Topologieoptimierung mittels Deep Learning ohne voroptimierte Trainingsdaten

Topology Optimization by Means of Deep Learning Without Pre-Optimized Training Data

Alex Halle[1]*, L. Flavio Campanile[1], Alexander Hasse[1]

[1] Professorship Machine Elements and Product Development, Chemnitz University of Technology

** Korrespondierender Autor:*
Alex Halle
Chemnitz University of Technology
Reichenhainer Straße 70 | R. 2/A312 (neu: C21.312)
09126 Chemnitz
Telefon: +49 371 531-30225
Mail: alex.halle@mb.tu-chemnitz.de

Abstract

Here a method for topology optimization is presented which is able to obtain optimized geometries without iterative optimum search. The optimized geometries are provided by an artificial neural network, the predictor, on the basis of boundary conditions and degree of filling as input data. In the training phase, geometries generated on the basis of random input data are evaluated with respect to given criteria and the results of those evaluations flow into an objective function which is minimized. Other than in state-of-the-art procedures, no pre-optimized geometries are used during training.

The trained predictor supplies geometries which are similar to the ones generated by conventional topology optimizers, but requires only a small fraction of the computational effort.

Keywords

deep learning, topology optimization, artificial neural networks, AI in design

 | https://doi.org/10.35199/dfx2020.11

1. Einführung

Bei der Topologieoptimierung (TO) wird die Materialverteilung über einen gegebenen Konstruktionsraum mit der Minimierung einer bestimmten Zielfunktion als Ziel unter Erfüllung vorgegebener Randbedingungen optimiert [1]. In den meisten Fällen wird das Optimierungsproblem mathematisch mit Hilfe eines geeigneten Suchalgorithmus gelöst. Der vorliegende Beitrag befasst sich mit der Lösung von TO-Problemen mit Hilfe der Künstlichen Intelligenz (KI). Der Stand der Forschung in diesem Bereich leifern eine KI, die benötigt wird, um optimale Strukturen zu erzeugen, die auf einer durch konventionelle TO gewonnenen Datenbasis trainiert wird. Daher unterliegen diese Methoden mehreren Einschränkungen, wie z.B. großem Rechenaufwand und problematischer Handhabung von multimodalen Formulierungen. Der hier vorgeschlagene Ansatz zielt darauf ab, diese Nachteile zu beseitigen, indem das für die Optimierung erforderliche AI-Wissen während der Lernphase generiert wird, ohne dass man sich auf voroptimierte Ergebnisse stützen muss.

1.1. Topologie Optimierung

In dieser Arbeit wird nur der Fall der Monomaterial-Topologieoptimierung betrachtet. Das Material, aus dem die Struktur aufgebaut werden soll, ist dann eine Konstante des Problems und die Geometrie wird als unbekannt betrachtet.

Im Falle einer Steifigkeitsoptimierung wird typischerweise die globale mittlere Nachgiebigkeit (fortan als Nachgiebigkeit bezeichnet) als skalares Maß für die zu minimierende Funktion gewählt, unter der Bedingung, dass eine bestimmte Materialmenge im Konstruktionsraum verwendet wird, ausgedrückt in Prozent der maximal möglichen Materialmenge (Füllgrad).

Die Minimierung der Nachgiebigkeit führt zu einer Maximierung der Steifigkeit. Der verfügbare Konstruktionsraum sowie die statischen und kinematischen Randbedingungen für die betrachteten Lastfälle werden typischerweise als Restriktionen betrachtet.

Dieser Beitrag konzentriert sich ebenfalls auf die Steifigkeitsoptimierung, obwohl die vorgestellte Methode von allgemeiner Gültigkeit ist und auf die Optimierung mit verschiedenen Zielfunktionen und Restriktionen angewendet werden könnte.

Es gibt zahlreiche mögliche Ansätze für TO [1]. Bei dem Ansatz "Solid Isotropic Materialwith Penalization" (SIMP) nach Bendsøe [2] wird der Konstruktionsraum in Elemente unterteilt. Für jedes dieser Elemente wird der Beitrag zur Gesamtsteifigkeit der Struktur mit einem zu bestimmenden Faktor skaliert (Dichte).

Der SIMP-Ansatz ist in der Lage, durch einen iterativen Prozess für viele praktische Fälle optimierte Geometrien bereitzustellen. Jede Iteration beinhaltet rechenintensive Operationen: Die kritischsten sind die Zusammenstellung der Steifigkeitsmatrix und die Lösung der Systemgleichung. Wenn Restriktionen vorliegen, wie z.B. Spannungsrestriktionen, erhöht sich die Komplexität des Optimierungsproblems [3], [4].

1.2. Künstliche Neuronale Netzwerke

Deep Learning (DL, Deutsch: Tiefes Lernen) und Künstliche Neuronale Netze (KNN) gehören beide zum Bereich des maschinellen Lernens (ML), der wiederum der Künstlichen Intelligenz (KI) zugeordnet wird. KNN sind in der Lage, komplexe Vorgänge zu lernen und auszuführen, was in den letzten Jahren zu bemerkenswerten Ergebnissen geführt hat. Beispielsweise sind KNN in der Lage, die auf Bildern dargestellten Objekte anhand ihrer Form und Farbe zu erkennen oder Weltmeister im Brettspiel "Go" [5], [6] zu schlagen.

