abgerufen würde, informiert diese immerhin darüber, wie an einer bestimmten Stelle des KNN die Verbindung zwischen zwei Neuronen gewichtet ist (ähnlich einer Postleitzahl in einer Postleitzahlenliste) – ein einzelner Parameter – etwa als Zahl 0,42 – ergibt also zwar alleinstehend keinen tieferen Sinn. Wenn die Wahrnehmende allerdings weiß, dass es sich um einen Parameter eines KNN handelt, ist zumindest klar, dass in dem KNN an einer Stelle der Output eines Neurons mit dem Gewicht 0,42 als Input des darauffolgenden Neurons weitergeleitet wird, und könnte möglicherweise ebenso als unabhängiges Element angesehen werden. Dies kann vorliegend jedoch zunächst dahinstehen.

299 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 17, Wandtke/Bullinger–*Marquardt*, PK UrhR, § 4 Rn. 10.

d) Zugänglichkeit der Elemente

§ 4 Abs. 2 S. 1 UrhG fordert ferner, dass die Elemente einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind. Die Zugänglichkeit ist gegeben, wenn es möglich ist, auf die Elemente unter Berücksichtigung der Anordnungskriterien zuzugreifen und sie abfragen zu können.[300]

Jede Framework-Variante stellt Methoden bzw. Funktionen bereit, mithilfe derer die Elemente einzeln abgerufen werden können. Teils ist dieses frameworkübergreifend möglich, teilweise kann ein KNN nur genau so wieder geladen werden, wie es auch gespeichert wurde. Für alle gilt jedoch, dass der Vorgang des Abspeicherns darauf angelegt ist, die Daten wieder zu laden und zugreifbar zu machen. Erwähnung finden sollte an dieser Stelle noch, dass die KNNs ganz üblicherweise nicht als gepackte ausführbare Dateien (wie z. B. eine „Word.exe“, die nach einen Doppelklick das Programm startet) das Endergebnis des Trainingsvorgangs darstellen, sondern sie müssen – durch einen Aufruf in Code oder Skript – geladen werden und sind dann auch zugänglich und veränderbar.

e) Zwischenergebnis

An anderer Stelle wird die Datenbankqualität eines neuronalen Netzes kategorisch abgelehnt mit dem Argument, die „Daten der Verarbeitungsschicht“ – die Gewichte der Neuronen – interessierten den Endnutzer nicht, ferner sei „ihr Wert als solcher für ihn auch keine verwertbare Information“.[301] Hierbei wird jedoch übersehen, dass der „Endnutzer“, zum Beispiel also der Verwender des trainierten *DeepDream*-Modells,[302] nicht zwingend die hier zu betrachtende Zielgruppe darstellt.

Richtig ist, dass etwa die Verwenderin eines KNN zur Bildbearbeitung nicht an den einzelnen Neuronenwerten interessiert ist. Anders jedoch ist die Lage für den Entwickler, der ein vortrainiertes Netz – oder einzelne Schichten daraus – weiterverwenden möchte. Zudem wird verkannt, dass nicht nur die Neuronengewichte als Datenbankinhalt relevant sind, sondern die gesamte Netzwerkstruktur. Möglicherweise hat sich in den vergangenen 20 Jahren[303] jedoch auch der Stand der Technik derart verändert, dass die

300 Nordemann/Fromm–*Czychowski*, UrhR, § 4 Rn. 36.
301 *Grützmacher*, Datenbanken, S. 66; *Ehinger/Stiemerling*, CR 12 2018, 761, 769.
302 Vgl. § 10 C.I..
303 Die Aussagen von *Grützmacher* sind von 1999.

Sichtweise nun eine andere zu sein hat. Insbesondere werden heute wohl häufig vortrainierte Modelle verwendet, es sind zahlreiche für bestimmte Analysezwecke (wie etwa die Gesichtserkennung) spezialisierte Modelle verfügbar, die den individuellen Anforderungen angepasst werden können. Insofern ist auch nicht (mehr) nur auf den Endnutzer im Sinne eines nicht an der Entwicklung beteiligten Benutzers abzustellen, sondern vielmehr auf Entwickler, die sich das Modell heraussuchen, das für ihre Zwecke besonders gut geeignet ist. Und für diese kann es mitunter sehr interessant sein, die „Daten der Verarbeitungsschicht" in Augenschein zu nehmen. Zudem ermöglicht eine rückwirkende Analyse dieser Daten einen Einblick in die Arbeitsweise des verwendeten Modells, etwa zur Fehlersuche oder zur Vermeidung von unzutreffenden Vorhersagen, die auf schlecht gewählten Trainingsdaten basieren: Im Rahmen dieser „KI-Erkläransätze" (Stichwort: „Explainable AI") können auch einzelne bzw. Gruppen von Neuronenwerten interessieren.[304]

Nicht zuletzt ist auch der Telos des Datenbankwerkschutzes zu berücksichtigen: Es ist nicht das Ziel, anhand der Qualität der Elemente einer Sammlung einen neuen Schutz der enthaltenen Elemente zu konstruieren. Das Gegenteil ist der Fall: Der Fokus liegt auf dem, was durch Auswahl und Anordnung geschaffen wird, und was gerade mehr als die Summe seiner Teile sein kann. Eine denkbare Parallele zu dem ungewünschten Zerlegen eines Buches oder Musikwerkes in seine Einzelteile ergibt sich hier nicht, denn die untersuchten ML-Modelle sind gerade nicht als Ganzes anderweitig urheberrechtlichem Schutz unterstellt. Vielmehr kommt ihnen aus den dargelegten Ausführungen viel eher der Charakter einer Datenbank zu, die Informationen enthält, die schlussendlich von Computerprogrammcode eingelesen und für die Erzeugung neuer Bilder oder Tonfolgen oder der Zuordnung von Kategorien eingesetzt werden.

Wie gezeigt, kann – bei entsprechender Definition des Begriffs des trainierten KNN – grundsätzlich eine Datenbank vorliegen. Die Werkqualität im Sinne einer persönlichen geistigen Schöpfung ist jedoch gesondert zu prüfen.

304 Vgl. z. B. die sog. „Sentiment-Analysis" zur Erkennung negativer Formulierungen in Texten, die die Gewichte einer „Attention"-Schicht besonders hervorhebt, https://traversals.com/blog/explainable-ai-for-sentiment-analysis/ (Stand: 22.02.2021) Gewichte mit dem Wert 0 können u. a. darauf hinweisen, dass zu untersuchende Eigenschaften gar nicht vorlagen bzw. Pfade im Netz blockiert sind, *Ancona* et al., Gradient-Based Attribution Methods, S. 179.

2. Persönliche geistige Schöpfung

Spätestens bei der Anforderung der persönlichen geistigen Schöpfung scheitern bisherige Subsumtionsversuche,[305] die ihre Prüfung jedoch auf eine Sammlung von Parametern (Gewichtungsinformationen) beschränken. Möglicherweise ergibt sich ein anderes Ergebnis, wenn das wie hier beschriebene gesamte trainierte KNN als Untersuchungsgegenstand herangezogen wird.

Die Anforderung der Werkqualität für Datenbankwerke ergibt sich – wenn nicht schon aus dem Begriff – zumindest daraus, dass § 4 Abs. 2 UrhG Datenbankwerke als Sammelwerke mit erweiterten Eigenschaften definiert. Die Sammelwerke wiederum finden ihre Legaldefinition in § 4 Abs. 1 UrhG, und dort heißt es: „Sammelwerke sind Sammlungen von Werken, (…) die aufgrund der Auswahl oder Anordnung der Elemente eine persönliche geistige Schöpfung sind."

a) Persönliche oder eigene geistige Schöpfung?

In Art. 3 Abs. 1 Datenbank-RL ist hingegen von einer „eigenen geistigen Schöpfung" die Rede, sodass zu fragen ist, ob hierin ein Unterschied zur „persönlichen geistigen Schöpfung" besteht und worin dieser liegt. *Grützmacher*[306] war noch der Ansicht, dass hierin ein maßgeblicher Unterschied bestünde, inzwischen ist jedoch weitgehend geklärt, dass der deutsche Gesetzgeber es schlicht nicht für erforderlich hielt, den Wortlaut in § 4 UrhG anzupassen (anders als in § 69a Abs. 3 UrhG, in den die „eigene" geistige Schöpfung Eingang gefunden hat), mit der Begründung, dass bereits vor Erlass der Richtlinie keine erhöhten Anforderungen (erhöht im Sinne einer „persönlichen" gegenüber einer „eigenen" geistigen Schöpfung) gestellt wurden.[307] Inzwischen hat auch der EuGH in der *Football Dataco/Yahoo*-Entscheidung[308] die Anforderungen an den Datenbankschutz explizit in die-

305 *Ehinger/Stiemerling*, CR 12 2018, Rn. 62 lassen es letztendlich an der persönlichen geistigen Schöpfung scheitern, *Hartmann/Prinz*, WRP 12 2018, Rn. 62 steigen schon bei der Anforderung an die Unabhängigkeit der Elemente aus und prüfen die persönliche geistige Schöpfung nicht mehr.

306 *Grützmacher*, Datenbanken, S. 181 ff..

307 Ahlberg/Götting–*Ahlberg*, BeckOK-UrhG, § 4 Rn. 25; Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 11; BT-Drs. 13/7385.

308 EuGH GRUR 2012, 386, 387– *Football Dataco/Yahoo*.

sem Sinne geklärt. Fraglich ist also nur, welche Anforderungen diesbezüglich an die hier untersuchten elektronischen Datenbankwerke zu stellen sind.

b) Schöpfung in Auswahl oder Anordnung

Gem. §4 Abs. 2 UrhG i. V. m. § 4 Abs. 1 UrhG müssten die Anordnung oder Auswahl der Elemente die persönliche geistige Schöpfung begründen.

aa) Auswahl

Auswahl ist das Sichten, Sammeln, Bewerten und Zusammenstellen unter Berücksichtigung besonderer Auslesekriterien,[309] wobei ein Entscheidungsspielraum hinsichtlich der Auswahlmöglichkeit erforderlich ist, damit ein Datenbankwerk im Sinne einer geistigen Schöpfung entstehen kann.[310]

bb) Anordnung

Anordnung hingegen meint die Einteilung, Präsentation und Zugänglichmachung der ausgewählten Elemente nach einem oder mehreren Ordnungssystemen.[311]

cc) Anordnung für Datenbankwerke i. d. R. programmseitig vorgegeben

Naturgemäß ergibt sich die konkrete Anordnung der Elemente von Datenbankwerken aus den technischen Gegebenheiten und ist bei „klassischen" elektronischen Datenbanken überwiegend von der Datenbanksoftware vorgegeben.[312] Es wird daher auf das Ausgabeformat der Daten abgestellt, also darauf, wie die Daten in der Ausgabe angeordnet sind. Die Elemente müssten systematisch und methodisch zugänglich sein,[313] sodass es für die Anord-

309 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 11.
310 Wandtke/Bullinger–*Marquardt*, PK UrhR, § 4 Rn. 9.
311 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 11.
312 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 19.
313 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 19.

nung auf die Schöpfungshöhe der Ausgabe- und Verknüpfungsmöglichkeiten ankommt.[314]

Spätestens an dieser Stelle könnte fraglich erscheinen, wie das oben mühevoll konstruierte ML-Datenbank„werk" überhaupt mit einer elektronischen Datenbank, wie sie allgemein in der Vorstellung existiert, zu vergleichen sein könnte, entspricht eine Speicherung von Objekten im Binärformat doch nicht der klassischen Interpretation des Begriffes „Datenbank", bei dem Daten in einer Eingabemaske gesucht oder in Form eines Indexes dargestellt werden können.

Das Verständnis erleichtert ein Blick hinter die Kulisse einer solchen „klassischen" Datenbank: Auch diese legt Informationen gewöhnlicherweise in einer oder mehreren Dateien ab, die dann – bei Aufruf – eingelesen und etwa einem im Code definierten Tabellenobjekt zugewiesen werden (wenn nicht schon Tabellenobjekte in serialisierter Form abgelegt wurden – dann ist die Parallele noch offensichtlicher). Dieses Tabellenobjekt enthält dann Informationen unter anderem darüber, wie es heißt, wie viele Spalten es enthält, wie diese Spalten heißen, und was ihr Inhalt ist. Wie diese Datenbank vom Menschen wahrgenommen werden kann, hängt davon ab, wie sie zum Beispiel im Browser präsentiert wird.

Folglich könnte auch ein trainiertes KNN, das als Objekt oder anderweitig in Dateien abgelegt wurde, in einer Form wiedergegeben werden, die es vermutlich einfacher machen würde, darin eine Datenbank zu erkennen. Auch bei trainierten KNN hat der Entwickler aber bis auf die Auswahl der verwendeten Frameworks keinen oder wenig Einfluss darauf, wie die Elemente angeordnet werden, sodass es für die persönliche geistige Schöpfung auf die Auswahl der Elemente ankommen muss.

dd) Schöpfungsspielraum in der Auswahl

Hinsichtlich dieser Auswahl ist die Konzeption derselben entscheidend, nicht jedoch, dass der Urheber die zur Durchführung der Auswahl erforderlichen Schritte selbst vornimmt.[315]

---

314 Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 19, Schricker/Loewenheim–*Leistner*, UrhR, § 4 Rn. 34.

315 BGH GRUR 2007, 685 Rn. 19, 23 – *Gedichttitelliste I*; Dreier/Schulze–*Dreier*, UrhG, § 4 Rn. 19.

Der Ersteller eines trainierten KNN ist in der Regel nicht Urheber der Abfragemöglichkeiten (diese Funktionalität wird durch die Frameworks bereitgestellt und vorgegeben). Durch die Auswahl der Speichervariante und des Frameworks wird jedoch zumindest insoweit Einfluss auf die „Abfrage- und Verknüpfungsmöglichkeiten" genommen, als es bei den zur Wahl stehenden Frameworks Unterschiede in dieser Hinsicht gibt. Zudem könnte differenziert werden, ob der Entwickler die Default-Speichervariante wählt – dann würde die Auswahl der zu speichernden Elemente durch das Framework getroffen, was gegen eine persönliche geistige Schöpfung spräche – oder ob etwa Variablen gezielt benannt und zur Speicherung ausgewählt werden.

