Masterarbeit

Parametervorhersage und automatisiertes Erzeugen von Verbindungstechnologien in CAX-Anwendungen von Automobil-Karosserien

parameter prediction and automated generation of Car-body-connector-technologies in CAX-applications

vorgelegt von
B.Sc. Maximilian Knorr

betreut durch
Prof. Dr.-Ing. Horst E. Friedrich
Prof. Dr.-Ing. Andreas Wagner
Dr. Ing. Christoph David

Institut für Verbrennungsmotoren und Kraftfahrwesen
UNIVERSITÄT STUTTGART

angefertigt am
Deutschen Zentrum für Luft- und Raumfahrt e.V.
Institut für Fahrzeugkonzepte

27. Oktober 2019

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe bzw. unerlaubte Hilfsmittel angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Stuttgart, den 27. Oktober 2019

Kurzfassung

In der vorliegenden Abschlussarbeit sind Grundlagen für ein Expertensystem dargelegt, das in der Lage ist, automatisiert Schweißpunkte auf Karosseriebauteilen im Preprozessing von CAX-Anwendungen zu erzeugen. Dabei lernt das System durch maschinelle Lernalgorithmen aus bereits existierenden Schweißpunktverteilungen.

Zunächst werden die Anzahl und Lage der Schweißpunkte sowie die Flanschbereiche der Karosserieblechen aus vorhandenen Datensätzen extrahiert. Die extrahierten Daten werden dann aufbereitet und abstrahiert, um verschiedene neuronale Netzearchitekturen mittels maschinellen Lernalgorithmen zu trainieren. Dabei sind mehrere Prozesse dargelegt, die Möglichkeiten beschreiben, die Schweißpunkte in CAX-Anwendungen zu erzeugen.

Mit den algorithmischen Ergebnissen dieser Arbeit können Flanschbereiche sowie Schweißpunktinformationen aus vorhandenen Datensätzen extrahiert werden, diese automatisiert abstrahiert und für das Training von neuronalen Netzen verwendet werden. Zudem sind Möglichkeiten dargelegt, die den Workflow von einem Expertensystem zur automatisierten Erzeugung von Schweißpunkten beschreiben.

Abstract

This thesis shows the basics of an expert-system that is able to automatically generate spotwelds on carbodyparts in the preprocessing of CAX-applications. The system is trained by machine learning algorithms using abstracted existing data sets.

First, the relevant spotweldparameters such as the number and position of the spot welds and the flange areas of the carbody panels are extracted from existing data records. With an abstraction of these extracted data neural network architectures are trained using machine learning algorithms. Several workflows are shown that describes ways to create the spotwelds in CAX-applications.

The results of this thesis are algorithms, that are able to extract flange areas as well as spotweld information from existing datasets, which are abstracted and used for the training of neural networks. In addition, possibilities are shown that describe the workflow of an expert system for the automated generation of spot welds.

Inhaltsverzeichnis

Abkürzungsverzeichnis

PEP Produktionsentwicklungsprozess

ID Identifikator

CAE Computer-Aided Engineering

FEM Finite-Elemente-Methode

CAD Computer-Aided Design

PEP Produktentwicklungsprozess

SOP Start of Production

PID Part Identificator

EID Element Identificator

NID Node Identificator

KI Künstliche Intelligenz

MLP Multilayer-Perzeptron

MSE mean square error

DOE Design of Experiment

PW Punktewolke

CNN Convolutional Neuronal Network

LHS Latin Hypercube Sampling

PLY Polygon File Format

CM Cuthill-McKee

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Stand der Technik

Der steigende Wettbewerbsdruck auf die Automobilentwickler fordert sinkende Entwicklungszeiten bei gleichzeitig steigenden Qualitäts- und Sicherheitsanforderungen. [1]
Um den Produktionsentwicklungsprozess (PEP) von Fahrzeugen zu verkürzen, ist aus der früheren seriellen Abarbeitung der einzelnen Entwicklungsstände eine überlappende Abarbeitung entstanden. Diese überlappende Abarbeitung wird auch als simultane Entwicklung bezeichnet. In Abb. 1 sind die Entwicklungskonzepte der seriellen und der simultane Entwicklung exemplarisch gegenübergestellt. Der Zeitgewinn Δt bis zum Start der Produktion (SOP) resultiert aus einem Parallelisieren und einem Ineinandergreifen der einzelnen Entwicklungsstände. Diese Arbeitsweise ist nur dann funktionsfähig, wenn der jeweils nächste Entwicklungsstand mit unreifen oder noch nicht detaillierten Daten und Informationen den Entwicklungsprozess beginnt und die Prozesse nach und nach mit Datenupdates aus den vorgeschalteten Entwicklungsprozessen aktualisiert. An den Übergangsbereichen, die in Abb. 1 rot markiert sind, erfolgt die Übergabe der Daten und Informationen an den nächsten Entwicklungsschritt. [2]

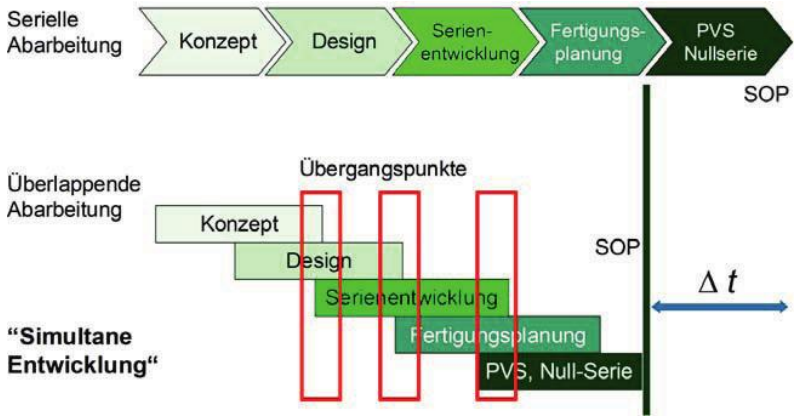


Abb. 1: Simultane Fahrzeugentwicklung [2]

Das Einsetzen der simultanen Entwicklungsstrategie in der Fahrzeugbranche fordert einen massiven Einsatz von nummerischen CAE-Anwendungen und dem damit zusammenhängenden synchronen Verwalten von Simulationsdaten. Durch nummerische CAE-Methoden können in der frühen Entwicklungsphase mit nicht aktuellen Datensätzen Prognosen über die Fahrzeugeigenschaften gestellt werden, um diese vor der Fertigung von ersten Prototypen zu beeinflussen. CAE steht für „Computer Aided Engineering“, also computergestütztes Entwickeln. Diese nummerischen Anwendungen werden in vielen Bereichen der Fahrzeugentwicklung eingesetzt, wie z.B. in der Karosserieentwicklung, der Auslegung von Aggregaten und Elektronikkomponenten, im Fahrwerksbereich sowie im Gesamtfahrzeugbereich. [3]

Bereits seit den 60er Jahren beschäftigen sich Ingenieure mit der Entwicklung nummerischer CAE-Anwendungen. Die Idee liegt darin, reale Bauteile in endliche Bereiche einzuteilen und diese durch Ansatzfunktionen mit freien Parametern zu beschreiben. Die entstehenden Gleichungen können dann von Computern berechnet werden. Heute werden CAE-Anwendungen in Bereichen wie Maschinenbau, Luft- und Raumfahrt und Automobilbau eingesetzt. Im Karosseriebereich werden CAE-Anwendungen zum Beispiel für Crash-Simulationen eingesetzt. Heute können aufgrund der Verlässlichkeit auf die Simulationsergebnisse ganze Prototypenbaureihen entfallen. Durch die weiteren Entwicklungen, wie der massiven Parallelisierung von Prozessoren und den immer günstiger werdenden Rechnern, steigt die Anzahl der Simulationen und die Kosten für die Arbeitsstunden des Anwenders für den Preprozess und den Postprozess sind vergleichsweise hoch. [1]

Ein manueller Teilprozess des Preprozessings ist das Definieren von Verbindungstechnologien für die einzelnen Bleche der Karosserie. Um einen Überblick über die Anzahlen solcher Verbindungen in Fahrzeugkarosserien zu bekommen ist in Abb. 2 eine Abbildung aller Punktschweißverbindungen des Toyota Yaris dargestellt. Insgesamt sind 3.847 Schweißpunktverbindungen im CAE-Modell des Toyota-Yaris definiert.

Um solche Prozesse zu automatisieren, ist der Trend zu Systemen der künstlichen Intelligenz erkennbar. Die Systeme der künstlichen Intelligenz können in zwei Bereiche aufgeteilt werden: Wissensbasierte Systeme und Expertensysteme.
Wissensbasierte Systeme sind ein Untergebiet der künstlichen Intelligenz und folgen zuvor definierten und in der Wissensbasis hinterlegten Regeln und Fakten, um angefragte Problemstellungen zu lösen. Durch wissensbasierte Systeme können so wiederkehrende Aufgaben für den Anwender übernommen werden und somit die Arbeitsstundenkosten gesenkt werden. Ein Beispiel für ein wissensbasiertes System ist die Erkennung von Konstruktionsradien durch ein CAM-System. Hierbei kann das CAM-System durch festgelegte Regeln und Fakten

Abb. 2: Schweißpunktverteilung-Toyota-Yaris

entscheiden, wie es mit den unterschiedlichen Radien von Konstruktionen beim Fertigen umgeht.[33]

Auf einen Anwendungsfall spezialisierte wissensbasierte Systeme sind Expertensysteme. In der Wissenschaft der Künstliche Intelligenz (KI) wird eine Software oder ein Programm als Expertensystem bezeichnet, sobald es in der Lage ist, Probleme aus einem eingegrenzten Fachgebiet in einer Qualität zu lösen, die mit der Lösung eines menschlichen Experten vergleichbar ist. Das übergeordnete Ziel ist die Generierung von Systemen, die so viele Prozesse wie möglich voll- oder teil-automatisiert durch Programme und Algorithmen übernehmen. Ein Beispiel für ein solches Expertensystem aus der Medizintechnik ist ein Programm, dass Röntgenaufnahmen diagnostiziert und auswertet.[34] Eine Herangehensweise, Expertensysteme mit Informationen und Daten zu versorgen, bieten maschinelle Lernalgorithmen.[4]
In CAE-Anwendungen können auf maschinellen Lernalgorithmen basierende Expertensysteme im Preprozessing sowie im Postporzessing eingesetzt werden und die Anwender bei immer wiederkehrenden Prozessen unterstützen. Aktuell gibt es Entwicklungen solcher Systeme hauptsächlich im Bereich des Postprozessings. Dabei werden diese Systeme beim Analysieren von Simulationsergebnissen eingesetzt. Das Expertensystem soll dann Vorschläge zur Aktualisierung der Karosserie- und Simulationsparameter geben, um definierte Simulationsziele zu erreichen. [36] Ein bisher gering erforschtes Gebiet ist der Einsatz eines Expertensystems im Preprozessing.
In dieser Arbeit sollen die Grundbausteine für ein Expertensystem gelegt werden, das in der Lage ist, aus bereits bestehenden Daten zu lernen und diese Informationen im Preprozessing

von CAE-Anwendungen auf neue Probleme anzuwenden. Das Augenmerk hierbei liegt auf dem Platzieren von Schweißpunkten auf Karosserieblechen für die Crash-Simulation.

1.1 Zielsetzung

Das Ziel dieser Masterarbeit ist es, eine Methode zu entwickeln, die Teile des Preprozessing in der Crashsimulation automatisiert. Genauer soll ein Basisalgorithmus erarbeitet werden, der in der Simulationsvorbereitung das Setzen von Punktschweißverbindungen automatisiert. Dabei soll der Algorithmus mittels maschinellem Lernen aus bereits vorhandenen Simulationsdatensätzen lernen und eine Vorhersage des Schweißpunktbildes für neue Bauteile in der Karosserieentwicklung treffen können.

1.2 Vorgehen

Um das zuvor beschriebene Problem anzugehen, ist der erste Schritt das Sichten der zur Verfügung stehenden Datensätze. Hierzu dient ein veröffentlichtes Crashmodell des Toyota Yaris, das die zivile US-Bundesbehörde für Straßen- und Fahrzeugsicherheit „National Highway Traffic Safety Administration“ bereitstellt. [https://www.nhtsa.gov/] Aus diesem aufbereiteten Crashmodel sollen relevante Daten für das Training des Algorithmus extrahiert und aufbereitet werden. Auf dieser Basis wird eine Methode zur automatischen Erzeugung von Schweißpunkten in CAD- und FEM-Anwendungen mit einer Vorhersage des Schweißpunktortes dargelegt.

2 Grundlagen Simulationsprozess

Im Folgenden sind die Grundlagen zu den Bereichen der nummerischen Anwendungen sowie die mathematischen Grundlagen zu künstlichen neuronalen Netzen erklärt und erläutert.

2.1 Simulationsprozess

Der allgemeine Simulationsprozess besteht aus vier Phasen der Konstruktion, dem Preprozessing, dem Solving und dem Postprozessing. In Abb. 3 sind diese vier Schritte mit den jeweiligen Unteraufgaben innerhalb der Einzelbereiche dargestellt. Im Preprozess werden die CAD-Daten aufbereitet, die Geometrien vernetzt, die Bauteileigenschaften definiert und die Lastfälle erstellt. Diese Informationen werden in Form einer maschinell lesbaren Eingangsdatei mit allen Informationen für den Solver lesbar abgespeichert. Der Solver durchsucht diese Datei nach den für ihn relevanten Informationen und wandelt diese in berechenbare Matrizen um und löst die aufgestellten Gleichungen. Im postprozessing findet die Auswertung, Visualisierung und die Analyse der berechneten Daten statt. Um ein definiertes Ergebnis zu generieren, kann dieser Prozess je nach Anwendung iteriert oder optimiert werden.

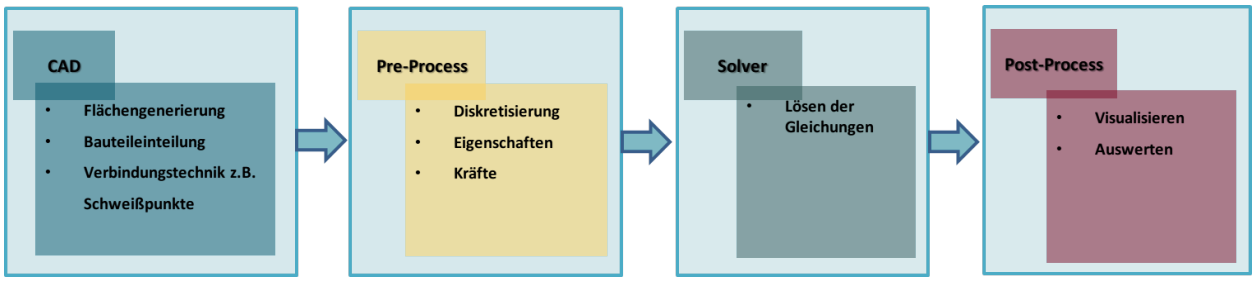


Abb. 3: Simulationsprozess

2.1.1 Preprozess

Beim preprozessing erfolgt die Bereitstellung und Vorbereitung der CAD-Datenumfänge für die Anforderungen der Netzerstellung. Beim Rechnen mit blechartigen Bauteilen, wie es im Karosseriebereich der Fall ist, sind die Blechbauteile aufgrund der geringeren Rechenzeit auf Mittelflächen abstrahiert. Nicht oder gering relevante Bauteile wie nicht crashrelevante Bauteile aus dem Motorraum sind als reduzierte Topologien dargestellt, um die Modelle und die Rechenzeit zu verkleinern. [2]

Durch automatische Mesh-Algorithmen, die in Form von Tools in modernen Preprozessoren

integriert sind, werden die einzelnen Flächen vernetzt. Für die zu vernetzende Geometrie wird ein Netz gesucht, das Randbedingungen wie zum Beispiel minimale und maximale Netzkanten einhält. Crash-Simulationen verwenden vorzugsweise regelmäßige und lineare Viereckснetze mit möglichst gleichen Kantenlängen.
Eine Beschreibung von Karosseriebauteilen ausschließlich mit Viereckselementen mit gleicher Kantenlänge ist bei gekrümmten Bauteilen oft nicht möglich. An solchen Stellen werden Dreieckselemente zugelassen, wenn eine Verzerrung der Viereckselemente dadurch vermeidbar ist. Ein solches typisches Netz für die Crash-Simulation aus gemischten Elementen ist am Beispiel eines Kotflügels des Toyota Yaris Modells in Abb. 4 zu sehen.