KNN sind in der Lage, komplexe Prozesse zu erlernen und dann die Prozessergebnisse in Abhängigkeit von den Eingaben mit hoher Genauigkeit zu reproduzieren. Die Ausgabe eines KNN wird im Folgenden als Vorhersage bezeichnet. Einige grundlegende Informationen zu

KNN sind im Abschnitt 2.1 beschrieben. Weitere Einzelheiten zum Lernen eines KNN können in Fachliteratur gefunden werden, zum Beispiel [7], [8].

Die Entwicklung von DL oder KNN schreitet stetig voran, zum einen durch die immer besser verfügbare hohe Rechenleistung und zum anderen durch die Entdeckung neuer Möglichkeiten zur Verbesserung des Lernverfahrens.

1.3. DL basierende Topologieoptimierung

Aus den beiden genannten Bereichen ergibt sich die Frage, ob die Vorteile von DL mit der Anwendbarkeit von TO kombiniert werden können. DL-basiertes TO zielt darauf ab, durch Vorhersage der Geometrie mittels KNN optimierte Ergebnisse in nur einem Bruchteil der Zeit zu liefern, die bei herkömmlicher Optimierung benötigt wird, indem der rechenintensive Teil auf den Trainingsalgorithmus verlagert wird, der nur einmal ausgeführt wird. Die von dem trainierten KNN gelieferten Ergebnisse können dann direkt verwendet, mit konventionellen Methoden verfeinert oder an die gewünschte Strukturgröße angepasst werden.

Es gibt bereits einige Versuche in diesem Bereich. So verwenden [9–11] viele tausende topologieoptimierte Geometrien als Trainingsdatensätze für die KNN. In [12] wurde ein anderer Ansatz entwickelt, bei dem Zwischenergebnisse konventioneller TO (das Ergebnis einer begrenzten Anzahl von Optimierungsiterationen konventioneller TO – inklusive der darin ermittelten Gradienten) als Trainingsdatensätze für den KNN verwendet werden. Hier wurden 10.000 Objekte mit jeweils 100 Optimierungsiterationen verwendet. Ein ähnlicher Ansatz wie [12] wird von [13] verwendet, jedoch erweitert auf den Einsatz in 3D.

Obwohl solche DL-Topologieoptimierungsverfahren in der Lage sind, die oben erwähnte Aufgabe einer schnellen und direkten Generierung optimierter Geometrien zu erfüllen, unterliegen die Vorhersagen einigen Beschränkungen.

Da topologieoptimierte Trainingsdaten verwendet werden und die Generierung dieser Daten mit konventionellen Methoden sehr zeitaufwendig ist, ist die Anzahl der zu berücksichtigenden Trainingsdatensätze begrenzt. Im Fall von [9] wurden 100.000 Datensätze generiert und dessen Generierung dauerte etwa 200 Stunden. Weitere 8h wurden für das Training benötigt. Diese Begrenzung wirkt sich negativ auf die Genauigkeit bei der Vorhersage unbekannter Geometrien (d.h. Geometrien, die im Rahmen des Trainings nicht verwendet wurden) aus. So sind z.B. in [9] ca. 3,4% der generierten Geometrien mehrteilig (mit theoretisch unendlich großer Nachgiebigkeit) und daher nicht verwendbar.

Aus diesem Grund ergibt sicht die Frage: Ist das Training eines KNN, zum erzeugen von topologieoptimierten Geometrien, ohne den Einsatz von topologieoptimierten Datensätzen möglich?

In diesem Beitrag wird die vom Stand der Technik abweichende Möglichkeit untersucht, die die Generierung von Trainingsdatensätzen und das Training selbst in einem einzigen Verfahrensschritt zusammengeführt.

Dadurch ist es möglich, eine viel größere Menge an Datensätzen für das Training in einer viel kürzeren Zeit zu verarbeiten. Und da die Nachgiebigkeit während des Trainings berechnet wird, lernt das KNN unerwünschte Ergebnisse zu vermeiden.

Die Verfahren nach dem Stand der Technik erfordern die Verwendung einer großen Anzahl optimierter Datensätze. Diese Datensätze müssen optimal sein, um als Trainingsdaten geeignet zu sein. Je nach Optimierungsformulierung ist dies möglicherweise nicht der Fall, da lokale Minima und Konvergenzprobleme auftreten können. Ein Verfahren, das keine optimierten Datensätze verwendet, unterliegt diesen Beschränkungen nicht.

2. Methode

Die vorgestellte Methode basiert auf einer KNN-Architektur namens Predictor-Evaluator-Network (PEN), die von den Autoren zu diesem Zweck entwickelt wurde. Der Prädiktor ist der

trainierbare Teil des PEN und hat die Aufgabe, auf der Basis von Eingabedatensätzen optimierte Geometrien zu generieren.

Wie bereits erwähnt, werden im Gegensatz zu den oben erwähnten Stand der Technik Methoden keine voroptimierten, topologieoptimierten Datensätze im Training verwendet. Die für das Training verwendeten Geometrien werden vom Prädiktor selbst auf der Basis zufällig generierter Eingabedatensätze erstellt und von den übrigen Komponenten des PEN, den sogenannten Evaluatoren, ausgewertet.