Allerdings kann für diese Prüfung durchaus auch auf einen noch früheren Zeitpunkt abgestellt werden: Der Entwickler wählt die Architektur des Netzes, die Funktionen, die Anzahl Trainingsdurchläufe etc. selbst aus, sodass sich das trainierte KNN als Ganzes als das Werk präsentiert, dessen Elemente einzeln abrufbar sind. Dem Argument, die Auswahlmöglichkeiten seien doch auf die infrage kommenden Werte für die Hyperparameter beschränkt, ist entgegenzuhalten, dass auch in der Musik der Tonraum im Wesentlichen auf 8 Töne (in verschiedenen Oktaven) beschränkt ist. Dennoch ergeben sich durch das Hinzuziehen von Rhythmus und Dynamik einzigartige Kombinationen, die als geistige Schöpfungen anerkannt sind. Insbesondere die Wahl der Hyperparameter hat wesentlichen Einfluss auf die Leistungsfähigkeit des Netzes und erfordert geistige Anstrengung und Kreativität, denn der Entwickler des KNN hat in der Regel eine konkrete Vorstellung davon, was das KNN leisten können soll, und nimmt aufwendige Anpassungen vor, die sich letztendlich in der Kombination aus Hyperparametern und Parametern niederschlagen. Mithin ist auch ein tagelanges Training eines KNN nutzlos, wenn die Hyperparameter nicht sorgfältigst gewählt wurden.

Solange der Entwickler sich also nicht Automatismen bedient, um Hyperparameter zu optimieren, sondern diese selbst wählt, dem KNN-Objekt zuweist, dieses „trainiert" und das gesamte Ergebnis der Reproduzierbarkeit halber speichert, spielt in einem trainierten KNN die Auswahl der Datenbankinhalte durch den Entwickler selbst eine derart bedeutende Rolle, dass beim Ergebnis i. d. R. von einer persönlichen geistigen Schöpfung ausgegangen werden kann. Selbstverständlich kann hier die Prüfung in Einzelfällen zu anderen Ergebnissen kommen, wenn etwa ein vortrainiertes Netz verwendet und angepasst wird, oder wenn der Beitrag des Urhebers nicht über die Reproduktion bekannter banaler Strukturen hinausgeht.

3. Ergebnis

Ein wie hier definiertes trainiertes KNN – ohne den aufrufenden Quellcode – erfüllt alle Anforderungen an ein Datenbankwerk und kann in der Auswahl der Elemente auch eine persönliche geistige Schöpfung darstellen, sodass es dem Schutz für Datenbankwerke gem. § 4 Abs. 2 UrhG grundsätzlich zugänglich ist.

4. Wer ist der Urheber?, oder: Schutzumfang und Folgen

Sobald die grundsätzliche Schutzfähigkeit festgestellt wurde, stellt sich unweigerlich die nächste Frage: wem gebührt der Schutz? Das Vorliegen einer persönlichen geistigen Schöpfung wurde bejaht aufgrund der Auswahl der Hyperparameter. Urheber ist folglich, wer die Netzwerkstruktur, Aktivierungsfunktion etc. festlegt (nicht, wer eine vorgegebene Architektur lediglich umsetzt). Zu überlegen wäre noch, inwiefern eine ggf. davon verschiedene Person, die die Trainingsdaten auswählt, unter Umständen als Miturheber anzusehen ist. Bei einer Personenmehrheit muss an der Stelle allerdings berücksichtigt werden, dass die Auswahl (und ggf. auch die Aufbereitung bzw. Vorbereitung) der Trainingsdaten höchstens einen Rahmen setzen kann. Die geistige Schöpfung liegt immer noch darin, die an die Trainingsdaten angepasste und die zur Erreichung der Aufgabenstellung optimale Kombination an Netzwerkeinstellungen auszuwählen.

Der Schutzumfang ist sorgfältig zu begrenzen, um ihn nicht in einen Ideenschutz ausufern zu lassen. Sinnvoll erscheint eine Begrenzung auf die konkrete Gestalt des trainierten KNN. Welche Gestalt dieses annehmen kann, wurde zu Beginn des Kapitels bereits dargestellt. Die beschriebenen Formen – in der Regel Computerdateien, eine einzelne oder mehrere – sind der Gegenstand, den es vor Vervielfältigung zu schützen gilt.

II. Investitionsschutz gem. §§ 87a ff. UrhG

Mit den Regelungen in §§ 87a ff. UrhG wird in Umsetzung der Datenbank-RL demjenigen, der eine Datenbank unter Aufwendung einer nach Art oder Umfang wesentlichen Investition erschafft, ein 15-jähriger sui-generis-Schutz gewährt. Dieser kann grundsätzlich auch neben einem urheberrechtlichen

Schutz aus § 4 Abs. 2 UrhG i. V. m. § 4 Abs. 1 UrhG bestehen.[316] Folglich sollen seine Voraussetzungen auch hier für trainierte KNN geprüft werden. Auch hier wird der aufrufende Quellcode wieder außer Acht gelassen. Zu prüfen ist, ob eine Datenbank sowie ein Investitionsgegenstand im Sinne des § 87a UrhG vorliegen und ob die Wesentlichkeit der Investition gegeben ist.

1. Datenbank

Während die Datenbank-RL einen einheitlichen Datenbankbegriff für den urheberrechtlichen ebenso wie den sui-generis-Schutz verwendet, umschreibt der deutsche Gesetzgeber Datenbanken in § 4 UrhG und § 87a UrhG jeweils unterschiedlich, die Voraussetzungen stimmen letztendlich jedoch überein.[317] Wenn eine Datenbank nach § 4 UrhG gegeben ist, kann also auch für § 87a UrhG vom Vorliegen einer Datenbank ausgegangen werden.

Im Rahmen der urheberrechtlichen Prüfung in e) wurde bereits festgestellt, dass ein trainiertes künstliches neuronales Netz Datenbankqualität aufweist, insofern erübrigt sich die Prüfung an dieser Stelle.

2. Investitionsgegenstand

Zusätzlich muss für § 87a UrhG im Rahmen der Beschaffung, Sammlung, Überprüfung oder Darstellung der Datenbankinhalte eine nach Art oder Umfang wesentliche Investition anfallen (§ 87a Abs. 1 S. 1 UrhG). Sowohl finanzielle Mittel als auch der Einsatz von Zeit, Arbeit und Energie können die Investitionen ausmachen.[318] Nicht berücksichtigt werden Kosten für die Erzeugung von Daten.[319]

Für die Identifizierung der Investition in die KNN-Entwicklung bzw. in die Erstellung eines KNN-Modells – also der Datenbank – ist es hilfreich, den Entstehungsprozess erneut unter die Lupe zu nehmen.

Der Entwickler wählt anhand der Aufgabenstellung und seiner Erfahrung initiale Hyperparameter aus, mithilfe derer das Modell trainiert wird. Am Ende des Trainingsvorgangs wird evaluiert, ob die Ergebnisse den Vorstellungen genügen. In diesem Zeitpunkt entsteht bereits eine erste Vorstufe

316 Dreier/Schulze–*Dreier*, UrhG, Vorbemerkung zu § 87a, Rn. 8.
317 Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 3.
318 ErwGr. 40 Datenbank-RL
319 Vgl. Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 13.

der Datenbank. Anschließend passt der Entwickler die Hyperparameter an und wiederholt den Trainingsvorgang, und wiederholt diesen Prozess, bis die Ergebnisse passen (z. B. bis die Genauigkeit einer bestimmten Mindestprozentzahl entspricht, bzw. bis der Entwickler den Eindruck hat, dass das Modell seine geistige Vorstellung der Problemlösung hinreichend abbildet). Insofern könnte dieser Vorgang als Datenbeschaffung oder Datensammlung (Daten sind hier nicht die Trainingsdaten, sondern eben die Elemente der entstehenden Datenbank) bezeichnet werden. Die eigentliche, persistente, nutzbare Datenbank entsteht erst, wenn der Entwickler sich dazu entschließt, die gesammelten Daten nicht zu verwerfen, indem zumindest Checkpoints[320] erstellt werden bzw. am Ende des Trainingsprozesses das Modell reproduzierbar im Speicher ablegt wird.

Es wird hier insbesondere nicht auf die Berechnung und Optimierung der Parameter abgestellt im Sinne einer Erzeugung neuer Daten. Die dafür entstehenden Kosten wären kein tauglicher Investitionsgegenstand. Vielmehr wird davon ausgegangen, dass die Zusammenhänge, die der Entwickler in den Daten vermutet, bereits bestehen und lediglich greifbar gemacht werden müssen, um daraus zum Beispiel Aussagen für die Zukunft treffen zu können. Die Muster in den Daten, mit denen die ML-Modelle arbeiten, werden nicht erst hergestellt, sondern aufgefunden.

3. Wesentlichkeit der Investition

Sodann ist zu klären, welche Kosten im Zusammenhang mit der Datenbankherstellung im Machine Learning-Kontext entstehen. Wird davon ausgegangen, dass lediglich Zusammenhänge zwischen bestehenden Daten ermittelt werden, so wären zumindest die Kosten für das „Ermittlungsprogramm“ – also den Algorithmus, bzw. die Entwicklung des Modells, sowie die erforderliche (Spezial-)Hardware (oder alternativ gemietete Online-Ressourcen) und das Gehalt für die das Modell trainierenden Data Scientists – zu berücksichtigen. Ferner dürften auch die Kosten für die Bereitstellung des Modells in Ansatz zu bringen sein. Auch die Beschaffung der Trainingsdaten – so

320 Checkpoints bezeichnen Speicherungen des Modells bzw. seiner Parameter, die die Wiederaufnahme des Trainings zu einem späteren Zeitpunkt ermöglichen, vgl. z. B. https://www.tensorflow.org/guide/checkpoint (Stand: 22.02.2021).

sie entgeltlich erfolgt – und damit auch der Aufwand für ihre Sammlung, dürften relevante Kosten darstellen.[321]

Anforderungen an das Merkmal der Wesentlichkeit, bzw. ab welchem Betrag, welchem Zeitaufwand oder welcher Qualität an Investition Wesentlichkeit gegeben ist, gehen weder aus § 87a UrhG noch aus der Datenbank-RL hervor.[322] Die Auslegung dieses unbestimmten Rechtsbegriffs ist vielmehr als „flexibles Kriterium"[323] der Rechtsprechung überlassen. Es liegt nahe, die Auslegung an dem Ziel der Datenbank-RL zu orientieren, und einen Schutz zu schaffen, der einen Anreiz für die Entwicklung solcher Speicher- und Verarbeitungssysteme bietet, und damit die Schwelle zur Wesentlichkeit nicht zu hoch anzusetzen,[324] sodass lediglich sog. „Allerweltsinvestitionen"[325] nicht erfasst sein sollen. Insbesondere sind keine Investitionen von substantiellem Gewicht vorausgesetzt.[326] Es sind also Einzelfallentscheidungen zur Auslegung des Wesentlichkeitskriteriums erforderlich.[327]

Es ist jedoch davon auszugehen, dass in Anbetracht des nicht unerheblichen finanziellen und zeitlichen Entwicklungsaufwands zumindest für komplexe, tiefere neuronale Netze die Wesentlichkeit der Investition gegeben sein dürfte. Im Zweifel muss es hier auf eine Einzelfallbetrachtung ankommen.

---

321 So wird etwa über die einzigartige Zusammenstellung von Informationen zu global erteilten Patenten von *IFSCLAIMS* berichtet, dass beispielsweise die Vereinheitlichung der Firmennamen besonders aufwendig sei, die im Rahmen der Standardisierung der Trainingsdaten vorzunehmen ist – so sei durch jahrelange Arbeit ein einzigartiges Datenset entstanden, vgl. https://www.cmswire.com/information-management/machine-learning-datasets-build-or-buy/ (Stand: 22.02.2021); auch der Zugang zu wertvollen Datensammlungen wie etwa die Inhalte von *Beck-Online* oder *Juris* erfordert in der Regel kostenpflichtige Abonnements.

322 Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 11; Wandtke/Bullinger–*Hermes*, PK UrhR, § 87a Rn. 52.

323 Wandtke/Bullinger–*Hermes*, PK UrhR, § 87a Rn. 52.

324 BGH GRUR 2011, 724, 725 – *Zweite Zahnarztmeinung II*; OLG Hamburg CR 2018, 22.

325 Wandtke/Bullinger–*Hermes*, PK UrhR, § 87a Rn. 54.

326 BGH GRUR 2011, 724, 725 – *Zweite Zahnarztmeinung II* Rn. 23; Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 14; *Haberstumpf*, GRUR 2003, 20, 26.

327 BT-Drs. 13/7385 S. 45.

4. Ergebnis

Der Hersteller eines trainierten KNN kann also als Datenbankhersteller auch in den Genuss des sui-generis-Datenbankherstellerschutzes gem. § 87a UrhG kommen.[328] Dieser erstreckt sich nicht auf den Quellcode.

5. Schutzumfang und Folgen

Der Schutzumfang des Datenbankherstellerrechtes ergibt sich aus § 87b Abs. 1 UrhG. Danach hat der Datenbankhersteller das ausschließliche Recht, die Datenbank insgesamt oder einen nach Art oder Umfang wesentlichen Teil der Datenbank zu vervielfältigen, zu verbreiten und öffentlich wiederzugeben. Datenbankhersteller ist, wer „die Initiative ergreift [die Datenbank herzustellen] und das Investitionsrisiko trägt",[329] mithin ist der Schutzinhaber nicht notwendigerweise identisch mit dem Urheber des Datenbankwerkes gem. § 4 Abs. 2 UrhG – ein Beispiel wäre ein Arbeitgeber-Arbeitnehmer-Verhältnis, in dem der Arbeitgeber eine Idee für ein ML-Modell und eine zu erfüllende Aufgabe entwickelt, aber seine Entwicklungsabteilung mit der Konzeption, der Datensammlung und dem Training des ML-Modells betraut.