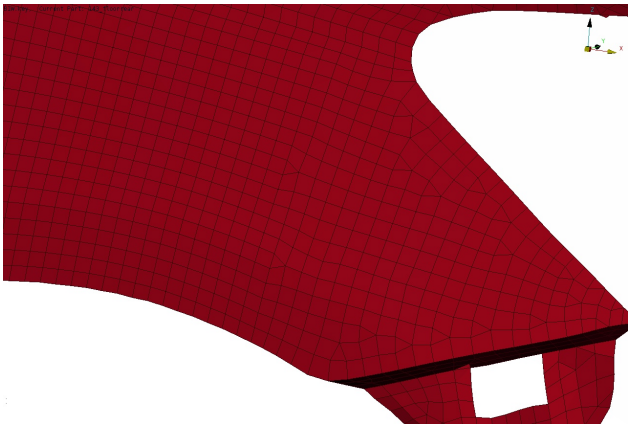


Abb. 4: FE-Netz des Toyota-Yaris-Kotflügels

Die grundlegende Zielsetzung des Preprozesses ist die Erstellung der maschinell lesbaren Eingangsdatei für den Solver. Die Eingangsdatei ist ein vom FEM-Solver lesbares Inputdeck, in dem die für das Modell relevanten Informationen beschrieben sind. Jeder Solver hat sein individuell standardisiertes Eingangsformat und eine standardisierte Syntax. Die Informationen, wie zum Beispiel die Part- und Elementdefinitionen, Lastfälle und die Randbedingungen sind innerhalb der Inputdecks von Keywords eingeleitet.

2.1.2 LS-Dyna spezifische Inputdecks

Im Folgenden sind Teile von LS-Dyna spezifischen Inputdecks dargelegt, um die Struktur dieser zu beschreiben. Die diskretisierten Einzelteile sind in Part ID's eingeteilt, welche durch das Keyword $\$PARTcards$ eingeleitet und somit für den Solver eindeutig identifizierbar sind. Jedes Einzelteil des Modells bekommt eine eigene PID in Form von durchnummerierten Zahlen zugewiesen. Anhand dieser PID ist das Bauteil zu jedem Zeitpunkt im Simulationsprozess eindeutig identifizierbar. Jeder PID sind unter dem Keyword $\$ELEMENTcards$ alle Elementdefinitionen zugeordnet. Die Definition der Elemente findet im Prozess der Diskretisierung statt. FE-Netze bestehen ausschließlich aus Vierecken und Dreiecken. Unter $*NODEcards$ ist eine Liste aller einzigartigen Knoten mit durchnummerierten Node-ID's hinterlegt. Das gesamte Modell ist dort also als komplette Punktewolke mit einzigartigen Punkten, die jeweils x-, y- und z-Koordinaten besitzen, hinterlegt. In Abb. 5 sind Auszüge aus dem Inputdeck des Toyota Yaris Crashmodells dargestellt, die das Prinzip der Bauteilbeschreibung in diesen beschreibt. [5]

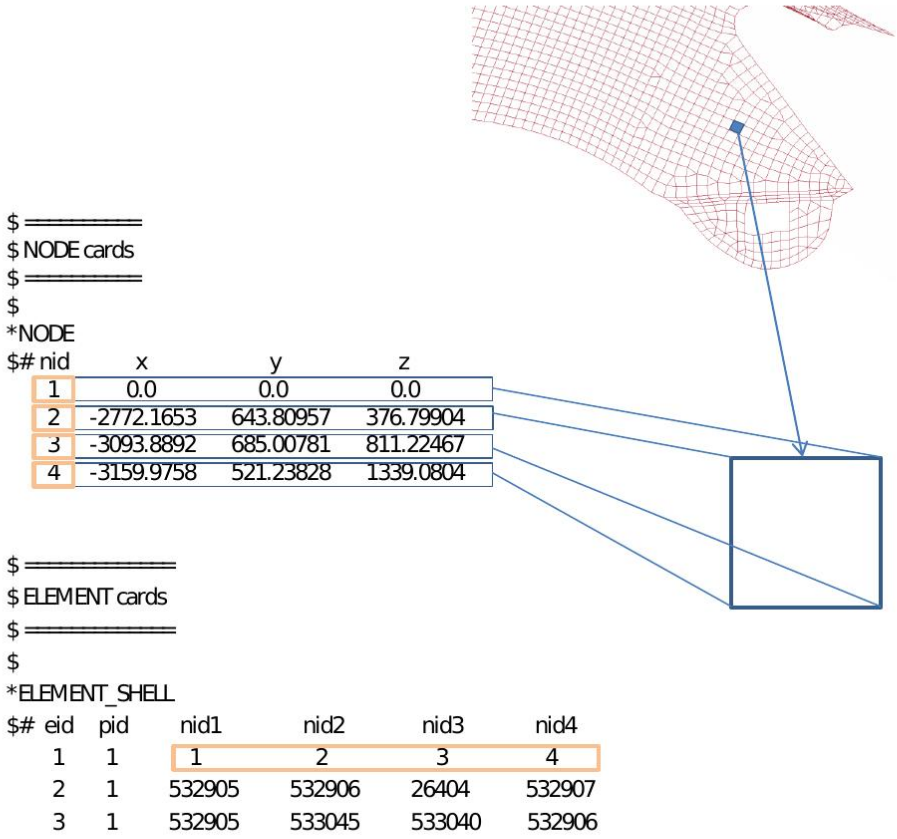


Abb. 5: Element- und Nodecards

Verbindungstechnologien wie zum Beispiel Schweißpunkte, die einzelne Bauteile verbinden, können in CAE-Anwendungen über vereinfachte Modelle realisiert werden. Einfache Schweißpunkte können über 1D-Stäbe mit den Eigenschaften der Schweißpunkte modelliert sein. In LS-Dyna-Inputdecks sind diese Verbindungsdaten mit dem Keyword $\$Connectioncards$ eingeleitet. Dieser gibt dem Solver die Information, dass im Folgenden die Verbindungen definiert sind. In Abb. 6 ist ein Ausschnitt der Verbindungskarte eines LS-Dyna-Inputdecks mit einer Punktschweißverbindung dargestellt. Im Fall eines Schweißpunktes befinden sich die relevanten Informationen über den einzelnen Schweißpunkt zwischen den Keywords $\times$ $*CONNECTION\_START\_SPOTWELD$ und $*CONNECTION\_END\_SPOTWELD$. Darunter ist die Koordinatenangabe des Schweißpunktmittelpunktes angegeben. Die Information unter den Keywords $*CONNECTION\_LAYER\_PART\_ID$ beinhaltet die Information über die mit diesem Schweißpunkt verbundenen Bauteile in Form der PID. [5]

```
$ ================
$ Connection cards
$ ================
$
*CONNECTION_START_SPOTWELD
      X              y              z
-2963.0132     606.89185      297.8277
*CONNECTION_LAYER_PART_ID
    148
*CONNECTION_LAYER_PART_ID
    153
*CONNECTION_ENTITIES
NODE                    531808   531809
BEAM                      1
*CONNECTION_END_SPOTWELD
```

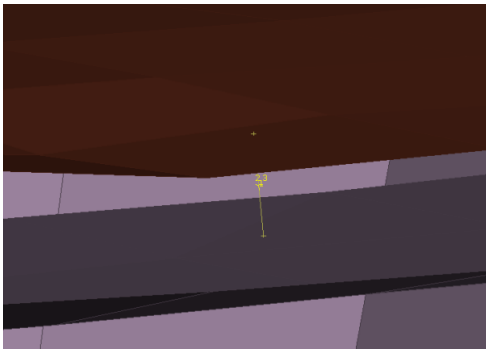


Abb. 6: Element- und Nodecards

2.1.3 Solving

Die Finite-Elemente-Methode basiert auf der Idee kontinuierliche Probleme in diskrete Probleme zu vereinfachen und dies durch aufgestellte Differentialgleichungen zu lösen.
Die Idee der Finiten-Elemente-Methode (FEM) in der Festigkeitslehre basiert auf der Überlegung, Berechnungen an einer diskreten Anzahl an Punkten durchzuführen, die über die gesamte Geometrie eines kontinuierlichen Körpers verteilt sind. Jedes kontinuierliche Objekt hat unendlich viele Freiheitsgrade. Es ist daher nur mit unendlicher Rechenleistung möglich, diese Probleme zu lösen. Die FEM reduziert die unendlichen Freiheitsgrade durch das Diskretisieren der Geometrie auf eine definierte und lösbare Anzahl an Knoten, Elementen und somit auch auf eine finite Anzahl an Differentialgleichungen. In Abb. 7 ist das Prinzip der FEM-Methode an einem einfachen Beispiel dargelegt. Auf der linken Seite ist ein 2D-Halbkreis dargestellt, der nicht diskretisiert ist und aus infiniten Elementen besteht. Die zu lösende Anzahl an Gleichungen ist deshalb ebenfalls unendlich. Auf der rechten Seite ist der 2D-Halbkreis über acht Knoten und 4 Elemente diskretisiert. Multipliziert mit der Anzahl an Freiheitsgraden ergeben sich 48 zu lösende Gleichungen. Diese können nummerisch von einem Rechner gelöst werden. Dasselbe Prinzip gilt auch für 1D und 3D Geometrien.[23]

Zunächst ist ein Beispiel in Form eines Stabes betrachtet. Es gilt das Hook'sche Gesetz: die Stabkraft F ist das Produkt aus der Verschiebung des Stabes und der Stabsteifigkeit $F = K \cdot U$. Für K gilt im linearen Bereich die Definition in Gleichung (9) wobei E das Elastizitätsmodul, A der Querschnitt des Stabes und l die Länge des Stabes ist.

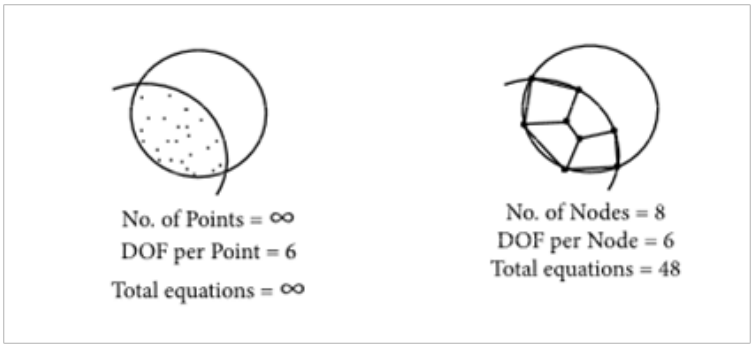


Abb. 7: Diskretisieren einer Geometrie

$$K = \frac{E \cdot A}{l} \tag{1}$$

Es gibt zwei Möglichkeiten den Stab zu belasten. Entweder durch das Aufbringen einer Kraft F oder durch eine aufgebrachte Verformung U. Dadurch entsteht eine Unbekannte, nach der das System aufzulösen ist. In Abb. 8 ist ein Stab dargestellt, an dessen Ende eine Kraft und eine Verschiebung jeweils in x-Richtung aufgebracht ist. Dieses System ist die allgemeine Definition eines Stabes mit Verschiebungsfreiheitsgraden. Die linearen Beziehungen zwischen den Verschiebungen und den Kräften des Systems sind in Gleichung (2) beschrieben. [24]

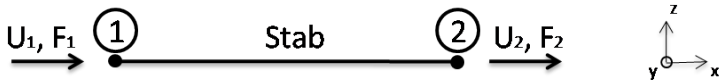


Abb. 8: Stabmodell

$$\begin{aligned} F_1 &= K \cdot U_1 - K \cdot U_2 \\ F_2 &= K \cdot U_2 - K \cdot U_1 \end{aligned} \tag{2}$$

Die Gleichung (2) in Matrixschreibweise ist in Gleichung (3) dargestellt:

$$\begin{bmatrix} K & -K \\ -K & K \end{bmatrix} \cdot \begin{bmatrix} U_{1x} \\ U_{2x} \end{bmatrix} = \begin{bmatrix} F_{1x} \\ F_{2x} \end{bmatrix} \tag{3}$$

Damit ist die Steifigkeitsmatrix für einen Stab durch $\begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$ in seinem lokalen Koordinatensystem beschrieben. Für ein System aus mehreren Elementen müssen die Steifigkeitsmatrizen jedes Elementes zu einer Gesamtsteifigkeitsmatrix zusammengefügt werden. In Gleichung (6) ist ein System aus zwei Stabelementen dargestellt, dessen Gesamtsteifigkeitsmatrix in Gleichung (5) kompiliert ist. [24]

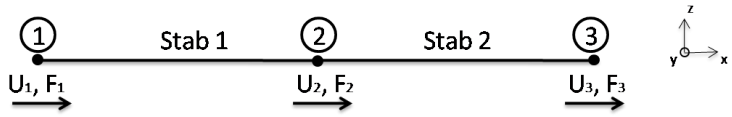


Abb. 9: Stabsystem aus zwei Elementen

$$K_1 = K_{Stab,1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad K_2 = K_{Stab,2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} K_1 & -K_1 & 0 \\ -K_1 & K_1 + K_2 & -K_2 \\ 0 & -K_2 & K_2 \end{bmatrix} \cdot \begin{bmatrix} U_{1x} \\ U_{2x} \\ U_{3x} \end{bmatrix} = \begin{bmatrix} F_{1x} \\ F_{2x} \\ F_{3x} \end{bmatrix} \tag{5}$$

Wie in Abb. 7 dargestellt, gibt es bei realen Systemen mehr Freiheitsgrade als in ihren diskretisierten Modellen. Um dennoch die Genauigkeit der Modelle im Vergleich zur Realität zu verbessern gibt es die Möglichkeit, die Freiheitsgrade des Systems zu erhöhen. Im Folgenden sind zu den Verschiebungsfreiheitsgraden der beiden Stäbe Rotationsfreiheitsgrade hinzugefügt. Diese sind in Abb. 10 dargestellt. Wobei U_{1x}, U_{2x}, U_{3x} Verschiebungen in x-Richtung und U_{4x}, U_{5x}, U_{6x} Rotationen um die x-Achse sind.

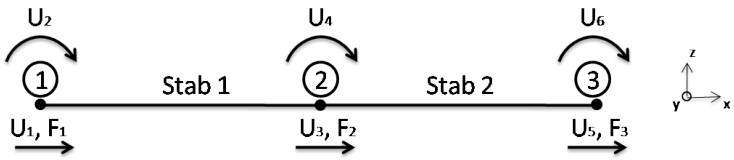


Abb. 10: Stabsystem mit mehreren Freiheitsgraden

Die Elementsteifigkeitsmatrizen ändern sich von einer (2×2)-Matrix zu einer (4×4)-Matrix, da jeder Knoten insgesamt 2 Freiheitsgrade besitzt und somit 2 Zeilen und 2 Spalten belegt. Das System ist im 2D-Raum dargestellt. In Gleichung (6) sind die (4×4)-Elementsteifigkeitsmatrizen zu sehen wobei die Indizes i die Knoten und j die Freiheitsgrade von $K_{i,j}$ darstellen.

$$K_1 = \begin{bmatrix} K_{11} & K_{12} & -K_{23} & -K_{24} \\ K_{11} & K_{12} & -K_{23} & -K_{24} \\ -K_{23} & -K_{24} & K_{11} & K_{12} \\ -K_{23} & -K_{24} & K_{11} & K_{12} \end{bmatrix} \quad K_2 = \begin{bmatrix} K_{23} & K_{24} & -K_{35} & -K_{36} \\ K_{23} & K_{24} & -K_{35} & -K_{36} \\ -K_{35} & -K_{36} & K_{23} & K_{24} \\ -K_{35} & -K_{36} & K_{23} & K_{24} \end{bmatrix} \tag{6}$$

Das Gleichungssystem des Gesamtsystems ist in Gleichung (7) dargestellt.

$$\begin{bmatrix} K_{11} & K_{12} & -K_{23} & -K_{24} & 0 & 0 \\ K_{11} & K_{12} & -K_{23} & -K_{24} & 0 & 0 \\ -K_{23} & -K_{24} & K_{11}+K_{23} & K_{12}+K_{24} & -K_{35} & -K_{36} \\ -K_{23} & -K_{24} & K_{11}+K_{23} & K_{12}+K_{24} & -K_{35} & -K_{36} \\ 0 & 0 & -K_{35} & -K_{36} & K_{23} & K_{24} \\ 0 & 0 & -K_{35} & -K_{36} & K_{23} & K_{24} \end{bmatrix} \cdot \begin{bmatrix} U_{1x} \\ U_{2x} \\ U_{3x} \\ U_{4x} \\ U_{5x} \\ U_{6x} \end{bmatrix} = \begin{bmatrix} F_{1x} \\ F_{2x} \\ F_{3x} \\ F_{4x} \\ F_{5x} \\ F_{6x} \end{bmatrix} \tag{7}$$

Für 3D-Systeme gibt es 6 Freiheitsgrade pro Knoten, welche sich aus 3 Verschiebungen in x-, y- und z-Richtung und 3 Rotationen um die x-, y-, und z-Achse zusammensetzen. Die Elementsteifigkeitsmatrizen der Stäbe haben dann 12 Zeilen und 12 Spalten. Die Matrixgröße ist ist immer durch $(P \cdot n) \times (P \cdot n)$ definiert, wobei P die Anzahl der Knoten und n die Anzahl der Freiheitsgrade ist.
Um die Steifigkeitsmatrizen von Fachwerken mit windschiefen Stäben in 2D- und 3D-System darzustellen, sind alle Elementsteifigkeitsmatrizen der einzelnen Stäbe in das globale Koordinatensystem mit den jeweiligen 2D oder 3D Transformationsmatrizen übersetzt. In Gleichung (8) ist diese Transformation dargestellt. Ein Fachwerk besteht aus einzelnen Stäben, die in den Knoten gelenkig miteinander verbunden sind.

$$K_{glob} = T^T \cdot K_{local} \cdot T \tag{8}$$

Nach der Transformation können die Einzelsteifigkeitsmatrizen K_{glob} wie zuvor beschrieben in eine Gesamtsteifigkeitsmatriz überführt werden. In der FEM-Berechnung mit Scheibenelementen ist die Berechnung der Steifigkeitsmatrizen komplizierter, da hierbei die aufnehmbare Energie durch die Trägheit des Elementes mit berücksichtigt werden muss. Eine detaillierte Beschreibung zur Erstellung von Scheibenelement - Steifigkeitsmatrizen ist in [25] beschrieben, für diese Arbeit aber nicht relevant, da die geometrischen Informationen auch aus den 3D-Fachwerken der Flanschnetze extrahiert werden können.