Die Evaluatoren führen mathematische Operationen durch, die konventionell oder in Form eines KNN implementiert sind. Im Gegensatz zum Prädiktor sind die von den Evaluatoren durchgeführten Operationen vordefiniert und ändern sich während des Trainings nicht.

Jeder Evaluator bewertet die Ergebnisse des Prädiktors in Bezug auf ein bestimmtes Kriterium und gibt einen entsprechenden skalaren Wert als Maß für die Erfüllung des Kriteriums zurück. Während des Trainings wird eine Skalarfunktion der Evaluatorausgaben (Zielfunktion J - siehe Abschnitt 2.6), die für eine Gruppe von Geometrien (Batch) berechnet wurde, durch Änderung der trainierbaren Parameter des Prädiktors minimiert. Auf diese Weise lernt der Prädiktor, wie optimierte Geometrien erzeugt werden können.

Der Prädiktor, die einzelnen Evaluatoren, ihre Aufgaben und ihre Arbeitsweise werden in den folgenden Abschnitten ausführlich erläutert.

2.1. Grundlegende Definitionen

2.1.1. Bezüglich Topologieoptimierung

Bei der Topologieoptimierung wird der Konstruktionsraum in der Regel durch geeignete Vernetzung in Elemente unterteilt. In Abbildung 1 werden Elemente (wobei ein Element schraffiert ist) und Knoten visualisiert.

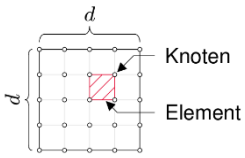


Abbildung 1: Element und Knoten

In dieser Arbeit werden nur Quadratgitter mit gleicher Anzahl von Zeilen und Spalten verwendet. Die Gesamtanzahl der Elemente beträgt d^2, wobei d die Anzahl der Zeilen oder Spalten ist (siehe Abbildung 1). Die d^2 Designvariablen $\mathrm{X_i}\ \{\mathrm{i} = 1 \ldots \mathrm{d}^2\}$ sind die oben eingeführten Dichten, welche die Beiträge der einzelnen Elemente zur Steifigkeitsmatrix skalieren. Die Dichte hat den Wert Eins, wenn der Steifigkeitsbeitrag des Elements vollständig erhalten bleibt, und Null, wenn er verschwindet.

Die Dichtewerte werden in einem Vektor $\mathbf{X}$ gesammelt, der mit Hilfe des Operators $\mathcal{R}_{2d}$ in eine quadratische Matrix $\mathbf{X_M}$ der Ordnung d transformiert werden kann.

Um mögliche Singularitäten der Steifigkeitsmatrix zu verhindern, wird ein unterer Grenzwert für die Einträge von $\mathbf{X}$ gesetzt [2], so dass $0 < X_{min} < X_i \leq 1, \{i = 1 \ldots d^2\}$ gilt.

Obwohl eine binäre Auswahl der Dichte erwünscht ist (Material vorhanden/nicht vorhanden), müssen aus algorithmischen Gründen Werte zwischen Null und Eins zugelassen werden. Um der gewünschten binären Auswahl der Dichten näher zu kommen, wird die sogenannte Bestrafung eingesetzt. Die Bestrafung wird durch eine elementweise Potenzierung der Dichten durch den Bestrafungsexponenten $p > 1$ [14] realisiert.

Das arithmetische Mittel aller X_i definiert den Füllgrad der Geometrie M_{is}. Der Zielwert M_{tar} ist der Füllgrad, der durch den Prädiktor erreicht werden soll.

Untersuchungen zeigten, dass die Trainingsgeschwindigkeit für hochauflösende Geometrien erhöht werden konnte, indem das Training in Stufen mit zunehmender Auflösung aufgeteilt wird. Da kleinere Geometrien um mehrere Größenordnungen schneller trainiert werden und das gewonnene Wissen auch für Geometrien mit höherer Auflösung genutzt werden kann, verringert sich die Gesamttrainingszeit im Vergleich zum Training mit hochauflösenden Geometrien. Höhere Level werden aus niedrigeren Leveln durch Hinzufügen verborgener und gefalteter Schichten erreicht. Die Ebenen sind mit der ganzen Zahl Λ gekennzeichnet.

Eine Erhöhung von Λ um 1 führt zu einer Verdoppelung der Anzahl d der Zeilen oder Spalten des Gitters des Konstruktionsraums. Dies geschieht durch Vierteln der Elemente der vorherigen Ebene. Auf diese Weise werden die Knoten der vorherigen Ebene in der neuen Ebene beibehalten. Die Anzahl der Zeilen oder Spalten auf der ersten Ebene wird als d_{inp} bezeichnet.

Die Eingabedaten des Prädiktors umfassen die statischen und kinematischen Randbedingungen sowie den angestrebten Füllgrad M_{tar}. Die Ausgabe des Prädiktors ist eine Geometrie $\mathbf{X}$. Eingabedaten können nur im ersten Level definiert werden und ändern sich nicht, während der Level geändert wird. Daher können neue Knoten nicht statischen oder kinematischen Randbedingungen zugeordnet sein. Der Ebenenwechsel erfolgt, nachdem eine bestimmte Abbruchbedingung, die später beschrieben wird, erfüllt ist.