III. Schutz als Computerprogramm gem. § 69 a UrhG

Möglicherweise kommt einem trainierten ML-Modell in *Python* auch der Schutz für Computerprogramme gem. § 69a UrhG zu.[330] In der oben (§ 6 D.III.) gefundenen Definition für trainierte KNN ist der Quellcode als ein wesentlicher Bestandteil des trainierten Modells genannt, wurde jedoch für den Datenbank(werk)schutz nicht berücksichtigt. Für den Schutz nach § 69a UrhG hingegen ist der Code – wie sich zeigen wird – von hoher Relevanz.

---

328 So wohl auch *Söbbing*, MMR 2021, 111, 114.

329 Erw.-Gr. 41 Datenbank-RL; Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 19.

330 Diskutiert wird das unter anderem von *Hartmann/Prinz*, WRP 12 2018, 1431, 1436 Rn. 47 ff.. und *Ehinger/Stiemerling*, CR 12 2018, 761, 764, Rn. 34 ff., insb. S. 767 Rn. 51, sowie *Grätz*, Künstliche Intelligenz im Urheberrecht, S. 46 ff..

1. Trainiertes Modell als Computerprogramm

Damit ein Schutz nach § 69a UrhG in Betracht kommt, muss ein Computerprogramm vorliegen. Weder das UrhG noch die durch §§ 69a ff. UrhG umgesetzte Computerprogramm-RL stellen eine Erklärung für die Bedeutung des Begriffes bereit, § 69a Abs. 1 UrhG spricht lediglich von „Programmen in jeder Gestalt". Infolgedessen sind andere Quellen für eine mögliche Definition zu suchen.

a) Begriffsklärung Computerprogramm

Anhaltspunkte liefern die DIN 44300 sowie die Definition der *WIPO* und das IEEE Standard Glossary:

- Ein Computerprogramm ist „eine Folge von Befehlen, die nach Aufnahme in einen maschinenlesbaren Träger fähig sind zu bewirken, dass eine Maschine mit informationsverarbeitenden Fähigkeiten eine bestimmte Funktion oder Aufgabe oder ein bestimmtes Ergebnis anzeigt, ausführt oder erzielt."[331]
- Ein Computerprogramm ist „eine zur Lösung einer Aufgabe vollständige Anweisung zusammen mit allen erforderlichen Vereinbarungen."[332]
- Ein Computerprogramm ist „eine Kombination aus Computerinstruktionen und Datendefinitionen, die Computerhardware dazu befähigen, berechnende oder kontrollierende Funktionen auszuführen."[333]

Mancherorts wird „Computerprogramm" zudem abgegrenzt zu „Software". Software soll dann alle digitalisierten Daten erfassen, also zwar auch Computerprogramme „im technischen Sinne", aber darüber hinaus auch Texte, Grafiken, Musikdateien und andere Daten.[334] Einigkeit scheint darüber zu be-

331 WIPO-Mustervorschriften, GRUR 1979, 306, § 1 (i); ebenso BGH GRUR 1985, 1041, 1047 – *Inkasso-Programm*.

332 Vgl. DIN 44300; Spindler/Schuster–*Wiebe*, Recht der elektronischen Medien, § 69a Rn. 3.

333 *Institute of Electrical and Electronics Engineers*, IEEE Standard Glossary of Software Engineering Terminology, S. 19.

334 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 2; Software als auch Computerprogramme umfassender, aber weiterer Begriff *Institute of Electrical and Electronics Engineers*, IEEE Standard Glossary of Software Engineering Terminology, S. 66.

stehen, dass es zur Abgrenzung für das Vorliegen eines Computerprogramms maßgeblich auf das „Vorhandensein von Befehls- und Steuerungsfunktionen" ankommt,[335] so verlangen auch die genannten Definitionen „eine Folge von Befehlen" bzw. eine „Anweisung".

b) (Keine) Einordnung von ML-Modellen als Computerprogramm in der Literatur

Teilweise wird der Schutz insbesondere künstlicher neuronaler Netze als Computerprogramm in einer Parallele zur Wissensbasis von Expertensystemen[336] kategorisch abgelehnt, ohne eine weitere Prüfung vorzunehmen.[337] An dieser Stelle soll jedoch eine differenziertere Betrachtung erfolgen.

Andernorts wird kritisiert, dass die „bestimmte Funktion", die die Definition der *WIPO* und damit auch der BGH verlangen, durch das „trainierte Netz" nicht gegeben sei, da sich die „Funktion" des „trainierten Netzes" im Rahmen des Trainingsvorgangs verändere.[338] Das scheint schon deshalb unpräzise, weil mit dem „trainierten Netz" der *finale* Zustand des KNN gemeint ist, und nicht der, der sich im Rahmen des Trainings noch verändert.[339]

Zudem seien die „Ausgabeparameter allein von den Eingangswerten abhängig",[340] weshalb das trainierte Netz dem „Datenaufbereiter" zuzurechnen sei.[341]

Möglicherweise wird hier zum einen verkannt, dass das „trainierte Netz" (zumindest nach dem hiesigen Verständnis des trainierten Modells) eben nicht nur – einmalig und flüchtig – in Form des Maschinencodes oder Bytecodes vorliegt, sondern vielmehr auf die Elemente abzustellen ist, die es

---

335 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 3; Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 12; DKM–*Kotthoff*, HK-UrhG, § 69a Rn. 5; OLG Rostock MMR 2008, 116.

336 Vgl. § 2 B.IV.2..

337 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 21.

338 *Hartmann/Prinz*, WRP 12 2018, 1431, 1436; *Linke*, GRUR Junge Wissenschaft 2019, S. 42.

339 Ein Unterschied könnte sich allenfalls im sogenannten „Online-Learning" ergeben: In dieser Variante wird ein Modell trainiert und implementiert, das dann aber im Praxiseinsatz weiter dazulernt (zur Begriffsklärung „online" in diesem Kontext vgl. *Goodfellow* et al., Deep Learning Handbuch, S. 310). Aber auch in diesem Fall müsste die „bestimmte Funktion" für den tauglichen Einsatz bereits bestehen.

340 *Hartmann/Prinz*, WRP 12 2018, 1431, 1437.

341 Dies., WRP 12 2018, 1431, 1437.

ermöglichen, das trainierte Netz auch einzusetzen. Das Verhältnis zwischen den Komponenten des wie hier definierten trainierten KNN und dem im Zuge der Ausführung entstehenden Maschinencode ist nicht anders zu beurteilen als das Verhältnis zwischen regulärem Quellcode und Maschinencode anderer Programme.

Zum anderen wird nicht berücksichtigt, dass die Ausgabeparameter zwar zu einem großen Teil von den Trainingsdaten abhängen, dass aber die Hyperparameter bzw. die Modellstruktur auch einen maßgeblichen Teil zum Ergebnis beitragen. Die Leistung allein auf den „Datenaufbereiter" zu begrenzen und dabei die Expertise unter anderem desjenigen, der die Hyperparameter wählt, außer Acht zu lassen, scheint zu kurz gegriffen.

c) Berücksichtigung der Bestandteile eines trainierten Modells für den Computerprogrammbegriff

Zu klären ist deshalb erneut, allerdings diesmal konkret in Bezug auf den Computerprogrammschutz, auf welche Komponente(n) des trainierten Modells bei einer Subsumtion unter § 69a UrhG abzustellen ist. Denkbare Anknüpfungspunkte wären der Quellcode, die erzeugten Dateien, sowie eine zur Laufzeit kombinierte Variante (im Ergebnis ähnlich Maschinencode) derselben. Für den Quellcode ist sauber zu trennen zwischen dem Quellcode, der für das Training des Modells verwendet wird, und dem Quellcode, der das trainierte Modell lädt und einsetzt. Für den Schutz *des trainierten KNN* als Computerprogramm ist ausschließlich letzterer relevant, denn das andere Programm hat eine ganz andere Zielrichtung bzw. Zwecksetzung, und möglicherweise auch eine sehr verschiedene Komplexität: Während der Code zur Erzeugung eines Modells Anweisungen zur Aufbereitung der Trainingsdaten, evtl. auch zu Funktionalitäten der einzelnen Modellbestandteile sowie Testprozeduren enthält, genügen für den Quellcode, der das trainierte Modell einsetzt, mitunter wenige Zeilen Programmcode, in denen das Modell und die Eingabedaten geladen und eine Vorhersage ausgeführt und ausgegeben wird (teilweise ist sogar eine einzeilige Skripteingabe in einer Kommandozeile ausreichend[342], allerdings hängt die Komplexität sowohl des Trainings- als auch des Ausführungscodes stark von der Aufgabenstellung und der individuellen Implementierung ab).