2.2 Mathematische Grundlagen

Im Folgenden sind mathematische Grundlagen zu baryzentrischen Koordinaten und der Laten-Hypercube-Sampling-Methode beschrieben, die zum Verständnis der im Verlauf der Arbeit beschriebenen Algorithmen notwendig sind.

2.2.1 Baryzentrische Koordinaten

In der linearen Algebra und in der Geometrie dienen Baryzentrische Koordinaten dazu, die Lage von Punkten in Bezug auf ein gegebenes Simplex (Dreieck, Tetraeder, etc.) zu beschreiben. Die Darstellung erfolgt durch eine Koeffizientendefinition einer Affinkombination. Gegeben sei ein Simplex mit einer beliebigen Anzahl an Punkten $P_B = P_0, P_1, P_2, ..., P_n$ $\{P_n \in \mathbb{N}^{\not<}\}$. Durch die Multiplikation mit den Skalaren $\alpha = \alpha_0, \alpha_1, \alpha_2, ..., \alpha_n$ $\{\alpha_n \in \mathbb{N}^{\not<}\}$ lassen sich affine Abbildungen dieser Geometrien in derselben Ebene darstellen. In Gleichung (9) sind die baryzentrischen Koordinaten eines Dreiecks mit P_n Punkten dargestellt.

$$P_B = \alpha_0 P_0 + \alpha_1 P_1 + \alpha_2 P_2 \tag{9}$$

Für die Skalare von α_n von Punkten innerhalb des Dreiecks gilt Gleichung (10):

$$0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1 \tag{10}$$

Ist ein Skalar $\alpha_n < 0$ oder $\alpha_n > 1$ befindet sich der Punkt außerhalb des Simplex. Ist ein Skalar $\alpha_n = 0$ befindet sich der Punkt auf einer der Linien, die das Simplex beschreiben [40]. In Abb. 11 ist dies beispielhaft anhand eines Dreiecks und drei Punkten P, Q und R, die mittels der beschriebenen Skalare ermittelt wurden, dargestellt.

- $P : \alpha_1 = \alpha_2 = \frac{1}{4}, \alpha_3 = \frac{1}{2}$
- $Q : \alpha_1 = \frac{1}{2}, \alpha_2 = \frac{3}{4}, \alpha_3 = -\frac{1}{4}$
- $R : \alpha_1 = 0, \alpha_2 = \frac{3}{4}, \alpha_3 = \frac{1}{4}$

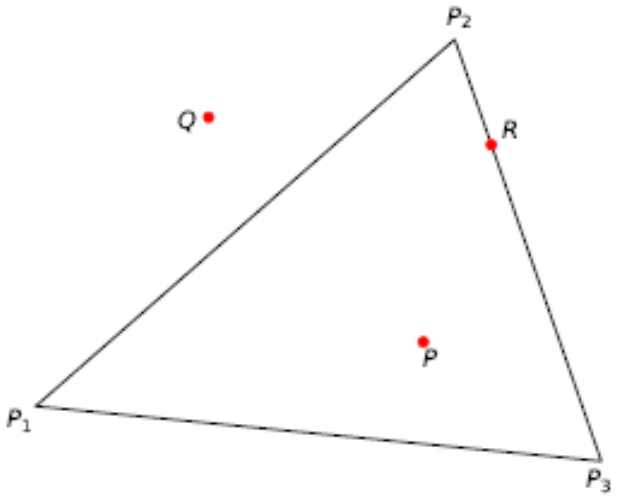


Abb. 11: Punkte durch Baryzentrische Koordinaten generieren

2.2.2 Latin-Hypercube-Sampling

Das Latin Hypercube Sampling (LHS) ist eine Methode zum Abtasten eines Modelleingaberaums, um Daten für das Training von Metamodellen zu generieren. Dabei ist das Ziel eine repräsentative Information über die Zielfunktion zu erhalten, bei so wenigen Stichproben wie möglich. Die Eingangsdaten spannen einen durch ein Raster aufgeteilten Raum auf. Durch statistische Stichproben werden Punkte auf diesem Raster so verteilt, dass in jeder Dimension mindestens ein Punkt auf jeder Zeile und Spalte liegt. Diese Punkte stellen die Eingangsparameter für die durchzuführenden Experimente dar.
Wenn eine Funktion von N Variablen abgetastet wird, wird der Bereich jeder Variable in M gleichwahrscheinliche Intervalle unterteilt. In Abb. 12 sind die variablen N = X und Y und die Intervalle M die Gitterstruktur, die in Intervalle von 0,2 eingeteilt sind.
Um die Latin-Hypercube-Anforderungen zu erfüllen, wird ein Abtastpunkt pro Intervall M platziert. Ein Abtastpunkt pro Intervall. Jede Variable M wird durch eine kumulative

Verteilungsfunktion dargestellt und diese in M Regionen unterteilt. Aus jeder dieser Regionen wird eine Stichprobe bestimmt. Dadurch wird die Wahrscheinlichkeit erhöht, dass der gesamte Bereich abgedeckt wird. In Abb. 12 ist die LHS-Methode anhand eines 2D-Beispiels dargestellt.

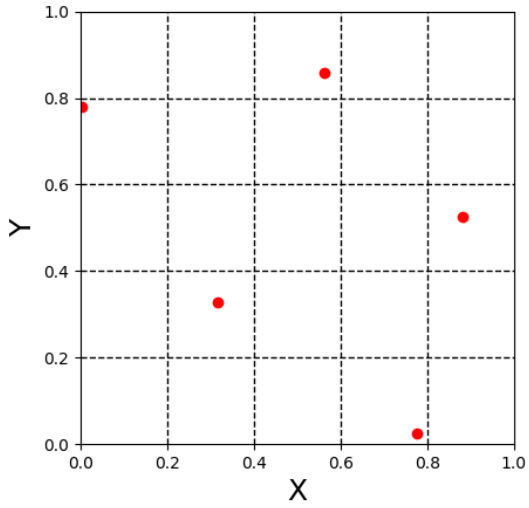


Abb. 12: Latin-Hypercube-Sampling 2D-Beispiel

Wie in Abb. 12 zu sehen ist, befindet sich in jeder Zeile und Spalte im (x, y) Raum nur eine Stichprobe.

2.3 Künstliche neuronale Netze

Einzelfunktionen des menschlichen Gehirns dienen als Inspiration für die in Computern simulierten neuronalen Netze, den künstlichen neuronalen Netzen. Diese stellen eine Teilarchitektur des menschlichen Gehirns in Form von Neuronen und deren Verbindungen dar. In den letzten Jahren hat die Modellierung von neuronalen Netzen einen starken Zuwachs durch die Veröffentlichung von Frameworks bekommen, um Anwendungsprobleme in Bereichen wie z.B. Statistik, Wirtschaft und Technik zu lösen. Im Zusammenhang mit neuronalen Netzen wird heute auch häufig über KI gesprochen. Dabei sind neuronale Netze mathematische Werkzeuge, derer sich KI-Algorithmen bedienen.
[8]

Um die Vorgänge innerhalb neuronaler Netze zu verstehen, wird zunächst das allgemeine Prinzip von künstlichen Neuronen dargelegt. Ein künstliches Neuron, das in Abb. 53 durch den Zellenkörper dargestelt wird, kann vereinfacht als Transistor mit einem Eingangs-, Ausgangs- und Schwellenwert für Informationen in Form von Signalen verstanden werden. Im Gehirn gibt ein Neuron ein Signal weiter, wenn ein Schwellenwert an elektrischen Impulsen, die von anderen Neuronen kumuliert am Eingang des Neurons anliegen, überschritten wird. Der Neuronenausgang gibt dann einen Impuls an weitere Neuronen ab. Biologische neuronale Netze sind aus Verkettungen von solchen Neuronen, die miteinander verbunden sind, aufgebaut. Die Signalübermittlungen zwischen den Neuronen übernehmen im menschlichen Gehirn die Synapsen, auch Axon genannt. Das Signal eines Neurons überwindet zunächst den sogenannten synaptischen Spalt, in dem variierbare chemische Vorgänge das Signal verändern. Überschreitet das kumulierte Eingangssignal einen Schwellenwert, gibt das Neuron selbst einen Impuls über die Dentriten an weitere Neuronen ab. Bei einer Nichtüberschreitung des Schwellenwertes endet der Informationsfluss in diesem Neuron. Der abgegebene Impuls ist also nichtlinear zum Eingangssignal. [6]

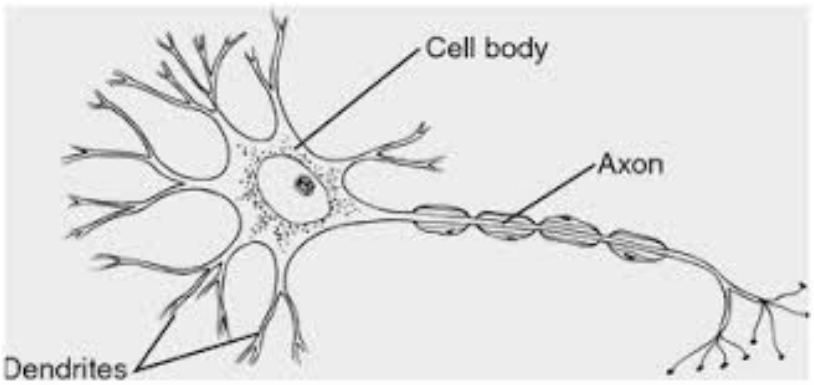


Abb. 13: Biologisches Neuron

2.3.1 Beschreibung von Einzelkomponenten neuronaler Netze

Ein künstliches neuronales Netz besteht aus einer Verkettung von Neuronen und Verbindungen. Das Multilayer Perzeptron (MLP) ist eine Feedforward Architektur von neuronalen Netzen. Es besitzt mindestens drei Neuronenebenen. Eine Eingangsneuronenebene, eine verdeckte Neuronenebene und eine Ausgangsneuronenebene. Diese Konstellation fällt unter den Begriff des maschinellem Lernens. Ein MLP mit einer verdeckten Neuronenebene ist in Abb. 14 dargestellt. Besitzt ein MLP mehrere verdeckte Neuronenebenen, wird das Netzwerk als ein sogenanntes tiefes Netzwerk bezeichnet. Ein häufig verwendeter Begriff für das Trainieren eines solchen Netzes ist Deep Learning.

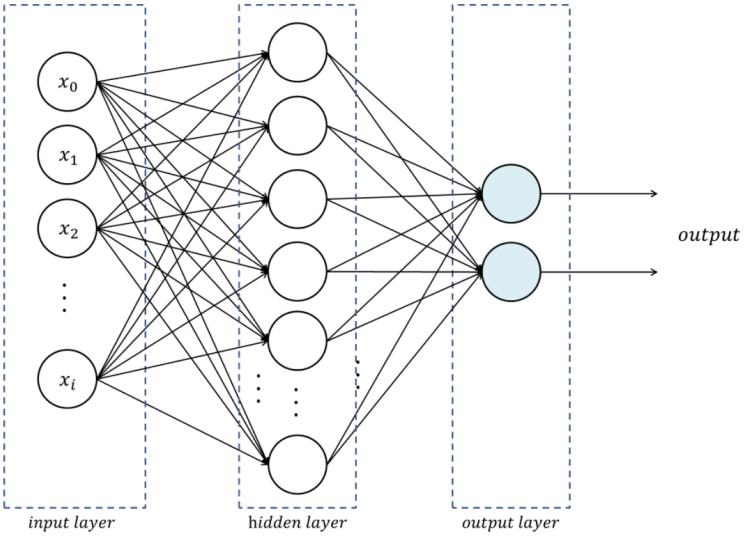


Abb. 14: Multilayer Perzeptron [10]

Im Folgenden soll auf die Einzelfunktionen der Komponenten innerhalb des MLP's eingegangen werden. In Abb. 15 ist das Prinzip eines Neurons und den Verbindungen schematisch dargestellt.
Jedes künstliche Neuron besitzt $x_n = x_0, x_1, x_2, ... \{n \in \mathbb{N}_0\}$ Eingangsvariablen. Vor dem Eingang in das Neuron werden diese Eingangsvariablen mit den Gewichten $w_n = w_0, w_1, w_2, ...\{w_n \in \mathbb{R}\}$ multipliziert. Das Aufsummieren dieser Produkte über die Gleichung (11) ergibt den Eingangswert z für ein Neuron. [7]

$$z = \sum_{i=0}^{n} w_n * x_n \tag{11}$$

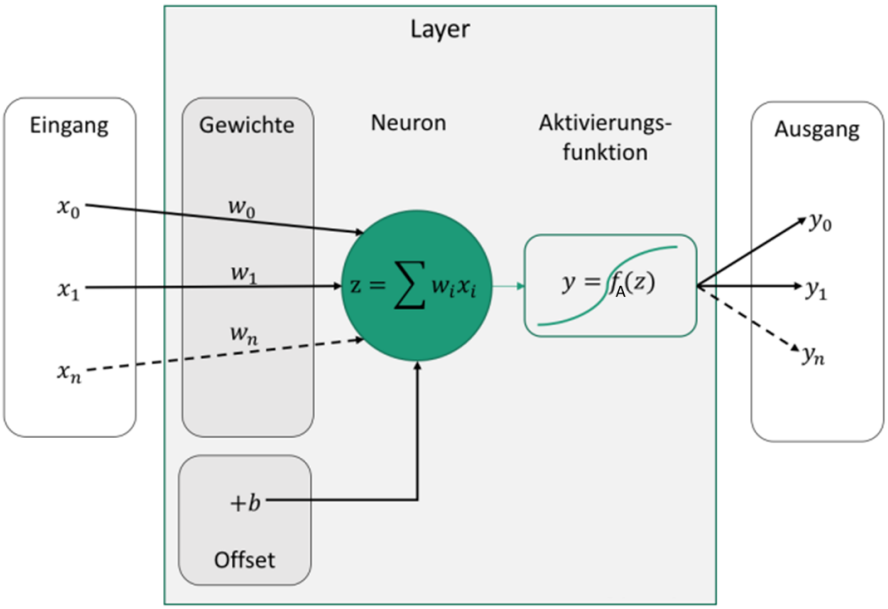


Abb. 15: Künstliches Neuron [6]