2.1.2. Bezüglich Künstlicher Neuronale Netze

KNN oder genauer gesagt Feedforward Neuronale Netze bestehen aus hintereinander geschalteten Schichten, die wiederum sogenannte Neuronen [15] enthalten. Ein Neuron ist das Grundelement eines ANN. Die Kombination aller Schichten wird auch als Netz bezeichnet.

Das Neuron erhält Eingaben, die elementweise mit den jeweiligen Gewichten multipliziert (Matrixmultiplikation), summiert, zu einer Konstante (Bias) addiert und als Argument an eine sogenannten Aktivierungsfunktion übergeben werden.

Es ist üblich, dass mehrere Neuronen der gleichen Schicht die gleichen Eingaben haben. Da jedes Neuron einen einzigen Ausgang hat, liefert jede Schicht mit einer gewissen Anzahl an Neuronen auch die gleiche Anzahl an Ausgaben. Die Ausgänge einer Schicht (außer für der letzten) dienen als Eingänge für die folgende Schicht. Die erste Schicht wird als Eingabeschicht und die letzte Schicht als Ausgabeschicht bezeichnet. Jede Schicht, deren Eingabe- und Ausgabewerte für den Benutzer nicht zugänglich sind, wird als verborgene Schicht bezeichnet. Die Anzahl der Schichten wird auch als Tiefe des Netzes bezeichnet, die auch das Attribut "tief" im Begriff "Deep Learning", der im Allgemeinen für Netze mit mehreren verborgenen Schichten verwendet wird, entstehen lassen. Durch das Vorhandensein mehrerer Schichten ist es möglich, ein komplexeres Übertragungsverhalten zwischen der Eingangs- und der Ausgangsschicht abzubilden (siehe Abschnitt 2.2).

Der durch den KNN realisierte funktionale Zusammenhang hängt von den Gewichten von den Bias ab, die im Rahmen des so genannten Trainings oder Lernens nach bestimmten Algorithmen angepasst werden. Der verwendete Lernalgorithmus besteht in der gradientenbasierten Minimierung eines als Fehler bezeichneten skalaren Wertes, der als Norm der Abweichungen der Sollergebnisse von den Istergebnissen erhalten wird. Die Werte, die während des Trainings verändert werden können, werden als trainierbare Parameter bezeichnet. Die Werte, die die Architektur des Netzwerks beschreiben und sich während des Trainings nicht ändern, wie die Anzahl der Neuronen in einer Schicht, werden als Hyperparameter bezeichnet.

Als Erweiterung zu den verborgenen Schichten gibt es die Faltungsschicht. Diese Schichten verwendet die Faltung anstelle der Matrixmultiplikation. Dieses Verfahren ist effizient für gitterartige Datenstrukturen und wird daher für viele moderne Bildanwendungen [16] verwendet.

2.2. Prädiktor

Der Prädiktor ist für die Erstellung einer optimierten Geometrie für einen gegebenen Eingabedatensatz zuständig. Seine ANN-Architektur besteht aus mehreren verborgenen Schichten, Faltungsschichten und einer Ausgabeschicht mit d^2-Neuronen.

Die Architektur des Prädiktors ist in Abbildung 2 in vereinfachter Form dargestellt.

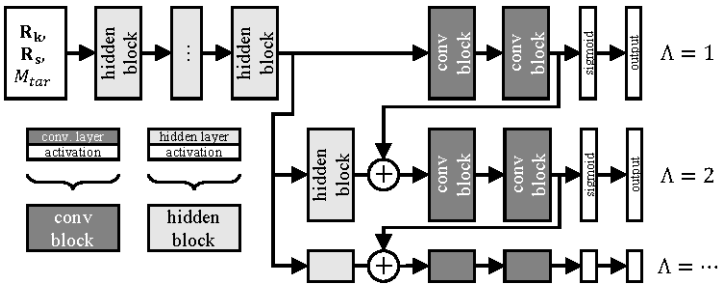


Abbildung 2: Vereinfachte Darstellung der Architektur des Prädiktors

Hier ist der Datenfluss durch den Prädiktor sowie die Addition von zusätzlichen Schichten bei der Erhöhung des Levels Λ zu sehen. Ein Eingabedatensatz (oben links) wird von mehreren aufeinanderfolgenden verborgenen Schichten verarbeitet und dann an zwei aufeinanderfolgende Faltungsschichten weitergegeben. Der letzte Schritt im ersten Level ist die Erzeugung der Ausgangsdaten im gewünschten Intervall durch die Sigmoid-Funktion [17].

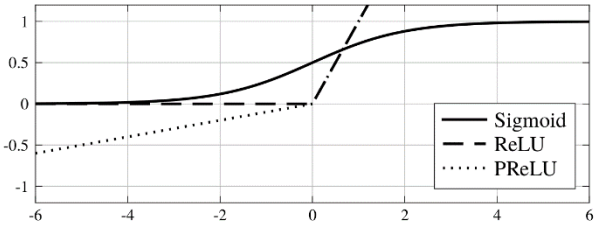


Abbildung 3: Sigmoid-, ReLU- und PReLU-Funktion

Für nachfolgende Ebenen werden die Ausgaben des letzten Faltungsblocks der vorhergehenden Schicht und die Ausgaben des letzten verborgenen Blocks der ersten Ebene addiert und dann, nach einem zusätzlichen Faltungsblock, in die gewünschte Ausgabedimension umgewandelt.