342 Vgl. https://www.tensorflow.org/guide/saved_model#run_command (Stand: 22.02.2021).

```
1   # neural network simple example
2   from numpy import loadtxt
3   from keras.models import load_model
4   # load the dataset
5   dataset = loadtxt('dataset.csv', delimiter=',')
6   #split into input (X) and output (y) variables
7   X = dataset[:,0:8]
8   y = dataset[:,8]
9   # define the keras model
10  model = load_model('my_model.h5')
11  # compile the keras model
12  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
13  # make class predictions with the model
14  predictions = model.predict_classes(X)
15  # summarize the first 5 cases
16  for i in range(5):
17      print ('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

Abbildung 7.1: Einfaches Beispiel in Keras: Laden eines Modells aus .h5-Datei, Quelle (leicht abgewandelt): https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/, 22.02.2021.

In Bezug auf den besagten Quellcode ergeben sich verschiedene Gestaltungsmöglichkeiten: So kann dieser die für den Produktiveinsatz erforderliche Struktur des Modells aufbauen, in dem der Programmierer die im Training ermittelten und optimierten Hyperparameter im Code fest vorgibt, und lediglich die Parameter aus Dateien nachladen, oder aber auch die Hyperparameter etwa im JSON-Format[343] bereitgestellt bekommen. Auch eine dritte Variante ist denkbar, aber praxisfern, bzw. nur für Testzwecke praktikabel: Der Programmierer könnte im Code zum einen die Hyperparameter festlegen, zum anderen das Modell zufällig mit Parametern versehen, das Training durchführen und im Anschluss (ohne das Modell persistent zu speichern) das Modell direkt zum vorgesehenen Einsatzzweck verwenden. Dies hätte allerdings eine enorm begrenzte, wenn nicht gar unmögliche Reproduzierbarkeit dieses Ergebnisses zur Folge. Das Beispiel soll nur verdeutlichen, welche Rolle der Quellcode beim Einsatz eines trainierten Modells einnehmen *kann*. Der Unterschied der dargestellten Varianten schlägt sich naturgemäß vor allem in der Komplexität des Codes nieder.

Abbildung 7.1 zeigt ein Minimalbeispiel für Quellcode in *Keras*, bei dem das Modell zuvor in einer `.h5`-Datei gespeichert wurde. In Zeile 6 werden die Testdaten eingelesen, die zuvor im CSV-Format (Comma-Separated-Values-

343 JSON (Java Script Object Notation) ist ein verbreitetes Format zur Strukturierung von Daten zur Datenbereitstellung bzw. Datenaustausch, das von vielen Programmiersprachen verarbeitet werden kann, für weitere Informationen vgl. https://www.json.org/json-de.html (Stand: 22.02.2021).

Format) zusammengetragen wurden. Diese werden für eine Vorhersage in Zeile 14 verwendet. `X` repräsentiert eine Sammlung von Testdaten, `y` die dazugehörigen Labels. In Zeile 10 wird das gespeicherte Modell eingelesen. Resultat ist bzw. sind die vom Modell gefundenen Labels für die eingegebenen Testdaten `X`.[344]

Ziel des folgenden Abschnitts ist, herauszustellen, welche Kombination von Elementen eines trainierten Modells mindestens gegeben sein muss, damit die Anforderungen an ein „Computerprogramm" im Sinne des § 69a UrhG erfüllt sind.

aa) Quellcode

Es liegt nahe, den Quellcode für die Subsumtion unter § 69a UrhG heranzuziehen. Quellcode meint dabei nur den für die konkrete Problemlösung entworfenen Code, die verwendeten („importierten") Frameworks mit ihren Klassen und Funktionen sind hier grundsätzlich außer Acht zu lassen, in Bezug auf diese kann nur die Auswahl der eingesetzten Klassen und Funktionen relevant sein, da die Implementierung durch andere Entwickler erfolgte.

Die aus unterschiedlicher Herangehensweise potenziell resultierende Komplexität ist erst für die Beantwortung der Frage, ob eine geistige Schöpfung vorliegt, von Relevanz.

Der Quellcode enthält in der Regel auch in seiner simpelsten Form eine Folge von aufgabenspezifischen Befehlen an eine Maschine, sei es der Import der notwendigen Frameworks (vgl. Abbildung 7.1 Zeilen 2 und 3), die Initialisierung eines Netzwerk-Objektes (vgl. Abbildung 7.1 Zeile 10) oder das Laden von Dateiinhalten.

Ein Computerprogramm im Sinne des § 69a Abs. 1 UrhG dürfte mithin im Rahmen des Quellcodes grundsätzlich gegeben sein.

344 In der Praxis müsste das Modell noch für den Einsatz in einer Anwendung vorbereitet werden – die Bereitstellung könnte z. B. über *Tensor Flow Serving* erfolgen, dafür würde das Modell noch mittels der *Tensor Flow*-Methoden vorbereitet, vgl. z. B. Ausführungen hier https://towardsdatascience.com/deploying-keras-models-using-tensorflow-serving-and-flask-508ba00f1037 (Stand: 22.02.2021).

bb) Hyperparameter und Parameter

Schon dem Begriff nach fällt es schwer, Daten, Datenbanken und Dateistrukturen – und damit also auch Hyperparameter – als Computerprogramme im Sinne des § 69a UrhG aufzufassen, zumal sie auch in den §§ 4 und 87a UrhG implizit aus dem Schutz ausgenommen werden.[345] Jedoch wird auch Quellcode in Dateien gespeichert, und auch kompilierter Code kann in Dateiform vorliegen. An dieser Stelle ist also zu klären, ob die in Dateien ausgelagerten Parameter und Hyperparameter ein Computerprogramm im Sinne des § 69a UrhG darstellen, oder doch zumindest ein Computerprogramm-Teil.

Wenngleich der Gedanke naheliegt, dass gespeicherte *Parameter* und *Hyperparameter* letztlich doch gespeicherten Steuerbefehlen ähneln könnten, so ist doch vernünftigerweise davon auszugehen, dass es sich lediglich um Informationen handelt, die von Steuerbefehlen *verwendet* werden, im Unterschied zu (kompiliertem) Code, der so, wie er vorliegt, an ein (Ausführ-) Programm übergeben und ausgeführt werden kann. Anders sieht dies *Grätz*, der unter Annahme eines weiten Begriffsverständnis von Computerprogrammen, das keine Steuerungsfunktion fordert, davon ausgeht, dass auch „Trainingsergebnisse“ (hier: Parameter) als Computerprogramm zu verstehen sind.[346] Dem ist nicht zuzustimmen. Hierbei wird übersehen, dass Trainingsergebnisse für sich nur eine Sammlung von Werten sind, die nicht ausgeführt werden können. Selbst wenn keine Steuerungsfunktion gefordert wird, fehlt den Daten eine für Computerprogramme übliche Struktur.

An der Steuerungsfunktion aber fehlt es: Hyperparameter und Parameter sind letztendlich nur Werte, die nach dem Laden der Datei durch den Quellcode im Programm Variablen zugewiesen werden können, aber es handelt sich eben nicht um eigenständige Steuerbefehle. Damit verfängt auch das Argument nicht, dass außerhalb des Quellcodes gespeicherte Werte das Verhalten des Computers auf gleiche Weise steuern wie im Code selbst hinterlegte Werte, weshalb auch die ausgelagerten Werte genauso Teil des Computerprogramms seien.[347] Die ausgelagerten Hyperparameter für sich sind faktisch nämlich nicht in der Lage, den Computer zu steuern.

---

345 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 17.

346 *Grätz*, Künstliche Intelligenz im Urheberrecht, S. 52.; einen weiten Computerprogrammbegriff ebenso erwägend, aber den Schutz aufgrund fehlender geistiger Schöpfung ablehnend *Ehinger/Stiemerling*, CR 12 2018, 761, 768; das OLG Hamburg setzt ebenfalls für die Annahme eines Schutzes voraus, dass die fragliche Datei Steuerbefehle enthält, s. OLG Hamburg MMR 1999, 230, 231 – *Superfun*.

347 *Nebel/Stiemerling*, CR 1 2016, S. 66.

Allerdings erfasst § 69a UrhG auch Entwurfsmaterial, und damit vorgelagerte Ausdrucksformen eines Programms, wie zum Beispiel Flussdiagramme oder andere Vorstufen des Computerprogramms.[348] Es könnte daran gedacht werden, die im JSON- oder ähnlichen Format abgelegten Hyperparameter (nicht jedoch die errechneten bzw. optimierten Parameter) über diese Vorschrift in den Schutz einzubeziehen, stellen sie doch in gewisser Weise den „Entwurf" des trainierten Netzwerkes dar, dergestalt dass sie den Aufbau des entstehenden KNN bestimmen.

Nach den Erwägungsgründen der für die Entstehung von § 69a UrhG maßgeblichen Computerprogramm-RL muss das am Schutz teilnehmende Entwurfsmaterial zur Entwicklung (oder zur Vorbereitung) eines Computerprogramms dienen, wobei die Art der vorbereitenden Arbeit die spätere Entstehung eines Computerprogramms zulassen muss.[349] Problematisch erscheint, dass die Hyperparameter nicht zur Vorbereitung auf die *Entwicklung eines Programms* dienen, sondern lediglich abstrakt das KNN (ohne die Gewichtungsinformationen) abbilden. Entwurfscharakter haben sie also allenfalls für das entstehende KNN, nicht aber für ein konkretes Computerprogramm. Die Subsumtion unter den Begriff „Entwurfsmaterial" will hier demzufolge nicht so recht passen.[350] Vielmehr sind sie als computeranweisungslose Daten einzuordnen, die keinen Schutz als Computerprogramm genießen.[351]

Weder die abgespeicherten Parameter noch die Hyperparameter sind folglich – für sich betrachtet – dem Schutz nach § 69a UrhG zugänglich.

cc) Kombination zur Laufzeit

Möglicherweise können die in Dateien ausgelagerten Informationen jedoch anderweitig in den Schutz aus § 69a UrhG einbezogen werden: Die Parameter und Hyperparameter werden aus den Dateien eingelesen und im flüchtigen Arbeitsspeicher mit den aus dem Quellcode generierten Steueranweisungen im Bytecode zusammengeführt. § 69a UrhG erfasst jede Ausdrucksform

348 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 14.
349 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 7; Erwägungsgrund 7 und Art. 1 Abs. 1 S. 2 Computerprogramm-RL.
350 Gegen einen Schutz als Entwurfsmaterial auch *Hartmann*/*Prinz*, WRP 12 2018, 1431, 1435 f..
351 Vgl. dazu Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 12.

eines Computerprogramms, auch den Maschinen- und Objektcode.[352] Damit könnte also unter Umständen auf das nur flüchtig existierende Endprodukt als Schutzobjekt abgestellt werden, wobei jedoch fraglich ist, ob dies eine ausreichend manifestierte Form darstellt. Grundsätzlich reicht jedoch die (auch einmalige) Wahrnehmbarkeit, eine permanente Fixierung ist nicht erforderlich.[353] Die entstehende „Kombination zur Laufzeit" ist eine andere Ausdrucksform des Computerprogramms, das bereits durch den Quellcode geschützt ist. Der entstehende Bytecode ist also, sofern er überhaupt außerhalb des Programmablaufs erreichbar ist – ebenso wie bereits der Quellcode – Computerprogramm im Sinne des § 69a Abs. 1 UrhG, jedoch nicht zusätzlich zum Quellcode, sondern in dessen Rahmen geschützt.

dd) Sonstige Schutzgegenstände

In Bezug auf KNN wird häufig auch „der Algorithmus" als Schutzgegenstand diskutiert,[354] wobei es wesentlich darauf ankommt, was als „der Algorithmus" verstanden wird.[355] Der „Algorithmus" im Sinne des aus dem Quellcode entstehenden Programms wurde bereits abgehandelt.[356] Es ist möglich, das Verständnis eines KNN auf die zugrundeliegenden statistischen Rechenregeln zu reduzieren. Diese wären sodann als Algorithmus im mathematischen Sinne aufzufassen und als Ideen und Grundsätze gem. § 69a Abs. 2 UrhG nicht schutzfähig. Die hier verwendete Definition eines KNN rekurriert jedoch nicht auf diese abstrakten Rechenregeln, sondern auf deren konkreter Umsetzung durch die in § 6 D.III. erläuterten Bestandteile eines trainierten Modells, sodass es allein auf deren Schutzfähigkeit ankommt. Eine Diskussion der Schutzfähigkeit des Algorithmus als gesonderter Schutzgegenstand ist also entbehrlich.

---

352 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 11; Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 19.

353 Wandtke/Bullinger–*Grützmacher*, PK UrhR, § 69a Rn. 11.

354 Vgl. z. B. *Hauck*/*Cevc*, ZGE 11 2019, 135, 160; *Linke*, GRUR Junge Wissenschaft 2019, S. 36 ff., S. 40 f..

355 Zur begrifflichen Ambivalenz vgl. Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 22; zur Einordnung in § 69a UrhG vgl. *Söbbing*, CR 4 2020.

356 Vgl. oben aa) und cc).

ee) Zusammenfassung: infrage kommende Schutzgegenstände

Für den Schutz als Computerprogramm kommt folglich nur der das Modell ladende und ausführende Quellcode, sowie damit auch der entstehende Bytecode infrage.

2. Eigene geistige Schöpfung

Als Schutzgegenstand wurde der das trainierte KNN ladende und ausführende Quellcode identifiziert. Für einen Schutz gem. § 69a UrhG muss diesem die Qualität eines individuellen Werkes im Sinne eines Ergebnisses einer eigenen geistigen Schöpfung des Urhebers zukommen (§ 69a Abs. 3 S. 1 UrhG). Auffällig ist hier, dass nicht – wie etwa in § 2 Abs. 2 UrhG – eine „persönliche", sondern eine „eigene" geistige Schöpfung gefordert wird. Dies ist jedoch der Umsetzung der Computerprogramm-RL, und damit der Absenkung der vormals noch deutlich höheren Schutzvoraussetzungen,[357] geschuldet,[358] es besteht Einigkeit darüber, dass – unabhängig vom Wortlaut in § 69a Abs. 3 S. 1 UrhG – eine individuelle persönliche geistige – menschliche – Schöpfung im Sinne des § 2 Abs. 2 UrhG erforderlich ist.[359] Folglich ist auch hier eine menschlich-gestalterische Tätigkeit erforderlich, die „einen geistigen Gehalt aufweist, zu einer Formgestaltung geführt hat und eine hinreichende Individualität erkennen lässt".[360]

a) Menschlich-gestalterische Tätigkeit

Das Kriterium der menschlich-gestalterischen Tätigkeit dürfte – sofern nicht modellprogrammerzeugende Algorithmen eingesetzt werden – unproblematisch für den Quellcode-Anteil trainierter Modelle vorliegen. Dieser wird auch nicht automatisiert geändert, die automatisierten Einfügungen der Parameterwerte und Hyperparameter erfolgen zum einen durch den Programmierer initiiert, und zum anderen erst bei Programmablauf. Abzustellen ist aber auf den Zeitpunkt der Herstellung des Quellcodes.

357 BGH GRUR 1985, 1041, 1047 – *Inkasso-Programm*
358 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 25.
359 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 25; Nordemann/Fromm–*Czychowski*, UrhR, § 69a Rn. 16; Ahlberg/Götting–*Kaboth/Spies*, BeckOK-UrhG, § 69a Rn. 13.
360 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.

Ferner müsste der Quellcode einen geistigen Gehalt aufweisen, zu einer Formgestaltung geführt haben und eine hinreichende Individualität erkennen lassen.[361]

b) Geistiger Gehalt

Geistigen Gehalt weist das Computerprogramm auf, wenn in ihm der menschliche Geist zum Ausdruck kommt[362] – die Idee, das Konzept, das der Programmierer oder Softwareentwickler umsetzen möchte, muss sich also in dem Quellcode wiederfinden. Im Quellcode eines trainierten KNN bedient sich der Programmierer einer Reihe verfügbarer Frameworks (sofern – wie im Regelfall – nicht der gesamte Quellcode neu geschrieben, also quasi „das Rad neu erfunden" wird). Schon die Auswahl derselben und daran anschließend die Wahl der Objekttypen, die Art des KNN, und die Netzwerkarchitektur, also die Auswahl der Anzahl der Schichten, der Aktivierungsfunktionen etc. sind ein Abbild der Idee, mit dem der Entwickler sein Ziel zu erreichen gedenkt, und mithin Ausdruck seines Geistes.

c) Wahrnehmbarkeit

Das Computerprogramm müsste ferner als Ergebnis der schöpferischen Tätigkeit des Programmierers der Wahrnehmung durch die menschlichen Sinne zugänglich sein,[363] es handelt sich bei Quellcode um niedergeschriebenen Text, der vom Menschen gelesen werden kann, und in der ausgeführten Fassung durch Interaktionsmöglichkeiten (Eingabe von zu analysierenden Daten, Ausgabe der Vorhersagen oder generierten Erzeugnisse) auch durch Nicht-Entwickler wahrnehmbar ist, auch dieses Merkmal liegt also unproblematisch vor.

d) Individualität

Möglicherweise scheitert die Subsumtion jedoch an der nötigen erforderlichen Individualität, also der „eigenpersönlichen Ausnutzung des bestehenden

361 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.
362 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.
363 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.

Gestaltungsspielraumes“[364]. Diese Schlussfolgerung liegt nahe, wenn berücksichtigt wird, dass der Vorgang der Initialisierung eines KNN (also *Quellcode 2* in Abbildung 6.2) für unterschiedlichste Anwendungszwecke sehr ähnlich bis identisch ausfallen kann.

Das wird umso deutlicher anhand eines Beispiels: Entwickler A möchte ein KNN darauf trainieren, Autos in Bildern zu erkennen, und verwendet dafür einen Datensatz mit Bildern von Autos sowie ein ML-Modell in einer bestimmten Konfiguration. Entwickler B möchte ein KNN darauf trainieren, Bäume in Bildern zu erkennen. Dafür verwendet er einen gänzlich anderen Datensatz mit Bildern von Bäumen, aber die gleiche Konfiguration des KNN. Der Quellcode für beide Modelle kann mithin identisch ausfallen, obgleich eine vollkommen andere Aufgabe erfüllt wird (und sich die trainierten Parameter auch wesentlich unterscheiden).

Möglicherweise existiert also ein „Basisbefehlssatz“, der für eine Vielzahl unterschiedlicher Aufgabenstellungen identisch und zugleich unerlässlich sein kann, insbesondere wenn die Hyperparameter in eine Datei ausgelagert wurden, und für den folglich ein Freihaltebedürfnis im Sinne der „Building Blocks“ eines ML-Modells bestehen könnte.

Zugleich darf jedoch die Schwelle der Individualität nicht zu hoch angesetzt werden.[365] Es verbietet sich an dieser Stelle eine pauschale Beurteilung, der Quellcode *kann* mitunter auch sehr individuell ausfallen. Zudem kann ein Programm, das zwar mit dem bis auf Dateipfade identischen Quellcode wie ein anderes Programm abläuft, dem aber gänzlich verschiedene Trainingsdaten und Hyperparameter zugrunde liegen, eine gänzlich andere Aufgabe oder eine gleiche Aufgabe mit höherer Präzision durchführen, sodass die Individualität auch in der Zusammenschau mit anderen Bestandteilen des Computerprogramms liegen kann.

Es wird vielmehr auf eine Einzelfallbetrachtung ankommen müssen. Nach der in § 7 B.I. und § 7 B.II. vorgeschlagenen Vorgehensweise (Schutz der Hyperparameter bzw. des trainierten KNN als Datenbank(werk)) dürfte es für den Entwickler, der einen „Standardquellcode“ zum Aufruf des KNN verwendet, jedoch auch nicht schädlich sein, wenn dieser nicht gem. § 69a UrhG geschützt werden kann, weil dann ein großer Teil seiner geistigen Schöpfung bereits Datenbankwerkschutz genießt. In vielen Fällen wird der Entwickler jedoch vermutlich zumindest einige Anpassungen des „Standardquellcodes“

364 Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.

365 Dies gilt grundsätzlich für den Computerprogrammschutz, vgl. BGH GRUR 2005, 860, 861 – *Fash 2000*; Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 26.

vornehmen müssen, sodass dann – aufgrund der sehr niedrigen Schutzanforderungen – auch dafür ein Schutz nach § 69a UrhG angenommen werden kann.[366]

3. Ergebnis

Trainierte ML-Modelle sind dem Computerprogrammschutz nach § 69a UrhG zugänglich, wenn der Quellcode als zentraler Schutzgegenstand herangezogen wird.

Inwiefern der urheberrechtliche Computerprogrammschutz für ein trainiertes Modell relevant ist, hängt zu großen Teilen davon ab, ob der Quellcode selbst die zentralen Funktionalitäten enthält, oder ob ein minimaler „Standardquellcode" verwendet wird, während die eigentliche „Magie" des trainierten Modells in ausgelagerten Hyperparametern und Parametern liegt. In letzterem Fall dürfte dem Schutzsuchenden wohl der Datenbank- und ggf. Datenbankherstellerschutz besser dienen;[367] wenn jedoch eine wesentliche Leistung des Entwicklers in der Implementierung des Quellcodes besteht, schützt der Computerprogrammschutz vor der Übernahme insbesondere der Gesamtstruktur und auch von (für sich als schutzfähig zu befindenden) Programmteilen.[368]

IV. Zusammenfassung

Trainierte ML-Modelle sind durch die verschiedenen infrage kommenden Schutzgegenstände in unterschiedlicher Weise urheberrechtlich schutzfähig. Sofern der Fokus der Betrachtung auf dem Quellcode liegt, kommt ein Computerprogrammschutz nach § 69a UrhG infrage. Für die gewählten und gespeicherten Hyperparameter und Parameter kommt in der Kombination außerdem Datenbankwerkschutz gem. § 4 Abs. 2, Abs. 1 UrhG sowie das sui-generis Datenbankherstellerrecht gem. §§ 87a ff. UrhG in Betracht.

366 Einen Computerprogrammschutz nur im Einzelfall für möglich haltend *Apel/Kaulartz*, RDi Nr.1 2020, 24, 28.
367 Vgl. dazu § 7 B.I. und § 7 B.II..
368 Vgl. Dreier/Schulze–*Dreier*, UrhG, § 69a Rn. 21, 23.