Durch das Definieren eines Offsetwertes $b \in \mathbb{R}$ kann der Eingangswert z beeinflusst werden. Der Wert y der Aktivierungsfunktion $f_A(z)$ ist der Ausgangswert des Neurons und somit der Eingangswert $y_n = y_0, y_1, y_2, ...\{n \in \mathbb{N}_0\}$ in die nachfolgenden Neuronen. Durch einen positiven Offsetwert steigt und durch einen negativen Offsetwert sinkt somit das Ansprechverhalten des Neurons. Der Einfluss ist in Gleichung (13) dargestellt.

$$y = f_A(z) \tag{12}$$

$$y = f_A(z + b) \tag{13}$$

Die Aktivierungsfunktion steuert das Ausgabeverhalten jedes Neurons innerhalb eines künstlichen neuronalen Netzes. Das heißt, die Aktivierungsfunktion entscheidet über den Informationsfluss, der von dem jeweiligen Neuron ausgeht. Diese Funktion ermöglicht dem neuronalen Netz, komplexe nichtlineare Beziehungen zwischen Ein- und Ausgangsvariablen zu definieren. Je nach Aktivierungsfunktion, verhalten sich die Ausgaben der Neuronen linear oder nichtlinear. Im Folgenden sind verschiedene Aktivierungsfunktionen dargestellt. [8]

Binärschritt-Funktion:

Diese Funktion stellt die einfachste der Funktionen dar. Unter anderem wird diese Funktion von binären Konvertern verwendet und gibt entweder eine Eins oder eine Null aus. Diese Tatsache spiegelt eine Eignung für Klassifizierungsprobleme, bei denen es um Ja oder Nein geht, wieder und wird daher hauptsächlich für die Klassifizierung verwendet. Diese Funktion ist in Abb. 16 dargestellt. [8]

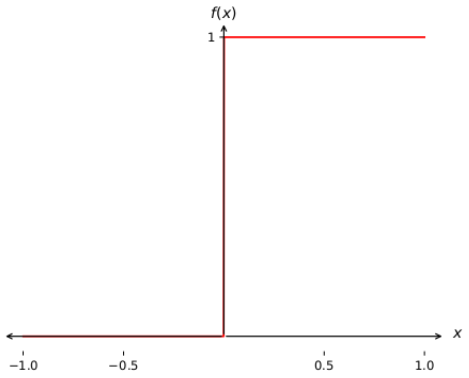


Abb. 16: Binaerschritt-Funktion

Der Gradient der Binärschrittfunktion ist $f(x) = 0$. Dadurch ist die Binärschritt-Funktion nicht für die Rückübertragung geeignet, die den Gradienten der Aktivierungsfunktion für die Fehlerberechnung verwendet, um die Gewichtungen zu verbessern und zu optimieren.
Sigmoid-Funktion:

Die Sigmoid-Funktion ist eine stetig differenzierbare Funktion. Ihr Vorteil im Gegensatz zur Binärschrittfunktion liegt in ihrer Nichtlinearität. Aus diesem Grund und ihrer nichtlinearen Ausgabe eignet sie sich auch für komplexe neuronale Netze. Ihr Rückgabewert steigt monoton von 0 bis 1 innerhalb eines definierten Intervalls an. In diesem Bereich bewirken kleine Änderungen des Eingangswertes große Änderungen des Ausgangswertes, was für eine Trainierbarkeit von komplexen nichtlinearen Problemen verwendet werden kann. Diese Funktion ist in Abb. 17 dargestellt.

Der Gradient der Funktion ist stetig differenzierbar, wodurch der Fehler bei der Rückübertragung ermittelt werden kann und die Gewichte dementsprechend aktualisiert werden.
Softmax:

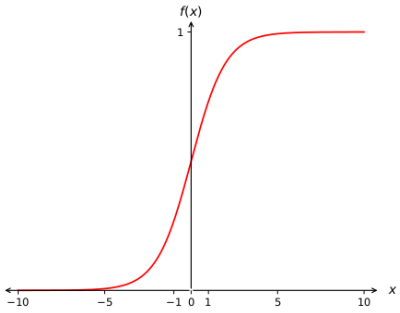


Abb. 17: Sigmoid-Funktion

Diese Funktion transformiert die Komponenten eines K-dimensionalen Vektors V_1 in einen K-dimensionalen Vektor V_2 dessen Komponenten im Interval [0, 1] liegen können. Alle Werte des Vektors V_2 besitzen aufsummiert den Wert 1. Die Softmax-Funktion wird für die Ausgangsebene von klassifizierenden neuronalen Netzen verwendet. Die Ausgangsebene gibt dann die Wahrscheinlichkeit an, mit der die Eingangsdaten einer Klasse zugewiesen werden. In Gleichung (14) ist die Softmax-Funktion und in Gleichung (15) ein Beispiel für einen Ausgangsvektor dargesetllt.

$$V_{2,i} = \frac{e^{V_{1,i}}}{\sum_{j=0}^{n} e^{V_{1,j}}} \tag{14}$$

$$V_1 = \begin{bmatrix} 2 \\ 1 \\ 0,1 \end{bmatrix} \xrightarrow{Softmax} V_2 = \begin{bmatrix} 0,7 \\ 0,2 \\ 0,1 \end{bmatrix} \tag{15}$$

ReLU-Funktion:

Die ReLu-Funktion ist eine im positiven gleichgerichtete Linearfunktion. Der Vorteil dieser Funktion gegenüber den anderen Aktivierungsfunktionen ist die Tatsache, dass nicht alle Neuronen gleichzeitig aktiviert werden. Dies geschieht dadurch, dass alle negativen Eingangswerte einen Ausgangswert von Null erzeugen und das Neuron sozusagen für diesen Fall deaktivieren. Der Gradient der ReLu Funktion ergibt wieder eine Binärschrittfunktion, die eine Eins oder eine Null ausgibt. Das Resultat daraus kann dazu führen, dass die Gewichte dieser Neuronen im Trainingsprozess nicht berücksichtigt werden und eine Gewichtung von Null entsteht. Dadurch können sogenannte tote Neuronen entstehen, die eine Gewichtung von Null besitzen und somit jeden Informationsfluss verhindern. Die ReLU-Funktion ist in Abb. 18 dargestellt.

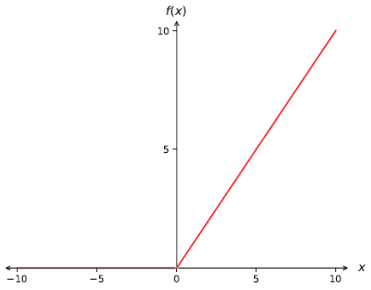


Abb. 18: ReLU-Funktion

Lernprozess:

Der Lernprozess von neuronalen Netzen beinhaltet typischerweise zwei Phasen: die Trainingsphase und die Testphase. In der Trainingsphase lernt das neuronale Netz anhand der vorgegebenen Lerndatensätze. Es ist wiederum zwischen zwei Lernarten zu unterschieden: dem überwachten Lernen, bei dem den 1D-Eingangsvektoren eindeutige Ausgangsvektoren zugeordnet sind und das Netz die Gewichtungen so optimiert, dass die Ausgangsdaten im Vergleich zu den Testdaten optimal sind und dem nicht überwachten Lernen, bei dem keine Ausgangsvektoren definiert sind. Bei diesem Trainingsverfahren ist die Änderung der Gewichte über die Ähnlichkeit und Abhängigkeit der Eingangsdaten gesteuert.

Ein Trainingseingangsvektor besteht aus einem beliebig langen Vektor $x = (x_1, x_2, x_3, ..., x_n)^T$ mit $\{n \in \mathbb{N}^+\}$ und einem gleichlangen Trainingsausgangsvektor $y = (y_1, y_2, y_3, ..., y_n)^T$ mit $\{n \in \mathbb{N}_0\}$. Das Trainingsset besteht aus k mit $\{k \in \mathbb{N}_0\}$ Eingangsvektoren $x_k = (x_1(k), x_2(k), x_3(k), ..., x_n(k))^T$ mit $\{n \in \mathbb{N}_0\}$ und k Ausgangsvektoren $y_k = (y_1(k), y_2(k), y_3(k), ..., y_n(k))^T$ mit $\{n \in \mathbb{N}_0\}$. Für jeden Eingangsvektor x_k des Trainingsdatensets berechnet der Algorithmus einen Ausgangsvektor über die Gewichtungen. Der Ausgangsvektor sei $o_k = (o_1(k), o_2(k), o_3(k), ..., o_n(k))^T$ mit $\{n \in \mathbb{N}_0\}$. Es ergibt sich ein Fehler E_k, der sich aus dem Vergleich des Sollausgangsvektors y_k und dem Istausgangsvektor o_k ergibt. Der Ausgabefehler E_k für jedes Neuron j ist in Gleichung (16) dargestellt. Aus den Residuen E_k des Ausgabefehlers kann eine Fehlerfunktion $E(W)$ definiert werden, wobei W ein Vektor ist, der die Gewichte einer Ebene des Netzes beinhaltet. [15]

$$E_{kj} = f_E(y_{kj}, o_{kj}) = \frac{1}{2}\sum_{j=1}^{n}(y_{kj} - o_{kj})^2 \quad \{n \in \mathbb{N}_0\} \tag{16}$$

Der Ausgangsfehler des gesamten Netzes für das Trainingsdatenset ist definiert als der „mean square error“ (MSE) und ist in der Gleichung (17) dargestellt, wobei m die Gesamtanzahl der Abweichungen $f_E(y_{kj}, o_{kj})$ ist.

$$MSE = \frac{1}{m}\sum_{k=1}^{m} f_E(y_k, o_k) \quad \{m \in \mathbb{N}_0\} \tag{17}$$

Der Backpropagation-Algorithmus ist ein Algorithmus, dem ein Gradientenabstiegsverfahren implementiert ist, mit dem Ziel, die Gewichte durch einen Optimierungsprozess so zu bestimmen, dass der MSE des gesamten Netzes minimal ist. Dafür stehen in den meisten Frameworks für neuronale Netze verschiedene Optimierungsalgorithmen zur Verfügung. Dieser Prozess der Gewichtsoptimierungen ist der Trainings- oder Lernprozess des neuronalen Netzes. Als Startwerte für die Optimierung dienen zunächst zufällige Werte im Intervall [-0.01, 0.01]. Der Lernprozess startet mit dem zuführen des Eingangsvektors x_k und dem Berechnen des neuen Ausgangsvektors o_k durch die zufälligen Gewichtungen. Dieser Prozess heißt Feedforward. Der Ausgangsfehler E_k wird nach Gleichung (16) errechnet und von ihm aus wieder Rückwärts in das Netz weitergeleitet, mit dem Ziel der Änderung der Einzelgewichte. Die Gewichtsänderung Δw ist durch $\Delta w = -\eta \nabla E(w_k)$ definiert, wobei η die Schrittgeschwindigkeit ist, die für die Lerngeschwindigkeit verantwortlich ist. Das Ziel ist es, das Minimum von $E(w_k)$ zu erreichen. Durch die Anwendung der Kettenregel und Modifikationen der Gleichung [15]

$$\Delta w_k = -\frac{\partial E_k}{\partial w_k} f\prime_A = \sum (E_k * w_k) * f\prime_A \tag{18}$$

Ist die Aktivierungsfunktion f_A eine stetig differenzierbare Funktion, ist automatisch auch die Fehlerfunktion $E(w_k)$, die durch alle Residuen E_k definiert ist, stetig differenzierbar. Durch ein Wiederholen dieses Vorganges und einer Konvergenz wird der Ausgangsfehler E_k immer kleiner, bis ein Abbruchkriterium erreicht ist. [9]

Die Testphase ist ein Prüfen des Trainingsprozesses, ob die im Trainingsprozess modifizierten Gewichtungen richtig eingestellt sind. Dazu ist wieder ein Datensatz notwendig, der sowohl 1D-Eingangsvektoren als auch die richtig bezeichneten 1D-Ausgangsvektoren besitzt. Mit dem Testdatensatz kann durch vergleichsweise ähnliche Datensätze wie die des Trainingsprozesses die Genauigkeit des trainierten Netzes getestet werden. Durch die Verwendung von Testdatensätzen, deren Eingangs- und Ausgangsvektoren sich stärker von denen des Trainingsprozesses unterscheiden, ist ein Testen auf die Übertragbarkeit der gelernten Daten auf andere Probleme möglich.

2.3.2 Convolutional neuronal networks

Die sogenannten Convolutional neuronal networks (CNN's) sind eine spezielle Art von künstlichen neuronalen Netzen. Ins Deutsche übersetzt ist der Wortlaut eines CNN's ein faltendes neuronales Netzwerk. Diese Netzwerke besitzen mindestens einen Convolutional-Layer, der sich vor dem MLP befindet. Der Convolutional-Layer macht es dem neuronalen Netz durch Faltmatrizen möglich, 2D-Eingangsmatrizen zu verarbeiten. Den Übergang von den 2D Matrizen zu 1D Eingangsvektoren in das MLP übernimmt ein Flatten-Layer, der die Matrizen in Vektoren umwandelt. [16]
Ein typisches Anwendungsgebiet für CNN's sind Bilderkennungsalgorithmen, die heute bereits in verschiedener Form im Einsatz. Ein Beispiel ist die Schilderkennung in modernen Fahrzeugen, bei der das Fahrzeug permanent Aufnahmen seiner Umgebung macht. Ein trainierter Algorithmus, der Muster in diesen Aufnahmen sucht, erkennt ihm bekannte Muster wie zum Beispiel Straßenschilder und gibt diese aus.

In Abb. 19 ist das Ablaufprinzip eines CNN's dargestellt, das eine Klassifizierung von handgeschriebenen Zahlen beschriebt. Im Wesentlichen besteht ein Convolutional-Layer aus faltenden Matrizenoperationen und den sogenannten Pooling-Layern, auf deutsch Vereinigungsebenen.[16]

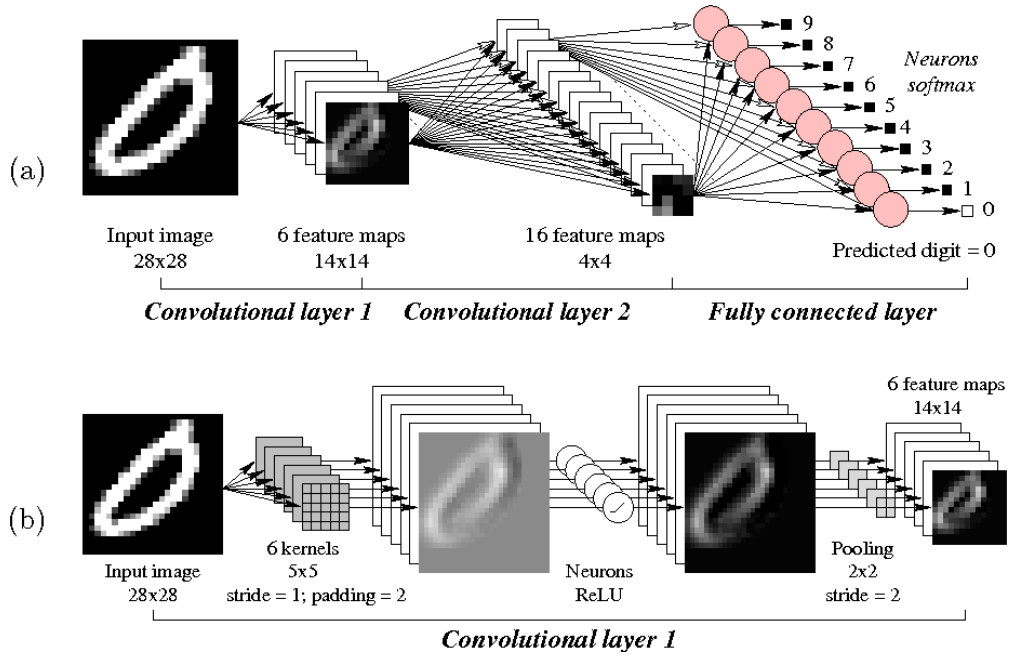


Abb. 19: Convolutional Neuronal Network [11]