2.3. Evaluator: Nachgiebigkeit

Die Aufgabe des Evaluators für die Nachgiebigkeit ist die Berechnung der Nachgiebigkeit. Zu diesem Zweck verwendet er einen ähnlichen Algorithmus wie den von Sigmund in „A 99 line topology optimization code written in Matlab“ [14] beschrieben. Die Nachgiebigkeit

$$c = \mathbf{U}^T\mathbf{K}\mathbf{U} = \mathbf{U}^T\mathbf{F} \tag{1}$$

wird mithilfe der Steifigkeitsmatrix $\mathbf{K}$, dem Kraftvektor $\mathbf{F}$ und dem Verschiebungsvektor $\mathbf{U}$ berechnet. Die Steifigkeitsmatrix hängt linear von der Geometrie $\mathbf{X}$ ab und wird ausgedrückt durch

$$\mathbf{K} = \sum_{\mathrm{i}=1}^{\mathrm{d}^2} X_i^p \mathbf{K}_i \tag{2}$$

wobei p den Bestrafungsexponenten und $\mathbf{K}_i$ die unskalierten Beiträge der einzelnen Elemente darstellt.

2.4. Evaluator: Füllgradabweichung

Die Aufgabe dieses Evaluators ist es, die Abweichung des Füllgrades M_{is} vom Zielwert M_{tar} zu bestimmen und als skalare Größe M an die Zielfunktion zu übergeben. Durch die Berücksichtigung der Abweichung des Füllgrades in der Zielfunktion wird der Prädiktor in dem Maße bestraft, wie er von dem angestrebten Füllgrad abweicht.

2.5. Evaluator: Filter

Der Filter-Evaluator sucht nach Schachbrettmustern in der Geometrie und gibt einen skalaren Wert C im Intervall $[0, 1]$ aus, der auf die Menge und das Ausmaß der entdeckten Schachbrettmuster hinweist. Diese Schachbrettmuster bestehen aus abwechselnd hohen und niedrigen Dichtewerten der Geometrie. Sie sind unerwünscht, weil sie nicht die optimale Materialverteilung widerspiegeln und schwer auf reale Teile übertragbar sind [18].

Mehrere Lösungen für das Schachbrettproblem wurden im Rahmen der konventionellen Topologieoptimierung [19] entwickelt. In diesem Werk wurde eine neue Strategie gewählt, die es erlaubt, den Schachbrettfilter in die Qualitätsfunktion einzubeziehen. Im vorliegenden Ansatz werden Schachbrettmuster zwar zugelassen, aber erkannt und entsprechend bestraft.

2.6. Qualitäts- und Zielfunktion

Die Qualitätsfunktion kombiniert alle Evaluatorwerte zu einem Skalar.

$$f_Q = (c + 1) \cdot (5M + 1) \cdot (C + 1) \tag{3}$$

Da eine Optimierung auf der Basis von Einzelgeometrien großen Rechenaufwand erfordert und zu Instabilitäten des Trainingsprozesses führen würde (große Sprünge der Zielfunktionsausgabe), wird eine vorgegebene Anzahl von Geometrien b_n (Batch, Deutsch: Stapel) generiert und die entsprechenden Qualitätsfunktionen zu einem skalaren Wert zusammengefasst, der als Zielfunktion für die das Training bestimmende Optimierung fungiert.

$$J = \frac{1}{b_n} \sum_{i=1}^{b_n} f_{Q\,i} \tag{4}$$

2.7. Training

Innerhalb eines Stapels werden die Eingabedatensätze zufällig generiert und der Prädiktor erzeugt die entsprechenden Geometrien $\mathbf{X}_i$. Anschließend wird die Qualitätsfunktion aus den Ausgaben der Evaluatoren gemäß Gleichung (3) berechnet. Die Zielfunktion J wird dann für den kompletten Stapel berechnet. Anschließend wird der Gradient der Zielfunktion in Bezug auf die trainierbaren Parameter berechnet. Die trainierbaren Parameter des Prädiktors für den nächsten Stapel werden dann nach dem Kriterium des steilsten Gefälles angepasst, um den Wert der Zielfunktion zu verringern.

Sobald der Level steigt, gibt der Prädiktor eine Geometrie mit höherer Auflösung aus. Es ist wichtig zu betonen, dass die PEN-Methode im Gegensatz zur herkömmlichen Topologieoptimierung nicht die Dichtewerte der Geometrie, sondern die Gewichtungen des Prädiktors optimiert.

3. Anwendung

3.1. Implementierung

Die Umsetzung der vorgestellten Methode erfolgt unter Verwendung der Programmiersprache Python. Zum Einsatz kommt das Framework Tensorflow mit der Keras Programmierschnittstelle (API), die sich gut zur Programmierung von DL-Algorithmen in

Python eignet. Tensorflow wird von Google entwickelt und ist eine Open-Source-Plattform für die Entwicklung von Anwendungen für maschinelles Lernen [17].