```
> getTree(fmodel, k=2, labelVar = TRUE) %>%
+     head(20)
   left daughter right daughter  split var split point status prediction
1              2             3         G2  0.50000000      1       <NA>
2              4             5       Fedu  0.37500000      1       <NA>
3              6             7       Walc  0.12500000      1       <NA>
4              8             9     health  0.37500000      1       <NA>
5             10            11         G2  0.44736842      1       <NA>
6             12            13 activities  1.00000000      1       <NA>
7              0             0       <NA>  0.00000000     -1       pass
8             14            15        age  0.64285714      1       <NA>
9             16            17   absences  0.01333333      1       <NA>
10            18            19     health  0.12500000      1       <NA>
11            20            21       Mjob  6.00000000      1       <NA>
12             0             0       <NA>  0.00000000     -1       pass
13            22            23   guardian  1.00000000      1       <NA>
14             0             0       <NA>  0.00000000     -1       pass
15             0             0       <NA>  0.00000000     -1       fail
16             0             0       <NA>  0.00000000     -1       fail
17            24            25   guardian  3.00000000      1       <NA>
18            26            27   absences  0.01333333      1       <NA>
19             0             0       <NA>  0.00000000     -1       fail
20            28            29 activities  1.00000000      1       <NA>
>
```

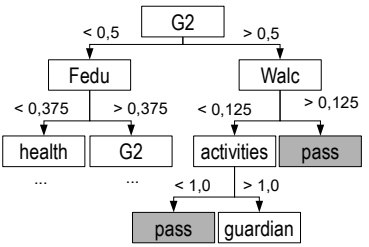

Abbildung 7.2: Die ersten 20 Einträge eines Baumes in einem Random Forest in Tabellen- und Baumform, Quelle: eigene Darstellung.

C. *Trainierter Random Forest in* R

Das Konzept der Random Forests wurde bereits in § 2 B.II.1. eingeführt, hier folgt eine Veranschaulichung anhand eines Beispiels in der Programmiersprache *R*, um die urheberrechtliche Analyse und insbesondere die sich von den dargestellten *Python*-Modellen unterscheidende Behandlung besser nachvollziehen zu können.

Abbildung 7.2 zeigt in tabellarischer Darstellung einen „Baum" aus einem Random Forest, der auf einem Trainingsdatenset über portugiesische Schüler[369] und deren Erfolgsquote im Schuljahr trainiert wurde.[370] Anhand dieser Abbildung kann die Funktionsweise eines Entscheidungsbaums nachvollzogen werden: Aufgabe des Modells ist es, für einen Studierenden vorherzusagen, ob er die Jahresabschlussklausur bestehen wird oder nicht. Der Datensatz enthält Informationen unter anderem über die Schule, das Geschlecht, das Alter, die Adresse, die Berufe der Eltern, bisheriges Nichtbestehen von Klausuren, Anfahrtszeiten zur Schule und Freizeitaktivitäten.

Das Modell wurde in der Programmiersprache *R* entwickelt. Die Tabelle zeigt aus Praktikabilitätsgründen hier nur die ersten 20 Zeilen des Baumes an und ist wie folgt zu lesen:

369 Datensatz: https://archive.ics.uci.edu/ml/datasets/Student+Performance (Stand: 22.02.2021); *Carvalho Brito*, 15th European Concurrent Engineering Conference 2008, ECEC '2008 [and] 5th Future Business Technology Conference, FUBUTEC '2008: April 9 - 11, 2008, Porto, Portugal.

370 Beispiel entwickelt anhand von https://www.machinegurning.com/rstats/deploying_models/ (Stand: 22.02.2021).

Der oberste Knoten (1) hat zwei Kindknoten („left daughter“ = Knoten Nr. 2 und „right daughter“ = Knoten Nr. 3). Die Eigenschaft der untersuchten Schüler, die im ersten Knoten überprüft wird („split var“), ist „G2“ – die in Prozent gemessene Note des zweiten Leistungsmessungszeitraums. Wenn die erreichte Note über 50 Prozent (0.5) lag, wird zum Knoten 2 gesprungen, es geht dann also in Zeile 2 weiter, ansonsten zu Knoten 3. Knoten 2 hat die Kindknoten 4 und 5, und überprüft auf das Kriterium der Ausbildung des Vaters („Fedu“). Knoten 3 hat die Kindknoten 6 und 7, und überprüft, wieviel Alkohol der Schüler am Wochenende konsumiert („Walc“). Für einen Alkoholkonsumwert unter 0.125 geht es bei Knoten 7 weiter, bei dem hier zum ersten Mal in der Spalte „prediction“ ein Wert angegeben ist: Der Schüler hätte also bei der Kombination gute zweite Klausur (G2) und wenig Alkoholkonsum am Wochenende vermutlich die Schuljahresendklausur bestanden.[371]

Die Tabelle in Abbildung 7.2 zeigt, dass in einem Random Forest-Modell – bzw. für Entscheidungsbäume im Allgemeinen – die trainierte Baumstruktur bereits ausreicht, um Vorhersagen für neue Daten treffen zu können. Die Tabellen aller Bäume in einem Random Forest können dann in eine Tabelle zusammengefasst und für Vorhersagen verwendet werden. Ein erneutes Einlesen in ein Programm in der Ausgangsprogrammiersprache ist nicht erforderlich, die Informationen können schlicht in einer „klassischen“ Datenbank gespeichert und mit einem beliebigen Computerprogrammcode, der die Tabelle versteht, ausgelesen werden – dabei ist dies nicht einmal zwingend erforderlich, sondern reduziert lediglich den zeitlichen Aufwand für den Menschen. Insofern gilt für ein Random Forest-Modell in *R* – im Unterschied zu *Python*-Modellen – dass ein trainiertes Modell und die trainierten Parameter nicht auseinanderfallen. Eine separate Prüfung erübrigt sich deshalb, es gelten die Aussagen, die diesbezüglich sogleich zum Schutz trainierter Parameter getroffen werden.[372] Die Prüfung erfolgt zusammengefasst mit den trainierten Parametern der *Python*-Modelle, um hier den Kontrast besser herauszustellen.

371 Die Split-Werte sind nicht als absolute Werte zu lesen (so ergibt der „Alkoholkonsumwert 0.125“ ohne weitere Informationen keinen Sinn): Für die Berechnung und Optimierung der Split-Werte werden die in allen betrachteten Datensätzen gefundenen Werte berücksichtigt, der größte Wert entspricht dann 1.0 und der kleinste Wert 0.0, alle anderen Werte liegen proportional dazwischen. So könnte etwa ein Konsum von 2 Liter Alkohol am Wochenende dem Wert 1.0 entsprechen, 0 Liter würde dann der Wert 0.0 zugeordnet.

372 Vgl. § 7 D..

Insbesondere gibt es zudem keinen Quellcode, der zwangsläufig Bestandteil des trainierten Random Forest-Modells in *R* ist. Dessen ungeachtet kann in der Anwendung des trainierten Modells ein schutzfähiger Quellcode zum Einsatz kommen, und auch für den Quellcode, mithilfe dessen der trainierte Random Forest erzeugt wird, kommt ein Schutz nach § 69a UrhG infrage.

D. Trainierte Parameter

Bereits geklärt wurde die urheberrechtliche Schutzfähigkeit trainierter *Python*-ML-Modelle, für die die Parameter jeweils nur ein Element des zu schützenden Werkes darstellen. Für Random Forests in R wurde festgestellt, dass das trainierte Modell (also die entstehende Tabelle) den trainierten Parametern entspricht. Für die Prüfung der Schutzfähigkeit der Tabelle wurde insofern auf die Prüfung trainierter Parameter verwiesen.

In einem nächsten Schritt ist also zu klären, ob auch die trainierten Parameter schutzfähig sind. Zum einen ist dabei zwischen trainierten Parametern, die nur einen Teil eines trainierten Modells in *Python* ausmachen, und trainierten Parametern, die das gesamte trainierte Random Forest-Modell in *R* darstellen, zu unterscheiden ist, zum anderen bestehen aber auch strukturelle Unterschiede zwischen Random Forest-Modellen und künstlichen neuronalen Netzen, die sich etwa auf den Datenbankcharakter auswirken könnten.

I. Schutz als Datenbankwerk gem. § 4 Abs. 2 UrhG

Für einen Schutz als Datenbankwerk müsste die Ansammlung von trainierten Parametern gem. § 4 Abs. 2 UrhG i. V. m. § 4 Abs. 1 UrhG einem Sammelwerk entsprechen, dessen Elemente systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel oder auf andere Weise zugänglich sind, und das aufgrund der Anordnung oder Auswahl der Elemente eine geistige Schöpfung darstellt.

1. Datenbankwerk

An dieser Stelle wird nicht kategorisch zwischen ML-Modellen in *Python* und Random Forests in *R* unterschieden, sondern zwischen KNN und Ran-

dom Forests an sich, da die Unterschiede der Modelle in Bezug auf die trainierbaren Parameter hier besonders hervortreten.

a) Künstliches neuronales Netz

Elemente der Sammlung trainierter Parameter eines KNN sind die einzelnen Werte, die indizieren, mit welcher Gewichtung der Output eines Neurons an das nächste Neuron weitergeleitet wird. Der Abruf eines einzelnen Wertes eröffnet darüber hinaus auch Informationen über die Position im Netzwerk, für die dieser Wert gilt (diese Information muss zwangsläufig enthalten sein, denn ansonsten könnten die Werte nicht wieder eingelesen werden, um das Training fortzusetzen bzw. das trainierte Netz produktiv zu verwenden). Analog zur Postleitzahl[373] könnte dadurch auch dem einzelnen trainierten Parameter ein Informationswert zuzusprechen sein.[374] Die methodische bzw. systematische Anordnung und die einzelne Zugänglichkeit, die schon für das Zusammenspiel zwischen abgespeicherten Parametern und Quellcode unerlässlich sind, liegen ebenfalls vor. Mithin sind nicht nur die Kriterien der systematischen bzw. methodischen Anordnung sowie der einzelnen Zugänglichkeit, sondern insbesondere aufgrund ihres jeweils vorhandenen eigenen Informationswerts auch das der Unabhängigkeit der Elemente erfüllt.

b) Random Forest

Elemente eines trainierten Random Forest-Modells könnten die einzelnen Bäume oder – eine Ebene tiefer – die Knoten samt ihrer Schwellwerte und Feature-Information darstellen. Der Knoten als Element gibt also Auskunft über ein in dem zugehörigen Baum untersuchtes Feature sowie den ermittelten Schwellwert (im Beispiel der portugiesischen Schüler: untersuchtes Feature könnte die am Wochenende konsumierte Alkoholmenge sein, der Schwellwert ist dann die auf einen Wert zwischen 0 und 1 skalierte Alkoholmenge, die im weiteren Verlauf voraussichtlich zu einem Nichtbestehen führt). Die Elemente sind folglich unabhängig informierend. Die Sammlung erfolgt zwangsläufig strukturiert, und die Elemente sind einzeln abrufbar.

373 Vgl. wieder Nordemann/Fromm–*Czychowski*, UrhR, § 4 Rn. 26; b).
374 A.A. *Hartmann/Prinz*, WRP 12 2018, Rn. 62.

c) Zwischenergebnis

Das Vorliegen einer Datenbank im Sinne des § 4 Abs. 2 UrhG kann also sowohl für Random Forests als auch für KNN angenommen werden.

2. Persönliche geistige Schöpfung

Es bleibt zu klären, ob jeweils auch eine persönliche geistige Schöpfung im Sinne des § 4 Abs. 1 UrhG vorliegt. An dieser Stelle genügt die Differenzierung zwischen KNN und Random Forest nicht mehr, sondern es ist anhand der verwendeten Technologie zu unterscheiden: Bei der Entwicklung von Random Forests in *R* nimmt die Parametersammlung eine andere Form an als in *Python*, jedoch können auch in *Python* Random Forests trainiert werden. Die weitere Prüfung unterscheidet daher zwischen ML-Modellen in *Python* und Random Forests in *R*.

a) ML-Modell in *Python*

In *Python* werden die betreffenden Parameter-Werte unzweifelhaft durch einen Computer erzeugt bzw. errechnet.[375] Anknüpfungspunkt für die persönliche geistige Schöpfung ist die Anordnung oder Auswahl der Elemente. Auch die Anordnung der Parameter wird vom Framework vorgegeben und durch den Computer vorgenommen, ohne dass der Mensch einzugreifen braucht. Auf die Auswahl hat der Mensch allenfalls beschränkten Einfluss durch die Auswahl der Hyperparameter und indem er bestimmt, wann er den Trainingsprozess für abgeschlossen erklärt (denn erst dann verändern sich die Parameter nicht mehr). In keinem Fall jedoch wählt der Mensch die Parameter gezielt selbst aus.

Die Idee, die der Entwickler mit dem KNN umsetzen möchte, ist ferner nicht (allein) in der Parametersammlung verkörpert. Das beabsichtigte Ergebnis kann beim Einsatz der vorgestellten *Python*-Frameworks erst im Zusammenwirken mit den ausgewählten Hyperparametern erzielt werden. Die trainierten Parameter sind – auch als Sammlung – für sich genommen also keine persönliche geistige Schöpfung des Entwicklers, sondern lediglich ein

375 Vgl. zur Berechnung im Rahmen des Trainingsvorgangs z. B. oben § 3 A.II..

Bestandteil einer umfassenderen Schöpfung, ohne selbst dieses Erfordernis zu erfüllen.

b) Random Forest in *R*

Eine differenzierte Betrachtung ist in Bezug auf Random Forest-Modelle in *R* geboten. Es besteht die Möglichkeit, das Ergebnis eines Trainingsvorgangs – also „den Random Forest" – in einer „klassischen" elektronischen Datenbank zu speichern und – ohne eine erneute Initialisierung eines Random Forest-Objektes in der ursprünglichen Programmiersprache – damit bereits Vorhersagen zu treffen. Mitunter entsprechen also die trainierbaren Parameter eines Random Forest-Modells dem trainierten Modell. Allerdings werden die Bäume und Knoten ebenfalls durch den Computer ermittelt. Die vom Entwickler gewählten Hyperparameter haben lediglich begrenzende Wirkung (bspw. kann eine maximale Baumtiefe vorgegeben werden). Der Entwickler überlässt es aber dem Algorithmus, die optimale Konfiguration zu finden. Die einzige Auswahlleistung des Menschen besteht erneut darin, zu bestimmen, wann das computergenerierte Ergebnis ausreichend den eigenen Vorstellungen entspricht. Aber auch hierbei wählt der Entwickler nicht gezielt Parameter, sondern bestimmt in der Regel anhand einer Metrik (wie zum Beispiel durch Anwendung einer Verlustfunktion, oder andere gängige, in der Regel nicht selbst entwickelte sondern in Entwicklerkreisen bewährte Funktionen, die die Abweichung der Vorhersage von der aufgrund der Labels erwarteten Vorhersage berechnen), wie nah das Ergebnis an das Optimum heranreicht, und lässt das Modell die Parameter so lange optimieren, bis ihm das Gesamtergebnis genau genug erscheint.