In Abb. 19 (a) ist die Eingangsmatrix eine handgeschriebene Null, die aus (28×28) Graustufenpixeln besteht. Graustufenpixel besitzen einen Wert im Intervall $[0, 1]$. Die Eingangsmatrix wird zunächst von *kernels*, sogenannten Filtern analysiert, die eine definierte Pixelgröße besitzen. In dem Beispiel in Abb. 19 (b) besitzt jeder dieser Filter eine Größe von (5×5) Pixeln. Jedem dieser Pixel ist ein Wert zugeordnet. Der Filter scannt den kompletten Bereich der Matrix mit einer definierten Schrittweite *strides* von links nach rechts ab, indem die Filtermatrix mit den jeweiligen Bereichen der Eingangsmatrix multipliziert wird und die Ergebnisse addiert werden. Daraus ergibt sich ein neuer Wert, der an der jeweiligen Stelle in eine neue Matrix eingetragen wird. Durch das Definieren des Paddingwertes kann die Laufweite eingestellt werden, die beschreibt, wie sich das Scannen beim Anstoßen an den Randbereich verhält. Bei einem Padding von 0 springt der Filter beim Erreichen des Randes in die nächste Zeile, wodurch die Ergebnismatrix nach dem Scannen kleiner ist. Bei einem Padding von 2, überragt die Filtermatrix die zu scannende Matrix im Randbereich, wodurch die Ausgangsmatrix dieselbe Größe wie die Eingangsmatrix besitzt. Ein Beispiel für diesen Vorgang ist in Abb. 20 dargestellt. Der mathematische Hintergrund dazu ist in Gleichung (19) beschrieben. [16]

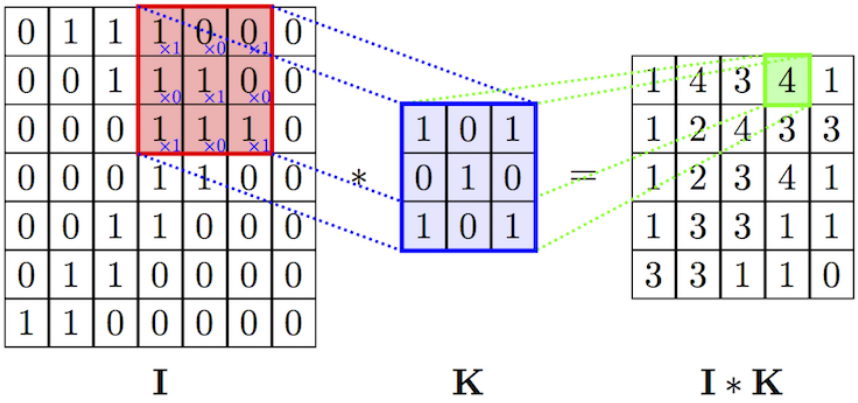


Abb. 20: Filter [12]

$$(I * K) = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} * I_{x+i-1,y+j-1} \tag{19}$$

Durch die Aktivierungsfunktionen der Neuronen in Abb. 19 (b) kann eine Nichtlinearität erzeugt werden, wodurch sich die Informationen der einzelnen Pixel ändern. Um die Matrizengrößen zu verkleinern und die wichtigsten Informationen herauszufiltern, besitzt die Beispielebene in Abb. 19 (b) vor ihrem Ausgang eine Vereinigungsebene. In den meisten Fällen ist das eine Max-Pooling-Ebene. In Abb. 19 (b) ist die Filtergröße der Vereinigungsebene eine (2×2)-Matrix und die Schrittweite *strides* ist 2, sodass keine Überlappung der Filtermatrizen beim Scannen entsteht. Die Ausgangsmatrix ist in dieser Konstellation nach dem Scannen um die Hälfte kleiner. Ein Beispiel dazu ist in Abb. 21 dargestellt. Das Ziel dieses Schrittes ist das Komprimieren der Matrizen. Die erste faltende Ebene hat die Eingangsmatrix auf 6 Matrizen mit der Größe (14×14) geändert. [16]

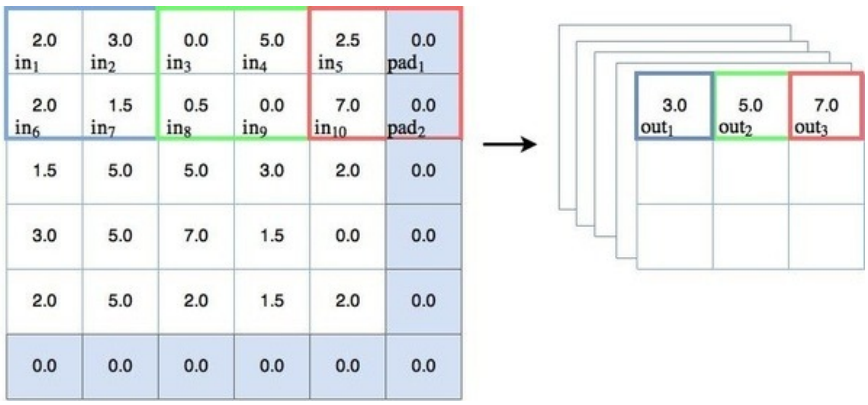


Abb. 21: Max-Pooling [13]

CNN's können aus beliebig vielen Sequenzen dieser Faltungsebenen bestehen. Die Ebene, die diesen Faltungsebenen nachgeschaltet ist, ist eine voll verbundene Ebene und ist mit der Eingangsebene des Multilayer Perzeptron gleichzusetzen. Wie in Kap. 2.3.1 beschrieben sind 1D Vektoren für den Eingang in das neuronale Netz vorgesehen. Ein vor der voll verbundenen Ebene geschalteter Flatten-Layer, auf deutsch abflachende Ebene, überführt die 2D-Matrizen in 1D-Vektoren. Ein Beispiel dieser abflachenden Ebene ist in Abb. 22 dargestellt. Die Zeilen der Matrix werden reihenweise übereinander gesetzt und verknüpft. [16]

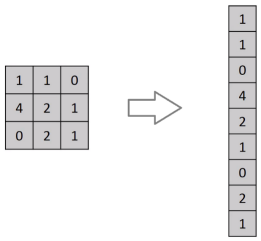


Abb. 22: Flatten-Layer [14]

2.3.3 Preprozessing von Bildern

Das Abstrahieren der Input- und Outputdaten ist notwendig, wenn die Daten nicht in dem von neuronalen Netzwerken lesbaren Formaten vorliegen. Ziel ist es, das vorliegende Datenformat auf eine vereinfachte und verallgemeinerte Repräsentation zu reduzieren. Dies kann auch durch das Entfernen von Attributen und die Konzentration auf eine Reihe von ausgewählten Attributen sein. Ein Beispiel für eine Abstraktion kann anhand eines Klassifizierungsalgorithmus, der Straßenschilder klassifiziert, erläutert werden. Hierzu steht ein Datenset in Form von Bilder mit unterschiedlichen Größen und Farbschemen zur Verfügung. In Abb. 23 ist ein solches Eingangsdatenset dargestellt. Über den Bildern ist jeweils die Klasse und daneben die Anzahl der zur Verfügung stehenden weiteren Bildern beschrieben.

Abb. 23: Verkehrsschilder [19]

Für die Bilderwiedergabe wird häufig das RGB-Farbmodell verwendet. Bei diesem Modell werden die drei Grundfarben Rot, Grün und Blau gemischt. Dabei ist jeder der drei Farben ein Wert von 0 bis 255 für die jeweilige Intensität zugeordnet. Je höher die Zahl ist, desto mehr ist diese Farbe in die Gesamtfarbe eingemischt. Null bedeutet, dass diese Farbe nicht eingemischt ist. Um die Farben eines kompletten Bildes zu beschreiben, besitzt jedes Pixel des Bildes einen Farbcode mit genau drei Werten. Übertragen auf ein Bild liegt dieses als Tensor mit der Form (Breite in Pixel, Höhe in Pixel, RGB-Werte) dar. In Abb. 24 ist ein Beispiel der Gestalt eines Bildes als Tensor mit der Form $(4 \times 4 \times 3)$ dargestellt.

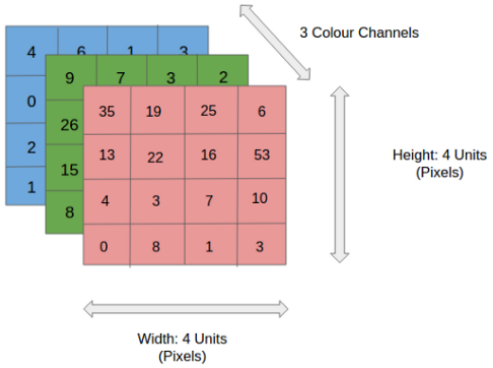


Abb. 24: Tensor (4 x 4 x 3) [21]

CNN's sind in der Lage, solche Tensoren zu verarbeiten. Die Inputdaten müssen alle die gleiche Größe besitzen. Der nächste Schritt ist das Skalieren der Bilder auf eine einheitliche Größe. In Abb. 25 ist dieser Schritt dargestellt. Auf der linken Seite sind die Bilder in Originalgröße mit verschiedenen Gestalten (Im Bild: shapes) dargestellt. Auf der rechten Seite sind die Bilder mit einer skalierten Pixelanzahl in der Gestalt $(28 \times 28 \times 3)$ dargestellt.

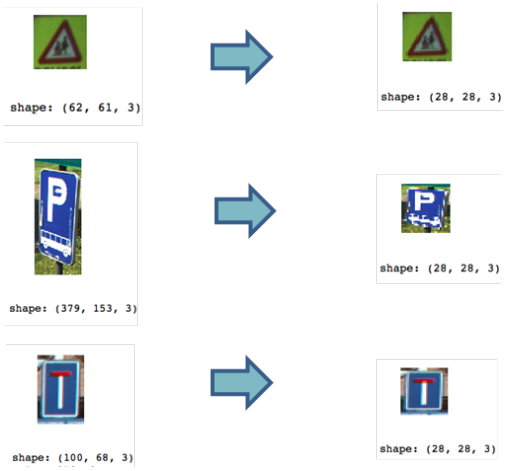


Abb. 25: Bildergestalten [20]

Um die Inputdaten zu vereinfachen, kann die Dimensionalität der RGB-Werte reduziert werden. Durch ein Graustufenverfahren kann die Dreidimensionalität in die erste Dimension überführt werden. Das heißt, die drei Zahlenwerte werden auf eine Zahl reduziert. Graustufenwerte können alle Werte im Wertebereich zwischen 0 und 1 annehmen, wobei 0 die Farbe weiß und 1 die Farbe schwarz bedeutet. In Abb. 26 links ist das Ergebnis der Graustufung dargestellt. Eine Möglichkeit besteht darin, die Werte in Binärschritte zu abstrahieren. Dies bedeutet, dass alle Werte unter einem gesetzten Schwellenwert 0 sind, die darüber 1. Das Ergebnis ist im Falle ein Schwarzweißbild, bei dem nur noch Konturen sichtbar sind. In Abb. 26 rechts ist ein Binärschritt bei dem Wert 0,5 zu sehen. Die Abstraktion von Bildern besteht in diesem Fall darin, die Informationen der einzelnen Pixel zu reduzieren.

Abb. 26: Bilder in Graustufe (links) und in Schwarzweiß (rechts) [20]

3 Detektieren und Extrahieren der Flanschflächen

Unabhängig vernetzte Karosserieelemente in der Crash-Simulation erfordern wie auch in der Realität Verbindungstechnologien, die den Festigkeiten und Steifigkeiten der Realverbindung angenähert sind. Flansche sind die Überdeckungsbereiche der Bauteile, die in Kontakt mit dem anderen Bauteil sind und auf denen die Verbindungstechnik wie z.B. Schrauben, Schweißnähte, Klebenähte und weitere Verbindungstechnologien angebracht sind. Die Flanschbereiche übertragen die Kräfte zwischen den Bauteilen und sind Parallelflächen im 3D-Raum.
In diesem Kapitel ist ein Lösungsweg dargestellt, dessen Ergebnis die extrahierten Flanschflächen der verbundenen Bauteile beinhaltet. Diese Flanschfächen werden für die weiteren Schritte in dieser Arbeit benötigt und als Punktewolken sowie Elementdefinitionen separat gespeichert.

3.1 Parsing von Keywords in LS-Dyna-Inputdecks

Die Trainingsdaten für das neuronale Netz sollen aus den Inputdecks von vergangenen Modellen entstammen. Hierfür liegt ein LS-Dyna-Inputdeck des Toyota Yaris vor, welches alle metallischen Karosseriebauteile (Klappen ausgenommen) und deren Verbindungen beinhaltet. In Abb. 27 ist die Geometrie dessen dargestellt.

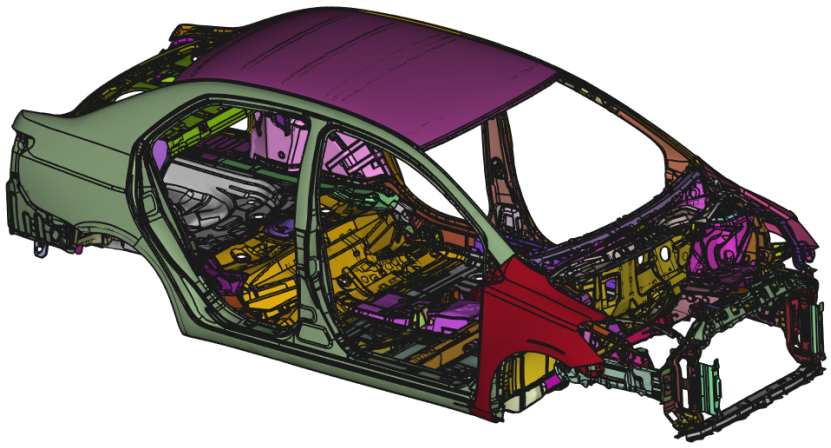


Abb. 27: Toyota-Yaris-CAD

Der erste Schritt ist die Informationsextraktion aus dem vorliegenden Inputdeck. Dafür ist eine Syntaxanalyse notwendig, die in der Lage ist, das Inputdeck auf die vom Anwender gewünschten Keywords zu durchsuchen und die relevanten Informationen separat abzuspeichern. Syntaxanalysealgorithmen, auch Parser genannt, sind Programme, die Texte oder Codes zerteilen und die essentiellen Informationen daraus für den Compiler, der Befehle für die Maschine übersetzt, ausliest. Der in dieser Arbeit verwendete Parser ist von der Firma SCALE GmbH speziell für Anwendungen mit LS-Dyna-Inputdeck bereitgestellt und in Python 2.7 anwendbar. Das Ergebnis aus der Anwendung des Parsers sind die extrahierten Element- und Knotendefinitionen der einzelnen Bauteile in Form von x-,y- und z-Koordinaten. In einem weiteren Schritt sind die Punktschweißverbindungen in Form von deren Koordinaten und den Bauteilkombinationen extrahiert, die sie verbinden.

3.2 Identifizieren der Kontaktelemente

Um die Flanschbereiche der Bauteile zu extrahieren, sind im Folgenden Eigenschaftskriterien für die Charakterisierung dieser Bereiche dargestellt. Diese Beschreibungen sind dann in algorithmisch verwertbare Formulierungen in Python übersetzt.

Kriterium 1:

Ein finites Element gehört zu einem Flansch, wenn sich ein Knoten des Elementes innerhalb eines definierten Radius um einen Knoten des Kontaktpartners befindet. Für die Definition des Radienwertes wird in dieser Arbeit die Annahme getroffen, dass sich die Außenflächen der Bauteile in direktem Kontakt miteinander befinden. Übliche Blechdicken s in aktuellen Karosserien befinden sich im Bereich von 0,5 mm bis 1,5 mm für Stahl und 1 mm bis 2 mm für Aluminium. Konservativ wird deshalb die Annahme getroffen, dass die Mittelflächen der beiden Bauteile also maximal mit $s = 2$ mm voneinander entfernt liegen. Zusätzlich wird die längste Elementkante der zwei Bauteile durch eine Schleife mit Gleichung (20) über alle Elemente errechnet.

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{20}$$

Aus diesen beiden Werten wird mit Gleichung (21) der Wert des Radius r für die jeweiligen Kontaktpartner festgelegt.

$$r = \sqrt{(s^2 + l^2)} \tag{21}$$

In Abb. 28 ist eine Prinzipskizze des Verfahrens dargestellt. Elemente, die einen Knoten besitzen, der innerhalb des Radius ist, werden als Flanschelement betrachtet und als solches abgespeichert. Der Prozess kann auch auf das gegenüberliegende Bauteil angewendet werden. Das Resultat sind zwei Flanschnetze wie in Abb. 30 zu sehen ist.