Die Topologie des Prädiktors mit allen Schichten und allen Hyperparametern ist in Abbildung 4 dargestellt. Die gewählten Hyperparameter erwiesen sich nach zahlreichen Tests, in denen die Unterschiede zwischen den Vorhersagen und den durch konventionelle TO erhaltenen Ergebnissen untersucht wurden, als die besten. Die Hyperparameter werden durch die Gestalt (numerischer Ausdruck über dem Pfeil, der vom Block wegführt) der Ausgangsmatrix eines Blocks oder durch den Kommentar in der Nähe des Faltungsblocks angezeigt. Die Beschriftung des Ausgabepfeils beschreibt die Dimensionen des Ausgabevektors oder der Ausgabematrix. Die Namen der Elemente in Abbildung 4 z.B. „Conv2D“, entsprechen den in Keras verwendeten Namen für ANN-Schichten.

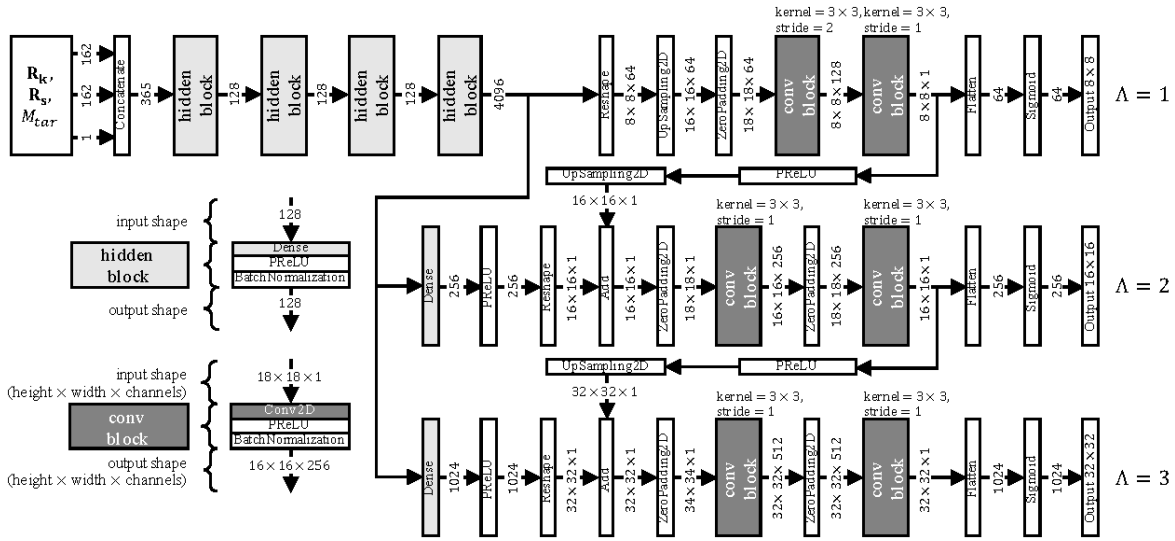
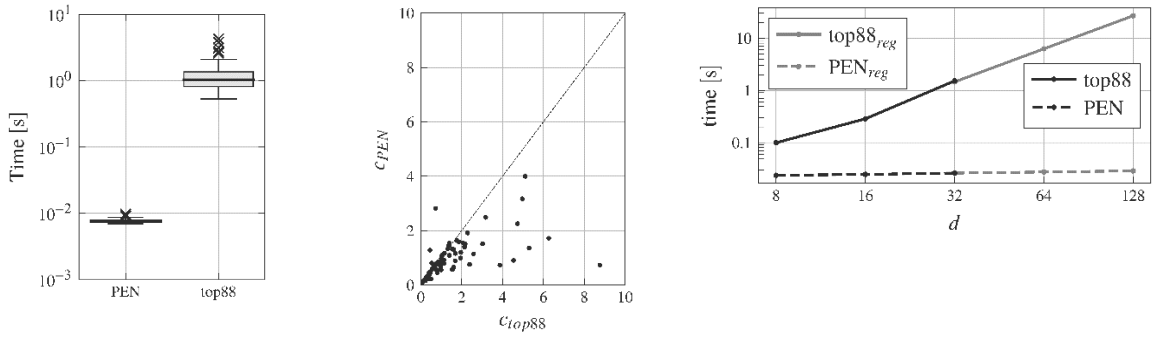


Abbildung 4: Topologie des Prädiktors

3.2. Ergebnisse

Die Ergebnisse wurden mit 100 zufällig generierten Eingabedatensätzen validiert. Diese Eingabedatensätzen waren nicht Teil der Trainingsdatensätze. Sie wurden ledeglich genutzt um mittels konventionellen Topologieoptimierer [20], der auf dem Algorithmus von Andreassen [21] „Efficient topology optimization in MATLAB using 88 lines of code“ (top88) basiert, Validierungsgeometrien zu erzeugen. Im Durchschnitt kann die PEN-Methode in etwa 8 ms nahezu das gleiche Ergebnis liefern wie die konventionelle Methode, während top88 im Durchschnitt 1,2 s benötigt und somit rund 150 mal langsamer ist (siehe Abbildung 5a). Bei höheren Geometrieauflösungen wäre der Unterschied in der Rechenzeit noch größer (siehe Abbildung 5c). Es ist auch zu erkennen, dass die Mehrzahl der von PEN erzeugten Geometrien eine Nachgiebigkeit aufweisen, die kleiner oder gleich der mittels top88 errechneten ist, siehe Abbildung 5b.