Fraglich könnte aber sein, ob der Computer von einem Menschen steuernd als Hilfsmittel bzw. Werkzeug zur Erzeugung der Struktur eingesetzt wurde. In diesem Fall könnte eine persönliche geistige Schöpfung bejaht werden, denn dann könnte eine menschlich-gestalterische Handlung des Entwicklers vorliegen.[376] Wenn etwa der Entwickler alle Rechenschritte, die der Computer durchführen soll, vorgibt, und diese dann lediglich automatisiert ausgeführt werden, könnte von einem Einsatz als Werkzeug gesprochen zu sprechen sein. Zwar gibt es bei Random Forest-Modellen auch eine Zufallskomponente, auf die der Entwickler keinen Einfluss hat: So werden etwa die Features

376 Schricker/Loewenheim–*Loewenheim*, UrhR, Rn. 39f., Dreier/Schulze–*Schulze*, UrhG, § 2 Rn. 8.

der einzelnen Bäume durch den Algorithmus zufällig ausgewählt, ebenso die Datensätze, die die Bäume durchlaufen. Sowohl die Features als auch die Datensätze haben einen maßgeblichen Einfluss auf bzw. bestimmen die berechneten Knotenwerte, sodass die Leistung des Entwicklers vermeintlich verblasst. Jedoch legt der Entwickler – durch Anwendung besagter Metriken, und aus eigener Erfahrung – fest, wann das trainierte Modell seinen Vorstellungen entspricht, und gibt damit zum Ausdruck, dass in dem Ergebnis, also dem trainierten Random Forest-Modell in Form der durch das als Werkzeug eingesetztes Computerprogramm ausgewählten Werten in der Tabelle, seine persönliche geistige Schöpfung zum Ausdruck kommt. Die Prüfung gestaltet sich insofern ähnlich der Prüfung, die später für Erzeugnisse von ML-Modellen vorzunehmen sein wird:[377] Der Entwickler wählt – für die Auswahl der Elemente seiner Sammlung, um die es hier als Bezugspunkt der geistigen Schöpfung geht – ein Framework aus, setzt begrenzende Parameter, und wählt aus mehreren Ergebnissen eines aus. Dies ist für trainierte Random Forests in *R* aus den genannten Gründen der Fall, im Unterschied zu trainierten Parametern eines Modells in *Python*, die nur einen Teil des Ergebnisses darstellen.

Folglich liegt in der Auswahl der Elemente der Sammlung „trainierte Random Forests in *R*“ eine persönliche geistige Schöpfung des Entwicklers bzw. Data Scientists.

3. Ergebnis

Im Ergebnis ist zu differenzieren zwischen trainierten Parametern der ML-Modelle in *Python* und den Ergebnistabellen trainierter Random Forest-Modelle in *R*. Während für erstere der Schutz spätestens am Vorliegen einer persönlichen geistigen Schöpfung scheitert, kann eine solche für Random Forests in *R* angenommen werden, mit der Folge des urheberrechtlichen Schutzes der Tabellen als Datenbankwerk nach § 4 Abs. 2 UrhG i. V. m. § 4 Abs. 1 UrhG.

377 S. unten d); dort in Anlehnung an *Dreier*, FS Kitagawa, S. 881.

II. Investitionsschutz gem. §§ 87a ff. UrhG

Wenngleich ein Schutz nach § 4 UrhG zumindest für ML-Modelle in *Python* nicht in Betracht kommt, so könnte für die trainierten Parameter doch ein sui-generis-Datenbankherstellerschutz gem. §§ 87a ff. UrhG einschlägig sein. Im Gegensatz zu einem Schutz nach § 4 Abs. 2 UrhG ist hierfür gerade keine persönliche geistige Schöpfung erforderlich, sondern eine Investition in die Beschaffung, Sammlung, Überprüfung, Aufbereitung und Darbietung des Inhalts der Datenbank.[378]

1. Vorliegen einer Datenbank

Sowohl für die trainierten Parameter eines KNN als auch für die Baumstruktur eines Random Forest-Modells wurde die Datenbankqualität bereits untersucht und bejaht (vgl. § 7 D.I.1.).

2. Investition

Zusätzlich muss für § 87a UrhG eine im Rahmen der Beschaffung, Überprüfung oder Darstellung der Datenbankinhalte anfallende, sowie nach Art oder Umfang wesentliche Investition vorliegen (§ 87a Abs. 1 S. 1 UrhG). Für die Berechnung der trainierten Parameter sowie der Baumstruktur sind mitunter teure Hardwarekomponenten sowie erheblicher Zeitaufwand erforderlich (unter anderem für die Sammlung der Trainingsdaten, Rechenzeit und manuelle Optimierung der Hyperparameter). Zu prüfen ist, in wieweit sich diese Aufwände den in § 87a UrhG aufgeführten Investitionszwecken zuordnen lassen.

a) Gegenstand der Investition

Als Bezugspunkte der Investition nennt § 87a UrhG die Beschaffung, Überprüfung oder Darstellung der Datenbankinhalte. Dabei bezieht sich die *Beschaffung* stets auf bereits bestehende Elemente, nicht auf die Erzeugung

378 Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 1.

derselben.[379] Dies ergibt sich auch schon aus den Erwägungsgründen zur Datenbank-RL: Es sollen Lösungen geschützt werden, die dazu beitragen, durch Informationsmanagementsysteme der wachsenden Datenmenge Herr zu werden – und nicht solche, die die Datenmenge erweitern.[380] Dies bekräftigt auch der EuGH in *British Horeseracing Board*, wenn er das Schutzziel der Richtlinie damit beschreibt, dass durch den sui-generis-Schutz ein Anreiz dafür geschaffen werden sollte, Systeme für die Speicherung und Verarbeitung vorhandener Informationen zu errichten, und eben nicht für die Erzeugung neuer Elemente, die dann „später in einer Datenbank zusammengestellt werden können".[381]

Fraglich ist folglich, ob vorliegend neue Elemente erzeugt werden.

aa) Berechnung bzw. Optimierung der Parameter

Die berechneten Parameter bzw. die Baumstruktur könnten möglicherweise als eine kondensierte Verkörperung der relevanten Informationen und Gemeinsamkeiten aus der Gesamtmenge der Trainingsdaten (also Bilder, Texte, …) anzusehen sein. Dann dienen gerade die berechneten bzw. optimierten Werte dazu, die aus den Massen an zur Verfügung stehenden Daten gewonnen Erkenntnisse zu fixieren, also den größtmöglichen Nutzen daraus zu ziehen, ohne dass dabei die ursprüngliche Datenmenge erweitert wird.[382] Es könnte dann von einer „Beschaffung" als tauglichem Investitionsgegenstand ausgegangen werden. *Hetmank/Lauber-Rönsberg* kommen zu dem Ergebnis, dass – sofern die zugrundeliegenden (Trainings-)Daten allgemein zugänglich sind – die Investition in die Analyse derselben durch ML-Modelle dem Schutz nach § 87a UrhG zugänglich sein müsste.[383] Eine begriffliche Abgrenzung zwischen bereits vorhandenen und neu erzeugten Daten trifft *Wiebe*: „Bereits vorhandene Daten sind allgemein verfügbar und können daher grundsätzlich von jedem Dritten mit gleichem Aufwand gesammelt werden, während er-

379 EuGH GRUR 2005, 244 – *British Horseracing Board, BHB-Pferdewetten*; BGH GRUR 2005, 857 – *Hit Bilanz*; Dreier/Schulze–*Dreier*, UrhG, § 87a Rn. 12.
380 Vgl. ErwGr 10 Datenbank-RL.
381 EuGH GRUR 2005, 244, 247 Rn. 31 – *British Horseracing Board, BHB-Pferdewetten*.
382 So wohl auch Ahlberg/Götting–*Vohwinkel*, BeckOK-UrhG, § 87a Rn. 49.
383 *Hetmank/Lauber-Rönsberg*, GRUR 2018, 574, 578.

zeugte Daten 'ihrer Natur nach' niemandem außer dem Datenerzeuger selbst bekannt sind."[384]

Die in der Literatur und Rechtsprechung erkennbare Neigung, den Investitionsbegriff im Kontext des Machine Learning zunehmend weiter auszulegen,[385] ist auch auf EU-Ebene nicht unbemerkt geblieben.[386] In einer im Jahr 2018 veröffentlichten zweiten Evaluierung der Datenbank-RL wird explizit Bezug auf maschinengenerierte Daten genommen – wenngleich eine konkrete Erörterung zu Machine Learning leider ausbleibt. *Machine-Generated Databases* werden gleich zu Beginn als nach allgemeinem Verständnis vom sui-generis-Recht ausgeschlossen bezeichnet.[387] Sodann wird jedoch in Auswertung einer durchgeführten Umfrage und nicht zuletzt auch im Lichte des *Autobahnmaut*-Urteils des BGH[388] zumindest festgestellt, dass – während das Verständnis des Investitionsgegenstands weiter eng zu fassen ist – eine Beobachtung der Situation erforderlich sei.[389] Im Rahmen der durchgeführten Studie wird zudem angezweifelt, ob das sui-generis-Recht bzw. die tatbestandlichen Voraussetzungen noch wirtschaftlich optimal angelegt sind.[390] Wenngleich im Rahmen dieser europäischen Untersuchung also keine eindeutige Tendenz zu einem weiteren Investitionsbegriff zu erkennen ist, bieten die Erörterungen vor dem Hintergrund der gebotenen „Beobachtung der Situation" doch Anlass dazu, einen genaueren Blick auf die Parametersammlung im Machine Learning-Prozess zu werfen.

Nach dem bisher Gesagten muss es wesentlich darauf ankommen, ob die Parametersammlung als Sammlung vorbestehender Informationen oder als Zusammenstellung erzeugter Informationen einzuordnen ist.

Vielleicht ist Machine Learning grundsätzlich als Informationsextrahierungsprozess[391] zu begreifen – dies implizierte, dass keine neuen Informationen bzw. Daten erzeugt, sondern lediglich vorhandene Informationen aus bestehenden Daten extrahiert und in maschinenlesbare Form gebracht wür-

384 *Wiebe*, GRUR 2017, 338, 341; vgl. auch *Leistner*, KuR 9 2007, 457, 460; zweifelnd noch im Jahr 2005 *Sendrowski*, GRUR 2005, 369, 372.
385 Vgl. Fußnote 384 sowie z. B. LG Köln MMR 2002, 689 – *Online-Fahrplanauskunft*.
386 *Europäische Kommission*, Evaluation DB-RL, S. 35, dort Fn. 184.
387 Dies., Evaluation DB-RL, S. 35.
388 BGH GRUR 2010, 1004 – *Autobahnmaut*.
389 *Europäische Kommission*, Evaluation DB-RL, S. 37.
390 Dies., Evaluation DB-RL, S. 40.
391 Dies legen die Definitionen des Machine Learning durchaus nahe, vgl. die Ausführungen zur Abgrenzung zu TDM in § 2 B.IV.1..

den.[392] Dann läge ein nach dem traditionellen Begriffsverständnis tauglicher Investitionsgegenstand vor.[393]

Ordnet man aber – wie eingangs bewusst vorsichtig im Konjunktiv angedeutet – die aus den Daten gewonnenen Informationen als in den Parametern repräsentiert ein, wird verkannt, dass die aufzudeckenden Zusammenhänge für den Fall der *Python*-Modelle gerade nicht nur durch die Parameter erfassbar sind, sondern dass erst im Zusammenspiel mit Hyperparametern und Quellcode bzw. Skriptaufruf eine Verkörperung dieser Erkenntnisse plausibel erscheint. Die Parameter selbst sind nur Zahlenwerte, die jeweils für sich genommen das Gewicht einer durch das Modell geleiteten Information in Bezug auf das Folgeneuron oder den Folgeknoten enthalten. Die dadurch in einem Parameter enthaltene Information (z. B. „gewichte alle eingehenden Werte mit dem Gewicht 0,42") entsteht erst im Laufe des Trainingsprozesses und existiert in dieser Form vorher nicht.

Insofern ist grundsätzlich davon auszugehen, dass es sich bei den Parametern allenfalls um neue, also hergestellte, Datenbankelemente handelt. Die Investition in die Erzeugung in Form der Berechnung bzw. Optimierung derselben ist folglich kein tauglicher Gegenstand des § 87a UhrG.

Noch eindeutiger ist die Situation, wenn ein ML-Modell nicht Zusammenhänge aus vorbestehenden Daten ermittelt, sondern – wie etwa im Fall von sogenannten Generative Adversarial Networks[394] – aus „weißem Rauschen" lernt, ein Bild zu erzeugen. Zwar ist der adversariale Part des Modells ein „klassisch" trainiertes Modell, aber der generative Part optimiert seine Parameter aufgrund des Feedbacks des adversarialen Teils. Die Parameter des generativen Parts sind mithin nicht eine Repräsentation vorhandener Informationen, sondern stehen für durch das Modell im Trial-and-Error-Prozess erlerntes „Wissen". Die gespeicherten Parameterwerte sind mithin als neue, „hergestellte" Informationen einzuordnen.

---

392 Vgl. auch BGH GRUR 2005, 857, 859 – *HIT BILANZ* – die „Feststellung vorhandener Vorgänge" wird als Ermittlung vorhandener Informationen eingeordnet; Schricker/Loewenheim–*Vogel*, Urheberrecht, § 87a Rn. 57 – Abfassung von Zeitungsartikeln könnte als Erfassung von Daten „aus dem Leben" einzuordnen sein; *Wiebe*, GRUR 2017, 338, 341 – für einen Schutz nach § 87a UrhG, wenn lediglich „in der Natur bereits vorhandene Daten" gesammelt werden, die „von jedem Dritten mit gleichem Aufwand gesammelt werden können".

393 Dagegen aber Wandtke/Bullinger–*Hermes*, PK UrhR, § 87a Rn. 41, der grundsätzlich Ergebnisse Big Data-Anwendungen als „Spin-Off"-Datenbanken und damit als nicht schutzfähig einordnet.

394 Vgl. § 2 B.II.3..

Wie bereits angedeutet, ist die Lage für trainierte Parameter von Random Forest-Modellen, die mit *R* erzeugt wurden, anders zu beurteilen. An dieser Stelle sei nochmals an die Abbildung 7.2 erinnert: Ergebnis des Trainingsvorgangs ist in *R* eine Baumstruktur, die in Tabellenform vorliegen kann. Diese Tabelle kann – zwar unter je nach Tabellengröße und Baumtiefe nicht unerheblichem Aufwand – bereits dazu eingesetzt werden, Zusammenhänge aus den Trainingsdaten auszulesen. Sie dokumentiert also Gemeinsamkeiten von Daten und zeigt Informationen auf, die auch ohne Einsatz des ML-Modells – oder unter Einsatz anderer Modelle – aufgedeckt werden können. Mithin spricht vieles dafür, die darin enthaltenen Informationen als bereits „in der Natur“ (bzw. in den Trainingsdaten) vorhanden anzusehen und die entstehende Tabelle lediglich als eine Sammlung und Sichtbarmachung dieser Informationen zu verstehen. Damit kommt für solche Modelle als Investitionsgegenstand auch die Investition in die Erzeugung dieser Ergebnistabelle in Betracht. Allerdings ist hier abzuwarten, wie sich die Rechtsprechung dazu äußert – oder ob seitens der EU noch weitere Klarstellungen erfolgen.