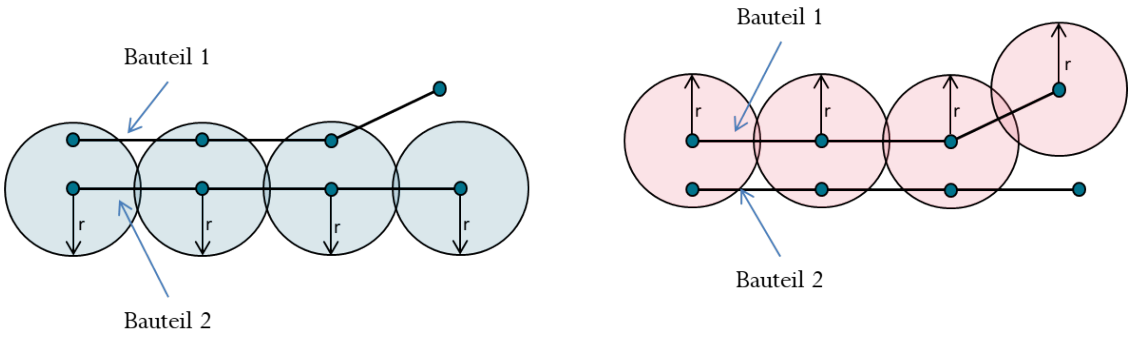


Abb. 28: Radien um Punkte

Kriterium 2:

Ein finites Element gehört zu einem Flansch, wenn der Winkel γ zwischen den sich gegenüberliegenden Elementnormalen einen definierten Grenzwert nicht übersteigt. Hierfür wird in dieser Arbeit die Annahme getroffen, dass die Flanschflächen der beiden Kontaktpartner parallel sind. Die Elementnormale wird über die Elementebene aus den Vektoren von zwei Elementkanten errechnet. Der Winkel zwischen den beiden Normalen wird über das Skalarprodukt der beiden Normalenvektoren $\vec{a}$ und $\vec{b}$ errechnet. Das Skalarprodukt aus zwei Vektoren, die sich im euklidischen Vektorraum befinden, ist von dem Winkel der zueinander stehenden Vektoren und dessen Beträgen abhängig und berechnet sich im kartesischen Koordinatensystem mit Gleichung (22).

$$\gamma = \arccos \frac{\vec{a} * \vec{b}}{|\vec{a}| * |\vec{b}|} \tag{22}$$

In Abb. 29 ist dieses Kriterium verbildlicht. Der Winkel γ zwischen Element $E3$ und Element $E6$ übersteigt den errechneten Grenzwinkel. Diese beiden Elemente sind somit als Flanschelement ausgeschlossen.

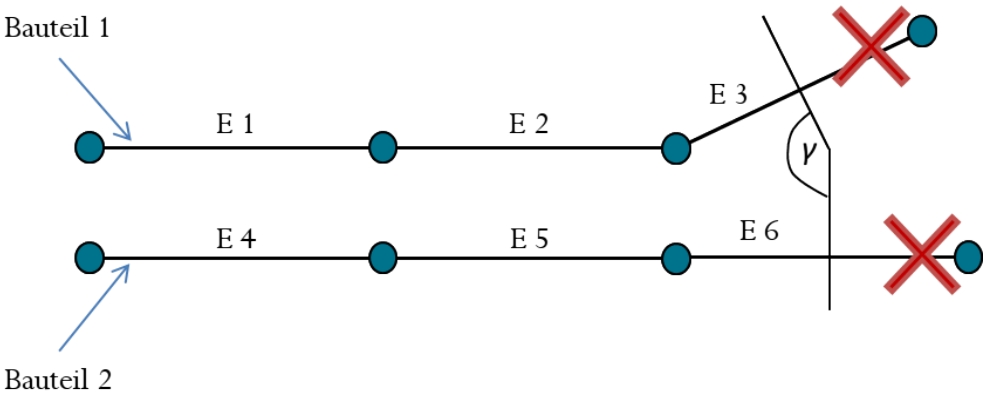


Abb. 29: Winkel zwischen zwei Elementen

Für die Ermittlung des Grenzwinkels ist der Durchschnitt aller Elementwinkel, der sich gegenüberliegenden Flanschelemente errechnet. Der Grenzwinkel kann vom Benutzer für spezielle Fälle eigenständig im Code geändert werden, falls der Bedarf besteht. Die Änderung hängt direkt mit der Genauigkeit der Flanscherkennung zusammen.

Kriterium 3:

Elemente, die die ersten beiden Definitionen erfüllen aber nicht mit mehr als drei Elementen eine zusammenhängende Elementgruppe bilden, gehören nicht zu einer Flanschfläche.

Durch das Umwandeln der 3D-Bauteile in Mittelflächenschalen für die Berechnung, entsteht ein Freiraum zwischen den sich überdeckenden Bauteilen, der sich aus der Summe der beiden Bleckdicken geteilt durch zwei ergibt. Dadurch gibt es zwei Bereiche, die beim Extrahieren der Flanschüberdeckung entstehen. Die Flanschüberdeckung auf Bauteil 1 und die auf Bauteil 2. Diese entsprechen nicht der Flanschfläche, die im Kontakt der beiden Bauteile liegt, da sich die beiden extrahierten Überdeckungen auf den jeweiligen Mittelebenen befinden. Es gibt zwei Varianten die Flanschflächen, die im Kontakt der beiden Bauteile liegt, zu errechnen. In Abb. 30 auf der rechten Seite ist ein Beispiel von zwei extrahierten Flanschelementen der gleichen Bauteilkombination dargestellt.
In Abb. 30 ist eine mit den beschriebenen Kriterien extrahierte Flanschfläche der Bauteile 485_*wheelwellrearouterR* mit der PID 216 und 123_*cpillarintersupportR* mit der PID 139 des Toyota Yaris dargestellt.

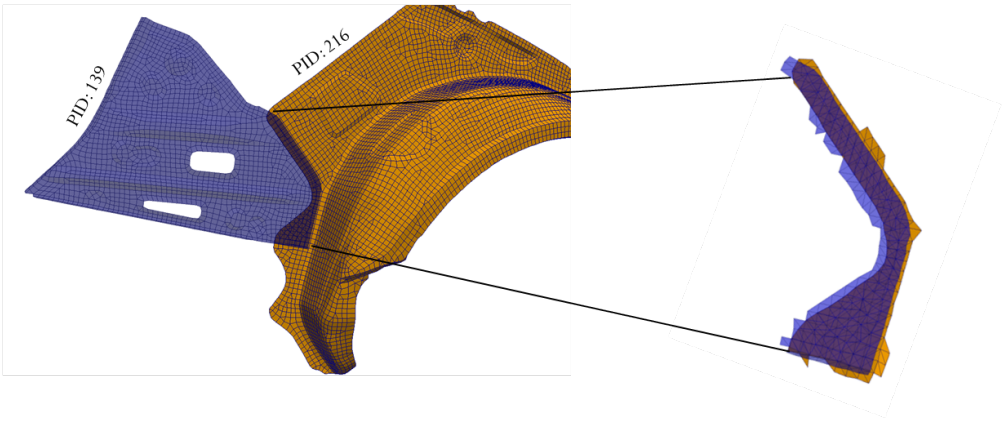


Abb. 30: Flanschextraktion aus zwei Mittelflächen

3.3 Ergebnis der Flanschextraktion

Das Ergebnis des in Abschnitt 3.2 beschriebenen Algorithmus auf das Gesamtmodell des Toyota Yaris angewendet ist in Abb. 31 dargestellt. Bei der Extraktion aller möglichen Flansche, die alle beschriebenen Kriterien erfüllen, sind weitaus mehr einzelne Flansche entstanden als es Bauteilkombinationen gibt. Das liegt an der Tatsache, dass dieselbe Bauteilkombination

mehrere Flanschbereiche aufweisen kann. Ein Beispiel hierzu ist die B-Säule, die bei der Karosseriebauart in Schalenbauweise aus zwei oder mehreren zusammengefügten Umformblechen. Diese Bleche sind als Näpfe ausgeformt und besitzen auf beiden Seiten jeweils eine Abstellung, die den Flansch darstellt.

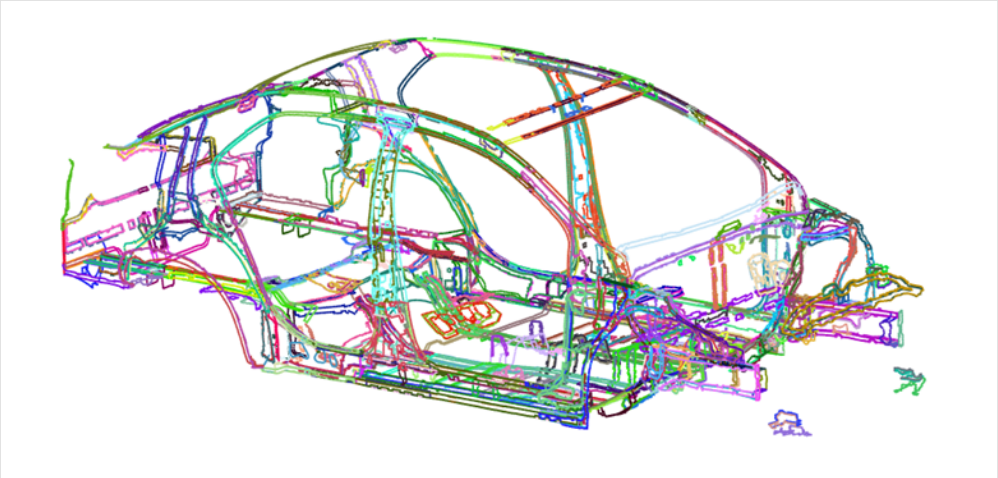


Abb. 31: Extrahierte Flanschbereiche des Toyota-Yaris

4 Teilautomatische Generierung von Schweißpunkten in CAX-Anwendungen

Im Folgenden ist ein Algorithmus beschrieben, der teilautomatisiert Schweißpunkte auf den extrahierten Flanschen platziert.

4.1 Point-Picking-Algorithm

Point-Picking-Algorithmen sind Algorithmen, die zum Beispiel Punkte nach gesetzten Randbedingungen aus Punktewolken mit einen möglichst gleichgroßen Abstand zueinander auswählen. Diese Art von Algorithmen können zum Beispiel in Design of Experiment (DOE) - Anwendungen verwendet werden, um die Anzahl der durchzuführenden Experimente zu reduzieren[37]. Hierzu wird versucht, durch möglichst wenige Experimente den Einfluss der Wechselwirkungen von einzelnen Parametern auf die Zielgröße zu bestimmen. Durch Picking-Point-Algorithmen können diese durchzuführenden Experimente bestimmt werden. In dieser Arbeit wird ein Point-Picking-Algorithmus verwendet, um aus einer gegebenen Punktewolke Punkte auszusuchen, die einen definierbaren Abstand zum Rand der Punktewolke und die maximale Distanz zueinander besitzen. Als Input hierfür dienen die Punktewolken der extrahierten Flanschnetze. Dadurch ist es möglich, Punktschweißungen möglichst gleich verteilt auf einem Flansch mit wenig Aufwand zu platzieren.

In Abb. 32 ist ein Ergebnis des in dieser Arbeit verwendeten Point-Picking-Algorithmus dargestellt, welcher auf den extrahierten Flansch aus Abschnitt 3.2 angewendet ist. Das Vorgehen des Algorithmus ist wie folgt gegliedert. Der Input ist die Punktewolke, welche in blau dargestellt ist sowie die Anzahl der zu verteilenden Punkte anzugeben. In dem Beispiel ist die zu verteilende Anzahl an Punkten auf fünf festgelegt. Die Randbedingung *Abstand zum Rand* wird durch einen Filter realisiert, der alle Randpunkte in diesem Abstand löscht, sodass sie nicht mehr zur Auswahl bereitstehen.

Beim Auswahlverfahren der Punkte werden zunächst die am weitesten auseinander liegenden Punkte berechnet. Hierzu wird das Modul *scipy.spatial.distance.cdist* aus der Scipy[1] Bibliothek verwendet. Das Modul kann eine Längenmatrix aufstellen, die Abstände aller Punkte innerhalb einer gegebenen Punktewolke beinhaltet. Dabei sind die Punktindizes sowohl auf der Abszisse als auch auf der Ordinate gleich sortiert, weswegen die Hauptdiagonale der Matrix die Distanzen 0 enthält. Aus dieser Matrix werden dann über eine Suchfunktion die beiden Zahlen ermittelt, die am Weitesten von einander entfernt sind. In Abb. 32 sind das die rot markierten Punkte, die am Weitesten von einander entfernt liegenden. Im Anschluss

[1]Scipy ist eine Pythonbasierte Open-Source-Software für mathematische und wissenschaftliche Anwendungen

werden die restlichen Punkte aus der Matrix gesucht, die mit Abzügen von Toleranzen aufgrund der endlichen Anzahl an Punkten, annähernd gleiche Abstände besitzen. Da insgesamt fünf Punkte ausgewählt werden sollen, sind das also die restlichen drei, die ebenfalls in rot dargestellt sind und zwischen den beiden Punkten mit dem maximalen Abstand zueinander liegen.

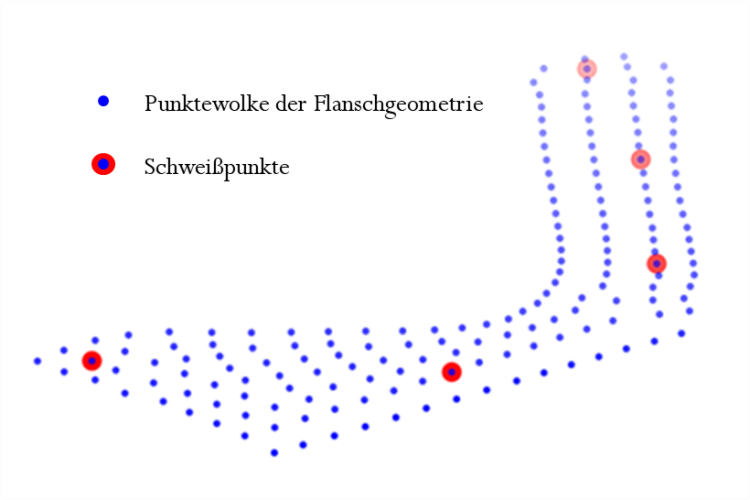


Abb. 32: Picking-Point-Algorithm

4.2 Visualisieren der Schweißpunkte im CAD (FreeCAD)

Für den Anwender ist die Visualisierung und die Konvertierbarkeit der erzeugten Schweißpunkte unerlässlich, um diese im Gesamtprozess bewerten und weiterverarbeiten zu können. In dieser Arbeit sind die automatisch generierten Schweißpunkte beispielhaft in FreeCAD[2] dargestellt. FreeCAD besitzt ein pythonbasiertes *applicationprogramminginterface* (API), mit der die Software gesteuert werden kann. Um die Python-API zu verwenden ist eine Installation der FreeCAD-Bibliothek als Nebenpaket notwendig. Die Installationsanleitung und eine umfangreiche Dokumentation ist unter [17] zu finden.

Die Karosseriebauteile sind als Mittelflächen der Bleche dargestellt. Die erzeugten Schweißpunkte sind durch senkrecht zur Mittelfläche ausgerichtete Zylinder dargestellt. In Abb. 33 ist das Ergebnis des Algorithmus mit sechs zu verteilenden Punktschweißungen zu sehen. Die Schweißpunkte sind als graue Zylinder auf den farbigen Mittelflächen der Bleche mit PID 216 (pink) und PID 139 (violett) dargestellt. Die definierten Zylinder können in weitere

[2]FreeCAD ist eine Open-Source-CAD-Software mit einer Python-API

Anwendungsprogramme wie Preprozessoren oder produktionstechnische Analyseprogramme konvertiert werden.