(a) (b) (c)

Abbildung 5: Vergleich von PEN und konvetionellen Topologieoptimierer

Die Beispiele in Abbildung 6 zeigen die gute Leistungsfähigkeit der Methode sowie einige Schwächen des Prädiktors. Beispielsweise sind die Geometrien in einigen Fällen verrauscht und enthalten unerwünschte Elemente, die nicht zur Steifigkeit beitragen (siehe Abbildung 6, Spalte zwei oder vier). Dies kann durch eine geeignete Wahl von Hyperparametern des Prädiktors und durch Anpassung der Qualitätsfunktion verbessert werden. In dieser Abbildung ist auch eine Reihe von konventionell topologieoptimierten Geometrien mit verschiedenen Parametern enthalten. Die Nachgiebigkeit der einzelnen Beispielgeometrien ist unten aufgeführt.

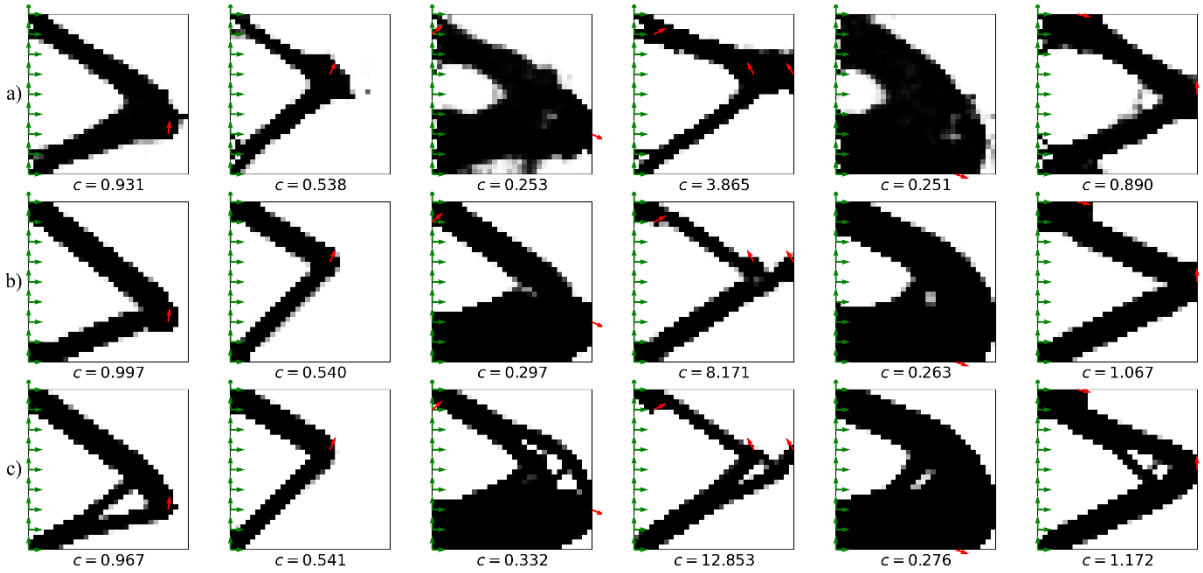


Abbildung 6: Beispielgeometrien a) PEN, b) top88, c) top88 mit $r_{min} = 1{,}5$ und $p = 3$

4. Zusammenfassung

In diesem Beitrag wurde eine Methode vorgestellt, die es ermöglicht, einen Topologie-Optimierer durch Deep Learning zu realisieren. Das für die Generierung topologieoptimierter Geometrien zuständige KNN benötigt für das Training keine voroptimierten Datensätze. Die generierten Geometrien sind in den meisten Fällen den Ergebnissen der konventionellen Topologieoptimierung nach Sigmund oder Andreassen sehr ähnlich.

Dieser Topologieoptimierer zeichnet sich durch seine Geschwindigkeit aus, da der rechenintensive Teil in das Training verlagert wird. Nach dem Training ist der Deep Learning basierte Topologieoptimierer in der Lage, Geometrien zu liefern, die nahezu identisch mit den von herkömmlichen Topologieoptimierern erzeugten sind. Das wird durch die Verwendung eines neuen Ansatzes, des Predictor-Evaluator-Network-Ansatzes (PEN), erreicht. PEN besteht aus einem trainierbaren Prädiktor, der für die Erzeugung von Geometrien zuständig

ist, und nicht trainierbaren Evaluatoren, die den Zweck haben, die Ausgabe des Prädiktors während des Trainings zu bewerten.

Die Ergebnisse der PEN-Methode sind mit der konventionellen Methode vergleichbar. Allerdings könnte sich die PEN-Methode bei der Handhabung von Anwendungen und Optimierungsproblemen höherer Komplexität, wie z.B. Spannungsbegrenzungen, nachgiebige Mechanismen und vielem mehr als überlegen erweisen. Diese Erwartung beruht auf der Tatsache, dass keine optimierten Daten benötigt werden. Alle Methoden, die voroptimierte Daten verarbeiten, leiden unter den Schwierigkeiten, die bei der konventionellen Optimierung beim Umgang mit den oben genannten Problemen auftreten.

Bislang wurden variable kinematische Randbedingungen nicht getestet. Das soll in der zukünftigen Forschung zusammen mit der Verbesserung der Auflösung, der Anwendung auf dreidimensionale Konstruktionsräume und der Berücksichtigung von Nichtlinearitäten und Restriktionen geschehen.