bb) Andere Investitionsgegenstände

Möglicherweise können jedoch andere Investitionsgegenstände, die im Zusammenhang mit der Herstellung der Parameterdatenbank anfallen, berücksichtigt werden. Neben der Beschaffung sind auch die Überprüfung und die Darstellung der Daten berücksichtigungsfähig (§ 87a Abs. 1 S. 1 UrhG). Die Parameter werden in der Regel nicht dargestellt, hier kommt also keine berücksichtigungsfähige Investition infrage. Für die Überprüfung der Parameter gilt Ähnliches wie für die Ergebnisausgabe: Die Überprüfung der Parameter außerhalb des Modells ist nicht relevant, denn erst mit den Hyperparametern können die Parameter überhaupt evaluiert werden. Insofern ist wieder auf das gesamte „trainierte Modell“ abzustellen, und nicht separat auf die Parameter. Ein anderer Investitionsgegenstand kommt also nicht in Betracht.

cc) Zwischenergebnis

Als Investitionsgegenstand kommt nur die Wissensextraktion aus den Trainingsdaten im Rahmen des Trainings von Random Forests in *R* in Betracht. Für ML-Modelle in *Python* kann hier für den isolierten Schutz der Parameter kein tauglicher Investitionsgegenstand festgestellt werden.

b) Wesentlichkeit

Ferner müsste für den Fall der Random Forests in *R* die Investition in die Wissensextraktion – also in das Training – wesentlich sein. Es dürfte sich also nicht um eine „Allerweltsinvestition" handeln. Hier gilt im wesentlichen das Gleiche wie schon für die Wesentlichkeit der Investition in trainierte KNN festgestellt wurde:[395] „Es kommt darauf an", aber für tiefe und aufwendige Modelle ist von der Wesentlichkeit der Investition auszugehen.

3. Ergebnis

Zumindest für trainierte Random Forests in *R* ist ein Schutz im Rahmen des Datenbankherstellerrechtes gem. §§ 87a ff. UrhG denkbar, während der Schutz für trainierte Parameter in ML-Modellen, die unter Einsatz der beschriebenen *Python*-Bibliotheken bzw. Frameworks optimiert bzw. erzeugt wurden, nicht infrage kommt.

4. Bewertung und praktische Relevanz

In der Praxis hat die Übernahme nur der Parameter eines Modells in *Python*, sei es KNN oder Random Forest, keine Relevanz: ohne die dazugehörigen Hyperparameter sind die Informationen nutzlos.

Anders sieht es aus für Random Forest-Baumstrukturen in *R*: Diese sind auch ohne Hyperparameter mittels Datenbankabfragen für Vorhersagen nutzbar, sodass ein Schutzbedarf für die reinen trainierten Parameter besteht. Die Rechte, die dem Datenbankhersteller[396] gem. § 87b UrhG zustehen, umfassen nach § 87b Abs. 1 S. 1 UrhG das ausschließliche Recht zu Vervielfältigung, Verbreitung und öffentlichen Wiedergabe der gesamten Datenbank oder eines wesentlichen Teils davon. In Bezug auf die Verwendung unwesentlicher Teile der Datenbank gesteht § 87b Abs. 1 S. 2 UrhG dem Datenbankhersteller ebenfalls ausschließliche Rechte zu, wenn die Nutzung wiederholt und systematisch erfolgt und einer normalen Auswertung der Datenbank zuwiderläuft oder die berechtigten Interessen des Datenbankherstellers unzumutbar beeinträchtigt. Insbesondere der Vervielfältigungsschutz dürfte in der

395 S. § 7 B.II.3..
396 Vgl. § 7 B.II.5.

Praxis relevant sein, denn eine umfangreiche und sorgfältig optimierte Random Forest-Struktur kann zum Beispiel einem Expertensystem als wertvolle Wissensbasis dienen.

III. Zusammenfassung

Auch für trainierte Parameter kommt ein Schutz nach § 4 Abs. 2 UrhG sowie nach §§ 87a ff. UrhG – in Betracht, sofern es sich um die trainierten Parameter eines Random Forest-Modell in einer *R*-Implementierung handelt. Mithilfe der eingangs beschriebenen *Python*-Frameworks bzw. Bibliotheken erzeugte Parametersammlungen können an dem Schutz jedoch nicht partizipieren.

E. Hyperparameter

Die rechtliche Lage bzgl. der trainierten Parameter und der trainierten Modelle wurde bereits geklärt. Offen ist noch, ob auch die Hyperparameter allein schutzfähig sein könnten. Ein Schutzbedürfnis besteht jedenfalls insofern, als anhand der Hyperparameter und der entsprechenden Trainingsdatenbasis die trainierten Parameter reproduziert werden können. Zudem nimmt die Bedeutung von Hyperparameterkombinationen sowie die Übertragbarkeit auf andere Anwendungskontexte zu: Es ist etwa denkbar, einzelne Schichten eines KNN zu definieren, die zum Beispiel Inputdaten in Form von Bildern besonders gut auf Kanten oder Formen untersuchen können. Selbst wenn die Hyperparameter auf einem Trainingsdatensatz A trainiert wurden, können diese mitunter in andere, umfangreichere Netzstrukturen integriert und für Trainingsdatensatz B eingesetzt werden.[397] Während die trainierten Parameter zumindest im Falle künstlicher neuronaler Netze alleinstehend wenig Wert haben dürften, so muss demzufolge ein Gleiches nicht unbedingt für die Hyperparameter gelten.

Als Schutzmöglichkeiten kommen wieder der Datenbankwerkschutz sowie das Datenbankherstellerrecht in Betracht, Computerprogrammschutz wurde bereits mangels Computerprogrammqualität der Hyperparameter abgelehnt.[398] Ein Schutz als Entwurfsmaterial im Rahmen des Computerprogrammschutzes für das trainierte Modell kommt im Übrigen auch nicht

397 Vgl. z. B. *Goodfellow* et al., Deep Learning Handbuch, S. 363.
398 Vgl. bb).

in Betracht, weil die alleinstehende Hyperparametersammlung zwar von dem Quellcode verwendet wird, aber nicht als Vorbereitung desselben dient. Vielmehr entsteht die Hyperparametersammlung erst mit der konkreten Umsetzung des (Trainings-)Programmcodes und wird dann vom (Produktiv-) Programmcode verwendet.[399]

I. Schutz als Datenbankwerk gem. § 4 Abs. 2 UrhG

1. Datenbank

Die Datenbankqualität gespeicherter Hyperparameter entspricht dem oben bereits Gesagten,[400] es handelt sich um eine Sammlung unabhängiger und einzeln abrufbarer Elemente in Form von Informationen zur Modellstruktur, wie etwa die Größe und Anzahl verwendeter Schichten und eingesetzter Funktionen, sowie oben auch der errechneten Parameter, die schon aus technischer Erforderlichkeit heraus systematisch angeordnet und einzeln mithilfe der Framework- bzw. Bibliotheksfunktionen abrufbar sind. Alleiniger Unterschied ist an dieser Stelle bei der Betrachtung der Hyperparameterdatenbank, dass die Parameter als Sammlungselement nicht mehr berücksichtigt werden. Dies ändert jedoch nichts daran, dass das Vorliegen einer Datenbank anzunehmen ist.

2. Persönliche geistige Schöpfung

Fraglich ist jedoch, ob auch ohne die trainierten Parameter eine geistige Schöpfung des Entwicklers gegeben ist, weil ohne die Parameter die Manifestierung der Idee des Urhebers unvollständig sein könnte. Auch hier muss in der Auswahl oder Anordnung der Elemente die Idee des Entwicklers Ausdruck finden. Wie schon bei den Parametern ist auch die Anordnung der Hyperparameter durch die technischen Gegebenheiten vorgegeben. Es muss also wieder auf die Auswahl der Elemente, also der einzelnen Hyperparameter, ankommen.

399 Gegen einen Schutz als Entwurfsmaterial auch *Hartmann/Prinz*, WRP 12 2018, 1431, 1435 Rn. 38.
400 Vgl. § 7 D.I.1. bzw. § 7 B.I.1..

Diese werden durch den Entwickler so gewählt, dass bei Anwendung des Modells auf die Trainingsdaten die Parameter optimal berechnet werden.

Die Idee des Entwicklers kann sich mithin auch schon in der Wahl der Hyperparameter manifestieren, die im Zusammenspiel mit den Trainingsdaten letztendlich die Parameter berechnen helfen, bevor die Parameter durch den Ablauf des Trainingsprozesses festgelegt sind. Die Hyperparameter sind insofern unabhängig von den trainierten Parametern, als sie letztlich den Grundstock für diese legen (während andersherum die Parameter vollständig von den Hyperparametern und den Trainingsdaten abhängig sind). Wichtig ist an dieser Stelle allerdings die Verbindung mit den Trainingsdaten: verschiedene Trainingsdaten können mitunter zu unterschiedlichen Parametern führen (je größer die Menge an Trainingsdaten, umso weniger relevant ist jedoch eine Veränderung derselben etwa durch Wegnahme einzelner Datenpunkte).

Wenn die isolierte Sammlung von Hyperparametern urheberrechtlich geschützt sein soll, ist jedoch – wie auch in allen anderen Bereichen des Urheberrechts – zu verhindern, dass sich ein solcher Schutz auch auf triviale und in Fachkreisen bekannte, nichtoriginelle Hyperparameterkonfigurationen erstreckt. Zwar dürfen auch nach der Datenbank-RL nicht allzu hohe Anforderungen an die Schöpfungshöhe gestellt werden,[401] jedoch ist eben „nach unten" genug Freiraum für weitere Innovationen zu lassen. Das Schutzbedürfnis besteht insbesondere bei aufwendigen, nicht lediglich zu Lehrzwecken vereinfachten, vielschichtigen und komplexen Modellen. So ist es denkbar, ein Bilderkennungsmodell zu konzipieren, das aus hunderten Schichten besteht, die auf unterschiedlichste Weise konfiguriert sind, um etwa ein Bild auf Farbe, Muster, Körperteile, Helligkeit, Komposition etc. zu untersuchen. In solch einem Fall muss auch die Sammlung der Hyperparameter ohne die errechneten Parameter bereits als schutzfähige Sammlung qualifiziert werden.

3. Ergebnis

Es wird also auf eine Einzelfallentscheidung ankommen müssen, um den Schutz nach § 4 Abs. 2 UrhG für Hyperparameter zu bestimmen. Der Schutz ist jedoch nicht von vorneherein ausgeschlossen.

401 ErwGr. 16 Datenbank-RL.

II. Investitionsschutz gem. §§ 87a ff. UrhG

Auch für Hyperparameter könnten Datenbankherstellerrechte gem. §§ 87a ff. UrhG infrage kommen.

1. Datenbank

Dass die Hyperparameter eine Datenbank im Sinne des § 4 Abs. 2 UrhG darstellen, wurde bereits festgestellt,[402] damit liegen in der Hinsicht auch die Voraussetzungen für § 87a UrhG vor.

2. Wesentliche Investition

Ferner müsste eine wesentliche Investition in die Datenbank getätigt worden sein. Auch hier wird eine Einzelfallbetrachtung erforderlich: Handelt es sich um Hyperparameter, die zu Beginn des Trainings lediglich aus Erfahrungssätzen geschätzt werden, wurden weder Zeit noch Geld in ihre Sammlung gesteckt. Insofern kann keine wesentliche Investition vorliegen. Handelt es sich um die im Rahmen eines Trainingsprozesses erarbeiteten bzw. gesammelten Hyperparameter, müssen die gleichen Erkenntnisse gelten wie sie für trainierte Modelle bereits festgestellt wurden: Der Prozess kann mitunter sehr viele Ressourcen fordern.[403] Es könnte beispielsweise passieren, dass ein Set an Hyperparametern in einem komplexen Modell über mehrere Tage getestet wird, woraufhin sich herausstellt, dass die Hyperparameter nicht optimal gewählt sind. Daraufhin werden die Hyperparameter angepasst und ein erneuter Trainingsvorgang wird gestartet. Dieser Prozess kann sich mehrfach wiederholen, also zeit- und kostenintensiv sein. Es handelt sich dann um wesentliche Investitionen in die Beschaffung und Überprüfung der Datenbankinhalte. Festzustellen ist zudem, dass die Hyperparameter-Sammlung nicht maschinell erzeugt, sondern jeder Wert durch den Menschen gewählt wird. Es handelt sich also um einen gem. § 87a UrhG berücksichtigungsfähigen Investitionsgegenstand. Darüber hinaus können Kosten entstehen, wenn

402 Vgl. § 7 E.I..

403 Vgl. *Graf*, Multitalent für Sprache (Spektrum.de vom 11.08.2020) – die Kosten für das Training von GPT-3 wurden dort etwa mit 5 Millionen Dollar beziffert, zudem werden die CO2-Emissionen betont, die durch die intensive Hochleistungsrechnernutzung entstehen.

etwa ganze Schichten aus anderen bewährten KNN übernommen werden sollen – sofern diese urheberrechtlich geschützt und nicht frei zur Verfügung gestellt worden sind, können Lizenzkosten anfallen. Damit läge eine Investition in die Beschaffung von Daten vor.

3. Ergebnis

Auch eine Sammlung an Hyperparametern kann folglich dem Schutz der §§ 87a ff. UrhG unterfallen.

III. Zusammenfassung

Für gewählte Hyperparameter kommt – unabhängig von der Implementierung in *R* oder *Python* – ein Schutz gem. §§ 4 Abs. 2, Abs. 1 und 87a ff. UrhG in Betracht, wobei jeweils Einzelfallbetrachtungen erforderlich werden hinsichtlich des Vorliegens einer geistigen Schöpfung in Bezug auf das Datenbankwerk sowie hinsichtlich des Vorliegens einer wesentlichen Investition bzgl. § 87a UrhG.

F. Untrainiertes Modell

Zuvor wurde bereits auf den Schutz trainierter Modelle eingegangen.[404] Die Frage des Schutzes „untrainierter“ Modelle liegt also nahe. Fraglich ist hier allerdings bereits, inwiefern diese sich begrifflich von der Struktur bzw. der grundlegenden Architektur der Modelle, also der soeben diskutierten Hyperparametersammlung unterscheiden: Wenn hiermit ein bereits konfiguriertes, aber noch „leeres“ Modell ohne trainierte Parameter zum Beginn bzw. vor Beginn des Trainingsvorgangs gemeint ist, müssen die Aussagen über die Hyperparameter Anwendung finden. Es liegt nahe, das „untrainierte Modell“ wie auch das trainierte Modell[405] als eine Kombination aus Parametern, Hyperparametern und Quellcode aufzufassen, wobei die Parameter lediglich zufällig gewählte Werte sind und sowohl Parameter als auch Hyperparameter nicht notwendig bereits in Dateien abgelegt wurden. Auch das Modell selbst

404 Vgl. § 7 B..
405 Vgl. § 6 D.III..