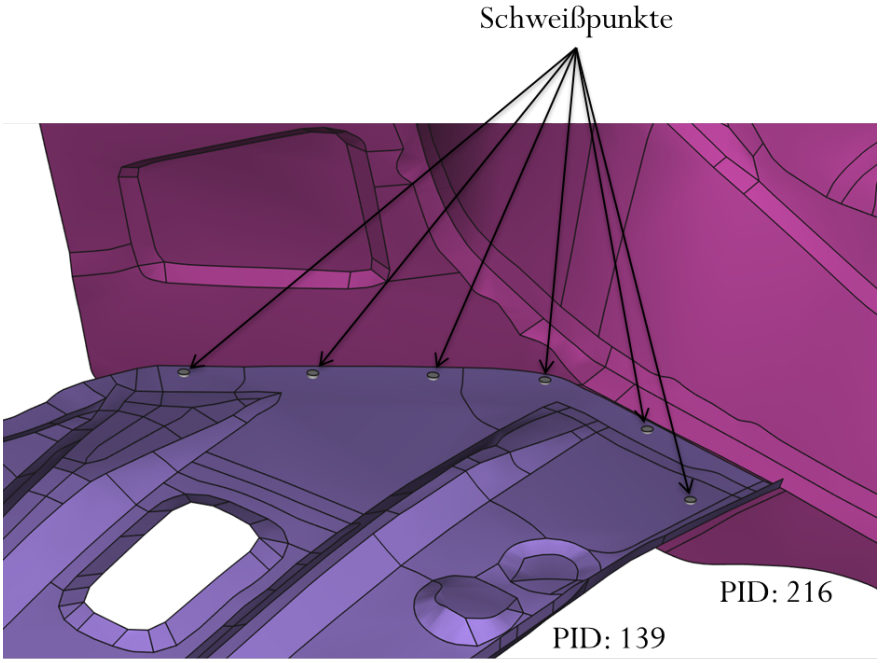


Abb. 33: Schweißpunkte in FreeCAD

5 Vorhersage der Schweißpunktlage

Um Kosten für langwierige manuelle Prozesse einzusparen, besteht ein berechtigtes Interesse darin, das Definieren von Punktschweißverbindungen in CAE-Anwendungen zu automatisieren. Eine Methode ist es, ein Expertensystem zu entwickeln, das Wissen von vorhergegangenen Modellreihen besitzt und dieses auf neue, noch nicht bekannte Modelle anwenden kann. Dazu sind im Folgenden zwei Strategien dargestellt, die jeweils eine Methodik für ein solches Expertensystem beinhalten. Strategie 1 ist dabei nur als Prozessschema dargelegt.

5.1 Strategie 1

Diese Strategie beinhaltet zwei Schritte. Im ersten Schritt werden die Flanschgeometrien klassifiziert und im zweiten Schritt die klassifizierten Flanschgeometrien mit den jeweiligen Altdaten durch einen Mapping-Algorithmus[3] übereinander gelegt und die Schweißpunkte übertragen.
Das Klassifizieren oder auch Kategorisieren von Daten ist ein Prozess, der Objekte mit unterschiedlichen Attributen in Klassen einteilt. Eine Klasse fasst die Objekte zusammen, die gesetzten Bedingungen der Klasse entsprechen. Ein einfaches Beispiel aus der Evolution ist das menschliche Klassifizieren von Reizen aus der Umwelt in gefährlich oder ungefährlich. In diesem Fall gibt es zwei Klassen. Um Probleme detaillierter aufzuschlüsseln, gibt es in der Realität häufig mehr als nur zwei Klassen, wodurch der Komplexitätsgrad steigt. Systeme, die in der Lage sind, Objekte automatisiert in solche Klassen einzuteilen, sind Klassifizierungs-Algorithmen. Das Ziel von Klassifizierungs-Algorithmen ist die Ermittlung einer Funktion, um zu beschreiben, welcher Klasse ein Objekt untergeordnet ist. Zur Klassifizierung von Daten gibt es verschiedene Methoden. Im Folgenden soll ein Überblick über ein drei dieser Methoden dargestellt werden.

[3]Mapping ist eine Datentransformation zwischen einer Datenquelle und einem Datenziel

K-Nearest Neighbors:

Bei diesem Verfahren sind die Trainingsdaten in n-dimensionale Werte abstrahiert. Dadurch entstehen Gruppierungen von Werten, die als Klassen definiert sind. In Abb. 34 ist ein Beispiel zu einer Klassifizierung durch das K-Nearest Neighbours Verfahren dargestellt. Die Trainingsdaten bestehen aus bekannten Objekten, die einer Klasse zugewiesen sind. In diesem Fall sind es zwei Klassen, eine rote und eine blaue. Ein neues Objekt, das als weißer Punkt mit einem Fragezeichen dargestellt ist, soll klassifiziert werden. Dabei werden die k nächsten Objekte herangezogen. Das neue Objekt wird der Klasse zugeordnet, deren bekannte Objekte dem Mehrheitskriterium entsprechen. Der Parameter k kann vom Anwender festgelegt werden und sollte eine ungerade Zahl sein, um einen Gleichstand grundsätzlich auszuschließen. Je größer k gewählt wird, desto größer ist die zugelassene Toleranz. Das Verfahren liefert immer eine eindeutige Lösung. Ein Nachteil dieses Verfahrens ist die Ungenauigkeit bei wenigen Trainingsdaten und die Tatsache, dass der Klassifikator immer alle Trainingsdaten im Speicher behält, was bei großen Datenmengen den Aufwand erhöht. Vorteilhaft ist die Nachvollziehbarkeit des Verfahrens.[18]

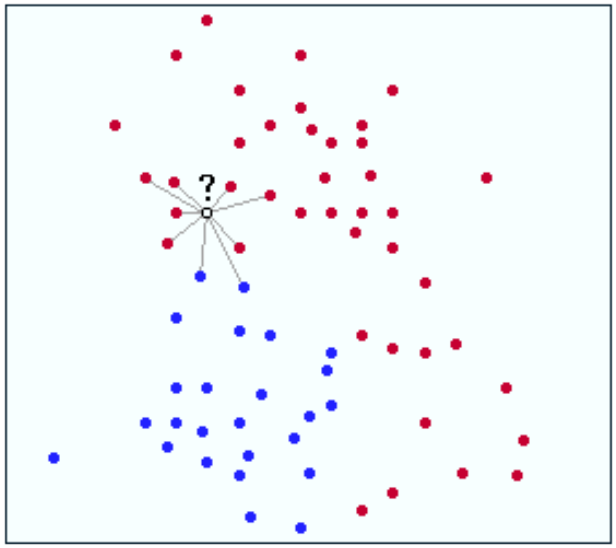


Abb. 34: K-Nearest-Neighbours[18]

Klassifizierungsbaum:

Die Klassifizierungsmethode des Entscheidungsbaumes basiert auf der Grundlage verschiedener Kriterien, die in immer kleinere Teilmengen unterteilt sind. Eine Aufteilung des Datensatzes erfolgt immer nach bestimmten Attributen[4] der Objekte. Dies ist durch ein Beispiel in Abb. 35 dargestellt. Die Trainingsobjekte sind in diesem Fall Formen, die jeweils v und w als Attribute besitzen. Für jedes Attribut gibt es zwei oder mehrere Zweige im Klassifizierungsbau. Die Trainingsobjekte geben dem Klassifizirungsbaum so seine Form. Die Klassifizierung neuer Objekte erfolgt durch das Durchlaufen des Baumes mit den jeweiligen Attributen der Objekte. Übertragen auf die in dieser Arbeit verwendeten Flanschgeometrien, können mögliche Attribute die Größe, Form, Verbindungstechnologie, etc. sein. Diese Methode ist durch die Verwendung von If - Anweisungen in Maschinensprache umsetzbar. Durch die hirarchische Struktur ist dieses Verfahren gut nachvollziehbar und liefert immer eindeutige Ergebnisse. Der Nachteil ist eine mit der Anzahl der Attribute pro Objekt größer werdende Komplexität.

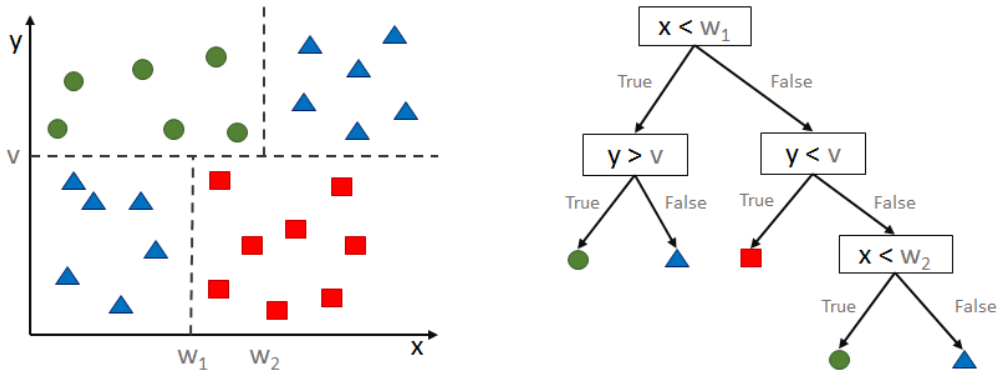


Abb. 35: K-Nearest-Neighbours[38]

Neuronale Netze:

Eine Klassifizierung ist auch durch neuronale Netze möglich. Die Funktionsweise von neuronalen Netzen ist bereits in Abschnitt 2.3.2 dargelegt. Typische Anwendungen von neuronalen Netzen für Klassifizierungsprobleme sind zum Beispiel die Erkennung von Bildern, Spracherkennung und Sprachsynthese. Eine nachgeschaltete Aktivierungsfunktion kann die Werte in die gewünschten Wahrscheinlichkeiten umwandeln. Hierfür kommt die Softmax-Funktion in Frage. Die Softmax-Funktion ist auch als normierte Exponentialfunktion bekannt.

[4]Attribute sind Eigenschaften von Objekten. z.B.: Objekt=Buch{Attribut=Titel, Attribut=Autor}

Um das Endergebnis der Klassifikation zu ermitteln ist es notwendig, den Wert $max(V_2)$ der Ausgabeebene zu ermitteln. Dieser sucht den Maximalwert und reduziert den Ergebnisvektor V_2 auf einen Wert, der die Klasse darstellt.
In [39] sind bereits Untersuchungen mit dem Klassifizieren von Geometrien mit neuronalen Netzen dargelegt. Die Klassifizierung in dieser Arbeit basiert auf der Methode des PointNets [22]. Die darin beschriebene neuronale Netzarchitektur ist in der Lage, Punktewolken zu segmentieren und zu klassifizieren. Die Eingangsdaten sind Punktewolken mit einer festgelegten Anzahl an Punkten mit x-, y- und z-Koordinaten [39]. Im Weiteren ist beschrieben, wie die Flanschdaten aufzubereiten sind, um als Eingangsdaten für das PointNet verwendet werden zu können.
In dieser Strategie wird mit dem Klassifizieren durch das neuronale Netz verwendet, da die Daten bereits als Punktewolke vorliegen und somit nicht mehr weiter Abstrahiert werden müssen.

5.1.1 Prozessablauf

Die Methodik basiert auf dem beschriebenen Klassifizierungsalgorithmus und einem darauffolgenden Mappingalgorithmus. Dazu ist eine gespeicherte Datenbank, die aus den Altdaten besteht, notwendig. Diese Altdaten müssen die Flanschgeometrie in Form von Punktewolken sowie die positionierten Schweißpunkte beinhalten. Als Altdaten kommen nur solche Karosseriestände in Frage, die den folgenden Anforderungen entsprechen.

- Ausreichende Festigkeit in Crash-Simulationen und/oder Crashversuchen
- Eine produktionstechnische Machbarkeit
- Wirtschaftliche Aspekte

Datenstände, die diesen Anforderungen entsprechen, sind zum Beispiel Karosseriestände, die eine Produktionsfreigabe erhalten haben. Um keine Datenstände mit in den Prozess einzubinden, die als nicht praktikabel gelten, sollten also nur solche Datenstände in der Datenbank abgelegt sein, die den Status $Produktionsfreigabe$ besitzen.

Die Flansch- und Schweißpunkdaten können in Form von Punktewolken mit x-, y- und z-Koordinaten in jedem beliebigen Format, das von existierenden Compilern schreib- und lesbar ist, abgespeichert sein. In dieser Arbeit sind die Daten im ascii-Format als $.json$ Dateien aus Gründen der einfacheren Kontrolle der Inhalte abgespeichert. Da bei dieser Methodik große Datenmengen entstehen können, ist für die spätere Anwendung die Verwendung von binär-Formaten besser geeignet. Der Klassifizierungsprozess ist in Abb. 36 dargestellt. Auf der

linken Seite ist beispielhaft die zu klassifizierende Punktewolke, die in die Eingangsneuronen des neuronalen Netzes eingeht dargestellt. Das neuronale Netz gibt die Klasse aus, die mit dem höchsten Wahrscheinlichkeitswert der Klasse der Eingabewerte entspricht.

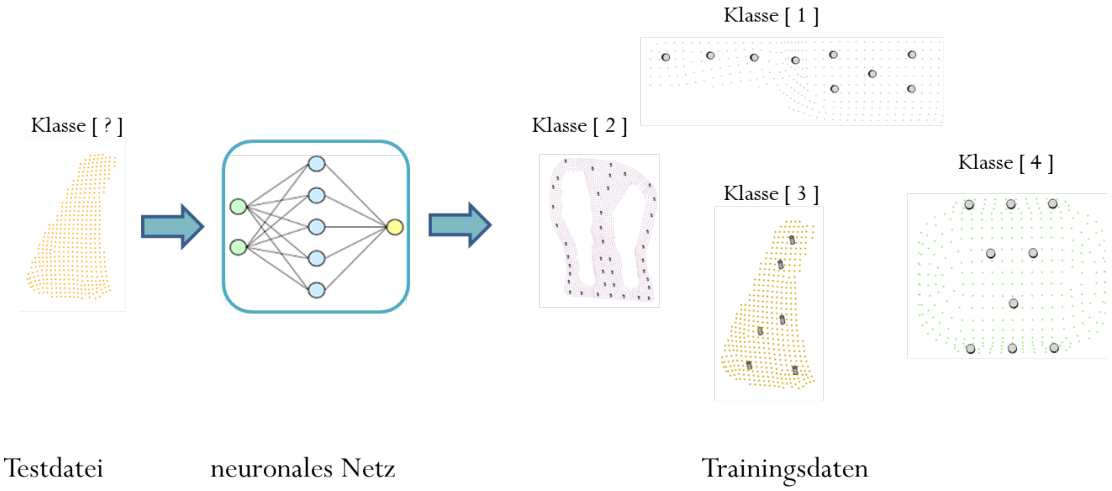


Abb. 36: Methode mit gespeicherte Trainingsdaten

Der nächste Schritt ist das Mapping der Schweißpunktverteilung aus der ermittelten Klasse der Trainingsdatenbank auf die klassifizierte Flanschgeometrie. In Abb. 37 ist das Übertragen der Schweißpunkte aus der Trainingsdatenbank auf die klassifizierte Flanschgeometrie schematisch dargestellt. Die eingekreisten Zylinder auf der linken Seite stellen die Schweißpunkte dar. Die Pfeile zeigen das Mapping auf die Punktewolke an, die der Mappingalgorithmus übernimmt.