Literaturverzeichnis

[1] SIGMUND, OLE ; MAUTE, KURT: Topology optimization approaches: A comparative review. In: *Structural and Multidisciplinary Optimization* Bd. 48 (2013), Nr. 6, S. 1031–1055

[2] BENDSØE, MARTIN P. ; SIGMUND, O.: *Topology optimization: theory, methods, and applications*. Berlin ; New York : Springer, 2003 — ISBN 978-3-540-42992-0

[3] PICELLI, R. ; TOWNSEND, S. ; BRAMPTON, C. ; NORATO, J. ; KIM, H. A.: Stress-based shape and topology optimization with the level set method. In: *Computer Methods in Applied Mechanics and Engineering* Bd. 329 (2018), S. 1–23

[4] LEE, EDMUND: *Stress-constrained Structural Topology Optimization with Design-dependent Loads*, Thesis, 2012

[5] DEEPMIND: *AlphaGo: The story so far*. URL https://deepmind.com/research/case-studies/alphago-the-story-so-far. - abgerufen am 2019-09-23. — Deepmind. — accessed 23.09.2019

[6] REDMON, JOSEPH ; DIVVALA, SANTOSH ; GIRSHICK, ROSS ; FARHADI, ALI: You Only Look Once: Unified, Real-Time Object Detection. In: *arXiv:1506.02640 [cs]* (2015)

[7] GOODFELLOW, IAN ; BENGIO, YOSHUA ; COURVILLE, AARON: *Deep learning*, *Adaptive computation and machine learning*. Cambridge, Massachusetts : The MIT Press, 2016 — ISBN 978-0-262-03561-3

[8] BASHEER, I.A ; HAJMEER, M: Artificial neural networks: fundamentals, computing, design, and application. In: *Journal of Microbiological Methods* Bd. 43 (2000), Nr. 1, S. 3–31

[9] YU, YONGGYUN ; HUR, TAEIL ; JUNG, JAEHO ; JANG, IN GWUN: Deep learning for determining a near-optimal topological design without any iteration. In: *Structural and Multidisciplinary Optimization* Bd. 59 (2019), Nr. 3, S. 787–799

[10] RAWAT, SHARAD ; SHEN, M.-H. HERMAN: A Novel Topology Optimization Approach using Conditional Deep Learning. In: *arXiv:1901.04859 [cs, stat]* (2019)

[11] ZHANG, YIQUAN ; CHEN, AIRONG ; PENG, BO ; ZHOU, XIAOYI ; WANG, DALEI: A deep Convolutional Neural Network for topology optimization with strong generalization ability. In: *arXiv:1901.07761 [cs, stat]* (2019)

[12] SASAKI, HIDENORI ; IGARASHI, HAJIME: Topology Optimization Accelerated by Deep Learning. In: *IEEE Transactions on Magnetics* Bd. 55 (2019), Nr. 6, S. 1–5

[13] BANGA, SAURABH ; GEHANI, HARSH ; BHILARE, SANKET ; PATEL, SAGAR ; KARA, LEVENT: 3D Topology Optimization using Convolutional Neural Networks. In: *arXiv:1808.07440 [physics, stat]* (2018)

[14] SIGMUND, O.: A 99 line topology optimization code written in Matlab. In: *Structural and Multidisciplinary Optimization* Bd. 21 (2001), Nr. 2, S. 120–127

[15] KARAYIANNIS, N. B. ; VENETSANOPOULOS, A. N.: *Artificial neural networks: learning algorithms, performance evaluation, and applications*, *The Kluwer international series in engineering and computer science*. Boston : Kluwer Academic, 1993 — ISBN 978-0-7923-9297-2

[16] GOODFELLOW, IAN ; BENGIO, YOSHUA ; COURVILLE, AARON: *Deep Learning* : MIT Press, 2016

[17] ABADI, MARTÍN ; AGARWAL, ASHISH ; BARHAM, PAUL ; BREVDO, EUGENE ; CHEN, ZHIFENG ; CITRO, CRAIG ; CORRADO, GREG S. ; DAVIS, ANDY ; U. A.: *TensorFlow: Large-scale Machine Learning on Heterogeneous Systems*, 2015

[18] DÍAZ, A. ; SIGMUND, O.: Checkerboard patterns in layout optimization. In: *Structural optimization* Bd. 10 (1995), Nr. 1, S. 40–45

[19] SIGMUND, O. ; PETERSSON, J.: Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. In: *Structural optimization* Bd. 16 (1998), Nr. 1, S. 68–75

[20] TOPOPT GROUP: *Topology optimization codes written in Python - TopOpt*. URL http://www.topopt.mek.dtu.dk/Apps-and-software/Topology-optimization-codes-written-in-Python. - abgerufen am 2020-01-27. — http://www.topopt.mek.dtu.dk

[21] ANDREASSEN, ERIK ; CLAUSEN, ANDERS ; SCHEVENELS, MATTIAS ; LAZAROV, BOYAN S ; SIGMUND, OLE: Efficient topology optimization in MATLAB using 88 lines of code. In: *Structural and Multidisciplinary Optimization* Bd. 43 (2011), Nr. 1, S. 1–16