```
1  # simple neural network with keras
2  from numpy import loadtxt
3  from keras.models import Sequential
4  from keras.layers import Dense
5  # load the dataset
6  dataset = loadtxt('dataset.csv', delimiter=',')
7  # split into input (X) and output (y) variables
8  X = dataset[:,0:8]
9  y = dataset[:,8]
10 # define the keras model
11 model = Sequential()
12 model.add(Dense(12, input_dim=8, activation='relu'))
13 model.add(Dense(8, activation='relu'))
14 model.add(Dense(1, activation='sigmoid'))
15 # compile the keras model
16 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['Accuracy'])
17 # fit the keras model on the dataset
18 model.fit(X, y, epochs=150, batch_size=10, verbose=0)
19 #save the model
20 model.save('my_model.h5')
```

Abbildung 7.3: Beispiel eines einfachen Quellcodes zur Erstellung eines Modells mit Keras, in Anlehnung an https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/, 22.02.2021.

liegt – im untrainierten Zustand – in der Regel nicht in Dateiform vor. Vielmehr wird das untrainierte Modell in Form von Quellcode gegeben sein, der die Modellstruktur definiert, und ggf. am Ende noch Befehle zum Trainieren und Speichern des Modells beinhaltet.

Veranschaulicht wird dies durch Abbildung 7.3. Ergebnis des ausgeführten Codes ist ein gespeichertes trainiertes Modell. Der Code selbst definiert jedoch zu Beginn ein untrainiertes Modell, außerdem liegt vor Ausführung noch kein trainiertes Modell vor, anhand dessen (z. B. durch Interaktion mit dem Nutzer) ein Einsatz etwa zur Klassifikation von Bildern erfolgen könnte. Stattdessen wird das Modell zunächst in Zeile 11 initialisiert, in den Zeilen 12 bis 14 werden die Schichten – also die Hyperparameter, bzw. die Topologie – definiert. Jeder Schicht wird eine Anzahl Neuronen zugewiesen sowie die für jedes Neuron der Schicht zu verwendenden Aktivierungsfunktionen. Das Trainingsdatenset, das in Zeile 6 geladen wurde, enthält Daten (`X`) und Labels (`y`), die dann in Zeile 18 für das Training des Modells dem Modell zugeführt werden.

Als Schutzgegenstand des untrainierten Modells kommt folglich nur der Quellcode infrage, mithin also nur Computerprogrammschutz gem. § 69a UrhG. Dass im Nachhinein ggf. noch Anpassungen – auch an den Hyperparametern – vorzunehmen sind, ist unerheblich. Auch der Code für ein untrainiertes Modell erfüllt bereits einen Zweck, nämlich ein Modell zu trainieren (und

ggf. zu evaluieren – hier nicht im Code-Beispiel enthalten) und enthält Steuerungsanweisungen. Auch die eigene geistige Schöpfung des Entwicklers – nämlich, einen Weg zu finden, ein Modell für die gewählte Trainingsaufgabe programmatisch festzulegen – liegt in dem Quellcode vor. Für den Schutz gem. § 69a UrhG ergeben sich keine Unterschiede zu dem, was bereits über den Schutz des Quellcodes eines trainierten KNN gesagt wurde[406] – ein Schutz als Computerprogramm ist also auch für das untrainierte Modell zumindest in Bezug auf den Quellcode regelmäßig anzunehmen.

406 Vgl. dazu oben aa).

§ 8 Ergebnis des dritten Teils

Der Schutz von Modellen maschinellen Lernens ist, wie gezeigt, komplexer als auf den ersten Blick vermutet und bisher in der rechtswissenschaftlichen Literatur diskutiert. Es ist zu differenzieren zwischen den unterschiedlichen Komponenten der Modelle.[407] Als potenzielle Schutzgegenstände identifiziert wurden trainierte ML-Modelle in *Python*, trainierte Random Forests in *R*, trainierte Parameter, Hyperparameter sowie untrainierte ML-Modelle. Diese wurden in Bezug auf ihre Schutzfähigkeit gem. §§ 4, 87a ff. und 69a UrhG untersucht.

Dabei wurde festgestellt, dass ein trainiertes Modell in *Python* durch die Eigenschaften seiner Komponenten (Hyperparameter- und Parametersammlung, Quellcode) potenziell Schutz sowohl als Datenbankwerk gem. § 4 Abs. 2 UrhG sowie auch als Computerprogramm gem. § 69a UrhG genießt und außerdem ein Datenbankherstellerrechtsschutz gem. §§ 87a ff. UrhG infrage kommt.[408]

Trainierte Parameter der Random Forests unterscheiden sich bei der Implementierung in *R* wesentlich von der Form, in der trainierte Parameter der *Python*-Modelle vorliegen, insofern kommt die Prüfung zu unterschiedlichen Ergebnissen: Trainierten Parametern der ML-Modelle in *Python* kommt weder Schutz als Datenbankwerk, noch im Rahmen des Datenbankherstellerrechtes Schutz zu (zum einen mangelt es an der persönlichen geistigen Schöpfung, zum anderen am tauglichen Investitionsgegenstand). Die trainierten Parameter der Random Forest-Modelle in *R* stellen jedoch zugleich das trainierte Random Forest-Modell dar, weshalb sowohl eine persönliche geistige Schöpfung als auch ein tauglicher Investitionsgegenstand vorliegt.[409]

Isoliert abgespeicherte Hyperparameter lassen sich als Datenbank charakterisieren und dürften im Regelfall sowohl gem. § 4 UrhG als auch gem. §§ 87a ff. UrhG schutzfähig sein.[410]

Für untrainierte Modelle kommt nur ein Computerprogrammschutz gem. § 69a UrhG in Bezug auf den Quellcode infrage.[411]

---

407 Zur Diskussion der Modellbestandteile s. oben § 6.
408 S. oben § 7 B.IV..
409 S. oben § 7 D.
410 S. dazu § 7 E..
411 S. oben § 7 F..

	§ 4 UrhG	§§ 87a ff. UrhG	§ 69a UrhG
Trainiertes Python-ML-Modell	+	+	+
Trainierter Random Forest in *R* (RF)	*Verweis auf trainierte Parameter*		
Trainierte Parameter	- / + (RF)	- / + (RF)	n/a
Hyperparameter	+	+	n/a
Untrainiertes Modell	n/a	n/a	+

Abbildung 8.1: Übersicht über die Ergebnisse für den Modellschutz, jeweils unter Vorbehalt einer Einzelfallbetrachtung. Quelle: eigene Darstellung.

Abbildung 8.1 stellt die Ergebnisse im Überblick dar. Ohne Schutz dürften nach dem Resultat dieser Prüfung nur gänzlich banale Modelle bleiben, ansonsten steht insbesondere der Datenbankschutz im Vordergrund.