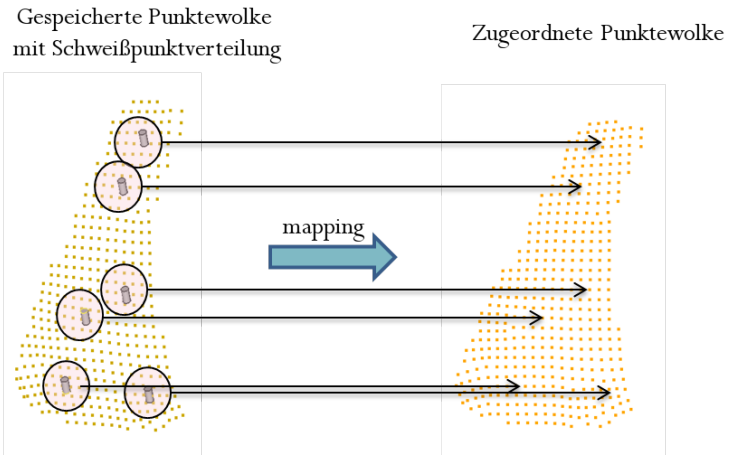


Abb. 37: Mapping von Schweißpunkten in CAX-Anwendungen

Für das Mapping sind die beiden Punktewolken zunächst auf einen Punkt zu referenzieren. Hierzu eignet sich der Ursprung des globalen Fahrzeugkoordinatensystems, das sich üblicherweise in der geometrischen Mitte der Vorderachse befindet. Als transformationsreferenz eignet sich der Flächenschwerpunkt beider Punktewolken. Das Verwenden von Verzeichnissen für die Punktewolken ist notwendig, um die Punkte zu jeder Zeit im Prozess eindeutig identifizieren zu können. Innerhalb von Verzeichnissen ist jedem Punkt eine Identitätserkennung (ID) zugewiesen, über die Punkte eindeutig identifiziert werden können.
Um den Flächenschwerpunkt zu ermitteln wird eine virtuelle Box errechnet, in der sich alle Punkte der Punktewolke befinden. Dazu sind die Werte ($x_{min,max}$, $y_{min,max}$, $z_{min,max}$) aus der Punktewolke jeder Koordinatenrichtung zu ermitteln. Diese Werte sind auch bekannt als Schachtelmaße. Anhand dieser Schachtelmaße kann der Flächenschwerpunkt ermittelt werden. In Gleichung (25) ist dargestellt, wie die x-, y- und z-Koordinaten des Flächenschwerpunkts ermittelt werden können.

$$x_s = x_{min} + \frac{x_{max} - x_{min}}{2} \tag{23}$$

$$y_s = y_{min} + \frac{y_{max} - y_{min}}{2} \tag{24}$$

$$z_s = z_{min} + \frac{z_{max} - z_{min}}{2} \tag{25}$$

Alle Punkte der Punktewolke müssen über die Gleichung (26) in den Ursprung des beschriebenen Koordinatensystems transformiert werden.

$$P_{trans} = (x - x_s, y - y_s, z - z_s) \tag{26}$$

Innerhalb dieses Systems muss ein $Best - Fit$ der beiden Punktewolke gefunden werden. Da es möglich ist, dass eine Flanschgeometrie kleiner ist als die andere, erfolgt eine Skalierung. Dabei ist es egal, welche der beiden Punktewolken auf die Größe der anderen skaliert wird. Die Faktoren s_x, s_y und s_z der Skalierung werden über den Quotienten aus den Kantenlängen beider Boxen, die die jeweilige Punktewolke komplett beinhalten, ermittelt. Die Skalierung der Punkte erfolgt durch die Skalierungsmatrix in Gleichung (27).

$$\begin{pmatrix} P_{x,scale} \\ P_{y,scale} \\ P_{z,scale} \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \cdot \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} \tag{27}$$

Der nächste Schritt ist die Rotation der beiden Punktewolken übereinander. Über die in Gleichung (28) und Gleichung (29) dargestellte Rotationsmatrix M_{rot} können Objekt im Raum rotiert werden, wobei der Winkel α die Drehung um die x-Achse , der Winkel β die Drehung um die y-Achse und der Winkel γ die Drehung um die z-Achse bestimmt.

- sinus = s
- cosinus = c

$$M_{rot} = \begin{pmatrix} c(\beta)c(\alpha) & -c(\beta)s(\gamma) & s(\beta) \\ c(\alpha)s(\gamma) + s(\alpha)s(\beta)c(\gamma) & c(\alpha)c(\gamma) - s(\alpha)s(\beta)s(\gamma) & -s(\alpha)c(\beta) \\ s(\alpha)s(\gamma) - c(\alpha)s(\beta)c(\gamma) & s(\alpha)c(\gamma) + c(\gamma)s(\beta)s(\gamma) & c(\alpha)c(\beta) \end{pmatrix} \tag{28}$$

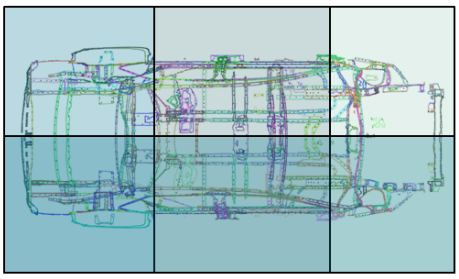
$$\begin{pmatrix} P_{x,rot} \\ P_{y,rot} \\ P_{z,rot} \end{pmatrix} = M_{rot} \cdot \begin{pmatrix} P_{x,scale} \\ P_{y,scale} \\ P_{z,scale} \end{pmatrix} \tag{29}$$

Um die Rotation zu finden, bei der die beiden Punktewolken so genau wie möglich übereinander liegen, ist eine Optimierung notwendig, die die optimalen Parameter des Rotationssystems findet. Die Parameter sind in diesem Fall die Winkel α, β und γ. Sie spannen einen 3D-Optimierungsraum auf. Optimal bedeutet in diesem Fall, dass ein ausgewähltes Residuum R minimal ist. Ein Residuum ist in der Nummerik eine Abweichung vom gewünschten Ergebnis. Dazu werden Näherungslösungen in eine Gleichung eingesetzt. Das Residuum R ist die Differenz der drei Punkte aus beiden Punktewolken, die am weitesten in der x-, y-, und z-Richtung vom Ursprung entfernt sind sein. PW1 sei die Punktewolke aus den Trainingsdaten und PW2 sei die zu rotierende Punktewolke aus der Testdatei.

$$R = \frac{|PW1_{x,max} - PW2_{x,rotmax}| + |PW1_{y,max} - PW2_{y,rotmax}| + |PW1_{z,max} - PW2_{z,rotmax}|}{3} \quad (30)$$

Nachdem ein Minimum des Residuums ermittelt ist, kann die ID der Punkte aus PW2 ermittelt werden, die den Schweißpunktkoordinaten aus PW1 am nächsten liegt. Damit ist die Lokation der Schweißpunkte mit einer Toleranz ermittelt. Die Toleranz ist von der Feinheit des Bauteilnetzes abhängig. Um diese Toleranz auszugleichen können die Schweißpunkte durch einen definierten Abstand zum Rand verschoben werden.
Unterschiedliche Bereiche der Fahrzeugkarosserie können andere Crash-, Herstellungs- oder wirtschaftliche Anforderungen besitzen. Zum Beispiel ist in der Fahrzeugfront ein anderes Crashverhalten erwünscht als im Dachbereich. Diesbezüglich sehen die Verbindungstechniken in den Fahrzeugbereichen unterschiedlich aus. Bemerkbar kann dies an der Anzahl, der Positionierung und den Herstellungsparametern sein. Da sich Flanschgeometrien aus unterschiedlichen Regionen des Fahrzeugs zum Beispiel der Front und dem Heck geometrisch ähnlich sein können, muss bei der Einteilung der Klassen aus den Trainingsdaten zwischen den verschiedenen Regionen unterschieden werden. In Abb. 38 ist auf der rechten Seite ein beispielhaftes Schema dargestellt, wie eine Einteilung der Karosserie in Regionen aussehen kann. Auf der linken Seite ist ein Flansch dargestellt, der in dieser Form zum Beispiel insgesamt sechsmal in derselben Karosserie in den Regionen Front, Mitte und Heck vorkommt. In der Realität kann diese Einteilung in Regionen je nach Fahrzeugtyp feiner sein.

Flanschgeometrie der Bauteilkombination
PID:230 und PID:243



Toyota Yaris aufgeteilt in Regionen

Abb. 38: Einteilung in Fahrzeugregionen

5.1.2 Erzeugen der Kontaktflanschfläche durch Projektion

Für die beschriebene Strategie ist es notwendig, im ersten Schritt die Daten in Punktewolken zu konvertieren. Der Algorithmus aus Abschnitt 3.2 extrahiert nur die Flanschelemente. Dazu ist ein Algorithmus geschrieben, dessen Eingang Elementdefinitionen und dessen Ausgang eine Punktewolke ist. Wie in Abschnitt 3.2 beschrieben wird nicht die reale Kontaktflanschfläche extrahiert. Die Methodik dieses Unterkapitels basiert auf der Projektion der Punkte auf den jeweils anderen Flansch. Dadurch werden die Konturungenauigkeiten aus der Extraktion, die in Abb. 39 außerhalb der roten Linie liegen, geglättet. In Abb. 39 ist die geglättete Kontur durch die rote Linie dargestellt.

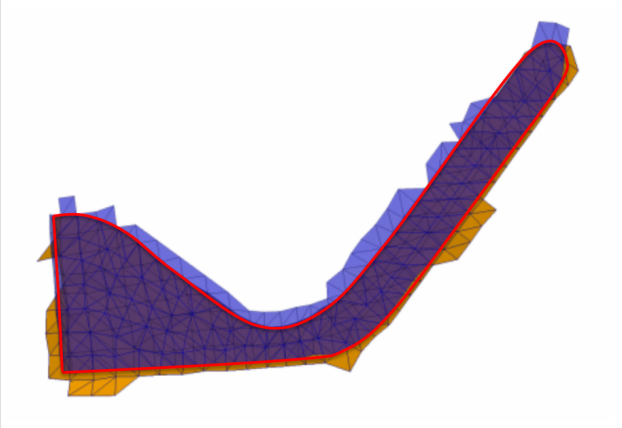


Abb. 39: Geglättete Flanschüberdeckung

Diese Variante hat zur Annahme, dass die Dicke der Elemente bekannt ist und aus dem Inputdeck für jedes Element extrahiert ist. Durch die definierten Elemente können die Elementebenen $\vec{x}$ mit der Gleichung (31) in Parameterform sowie die Elementnormalen $\vec{n}$ mit der Gleichung (32) bestimmt werden. Dabei sind $\overrightarrow{AB}$ *und* $\overrightarrow{AC}$ die Richtungsvektoren der jeweiligen Knotenverbindung und $\overrightarrow{OA}$ der Ortsvektor. Alle Punkte können dann mit dem Betrag der halben Blechdicke in die errechnete Elementnormalenrichtung $\vec{n}$ verschoben werden.

$$E : \vec{x} = \overrightarrow{OA} + \lambda * \overrightarrow{AB} + \mu * \overrightarrow{AC} \tag{31}$$

$$N : \vec{n} = \overrightarrow{AB} \times \overrightarrow{AC} \tag{32}$$

In Abb. 40 ist das Verfahren schematisch dargestellt. Jeder Knoten aus den Elementen des oberen Flansches wird in Richtung der ermittelten Elementnormalen $\vec{n}$ auf die Ebenen der Elemente des unteren Flansches projiziert. Die zu prüfenden Elemente werden innerhalb eines definierten Radius gesucht, um nicht alle Elementebenen durchlaufen zu müssen. Den Radius legt dabei der Anwender fest. Liegt der projizierte Knoten innerhalb von einem Element des anderen Flansches, so gehört dieser Punkt zum realen Flansch und wird abgespeichert.

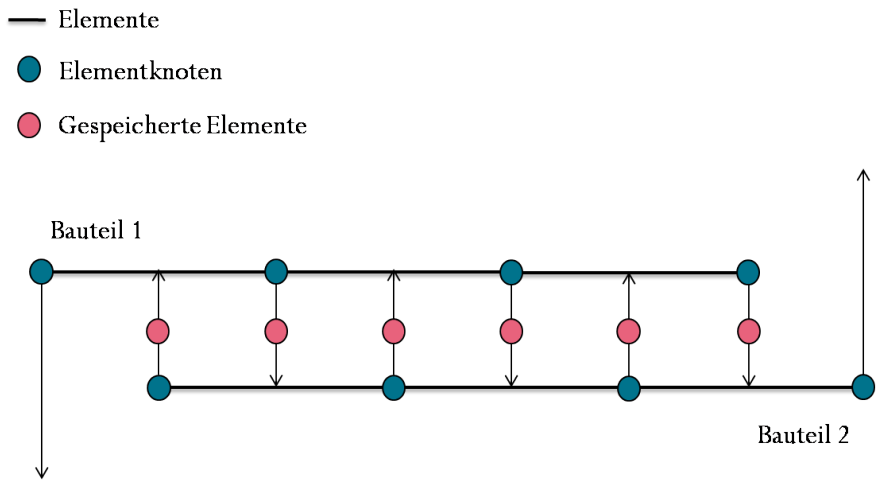


Abb. 40: Projizierte Punkte

Die Ermittlung, ob ein projizierter Punkt in einem Element liegt, kann über die Dreiecksfläche ermittelt werden. Ist die Summe der Flächen $A_{ABP} + A_{ACP} + A_{CBP} = A_{ABC}$ wie in Abb. 41 auf der linken Seite dargestellt, liegt der Punkt im Element. Ist die Summe der Flächen $A_{ABP} + A_{ACP} + A_{CBP} > A_{ABC}$ wie in Abb. 41 auf der rechten Seite dargestellt, liegt der Punkt nicht im Element. Die entstandenen Punkte liegen auf der Fläche des realen Flansches und werden abgespeichert. In Abb. 40 sind diese Punkte in rot dargestellt.

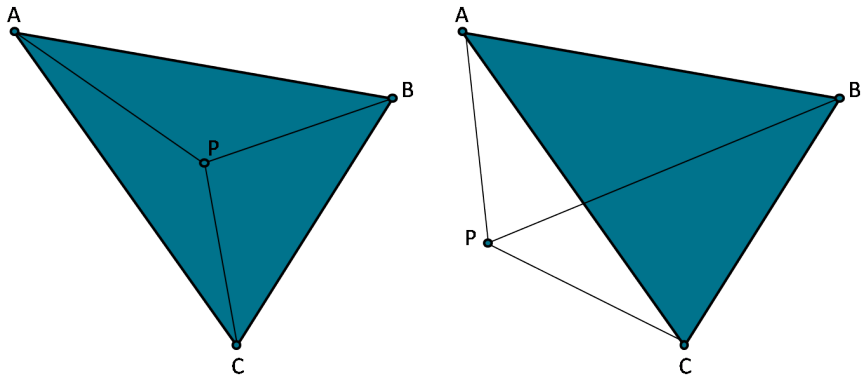


Abb. 41: Liegt P im Dreieck ABC

Das Ergebnis der ermittelten Kontaktflanschgeometrie ist in Abb. 42 dargestellt.

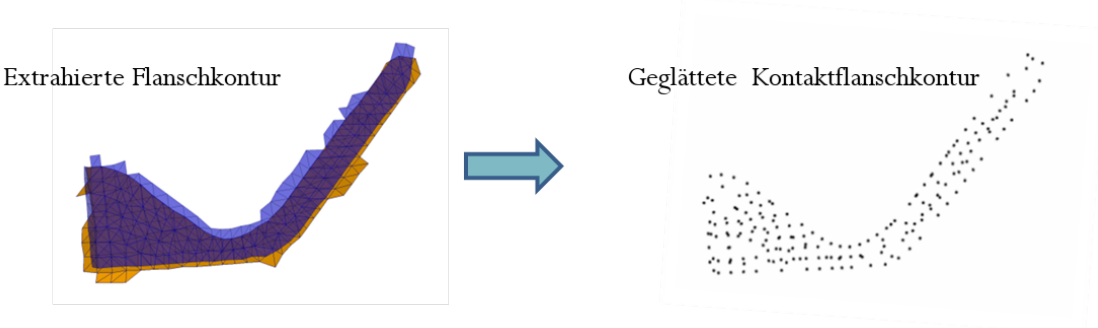


Abb. 42: Kontaktflanschgeometrie geglättet

5.1.3 Erzeugen von annähernd gleich verteilten Punkten

Für den in Abschnitt 5.1 beschriebenen Klassifizierungsprozess ist eine gleichverteilte und definierte Anzahl an Punkten auf allen zu klassifizierenden Flanschgeometrien notwendig. Dabei ist die ideale Anzahl in einer Parameterstudie herauszufinden, in der das neuronale Netz mit einer verschiedenen Anzahl an Punkten trainiert wird und die Genauigkeiten verglichen werden. Um die Kontaktflanschgeometrien mit einer definierten Anzahl an gleich verteilten Punkten zu generieren ist die folgende Methodik dargelegt.

Mithilfe des beschriebenen LHS-Algorithmus werden in dieser Masterarbeit die Skalare α_n abgetastet und diese Skalare für das Erzeugen von Punkten innerhalb der Dreieckselemente mit bayzentrischen Koordinaten verwendet. Bei einem Nichtverwenden des LHS-Algorithmus besteht die Gefahr, dass sich bei dem Erzeugen von mehreren Punkten innerhalb eines Elementes in gewissen Regionen Punktebündel bilden können. Das Ziel ist es, möglichst gleichverteilte Punkte auf der Flanschgeometrie zu erzeugen.
In Abb. 43 ist der Workflow dieser Methode dargelegt. Dabei wird ein beliebiger Punkt auf einem Element der Flanschgeometrie erzeugt und auf die zweite Flanschfläche projiziert. Liegt der Punkt in einem Element, wird der Punkt abgespeichert und gehört zur Flanschüberdeckung. Liegt der Punkt nicht auf einem Element des zweiten Flansches, wird ein neuer Punkt erzeugt. Die Schleife wird sooft durchlaufen, bis eine definierte Anzahl an Punkten gefunden wurde, die in einem gegenüberligenden Element liegt.

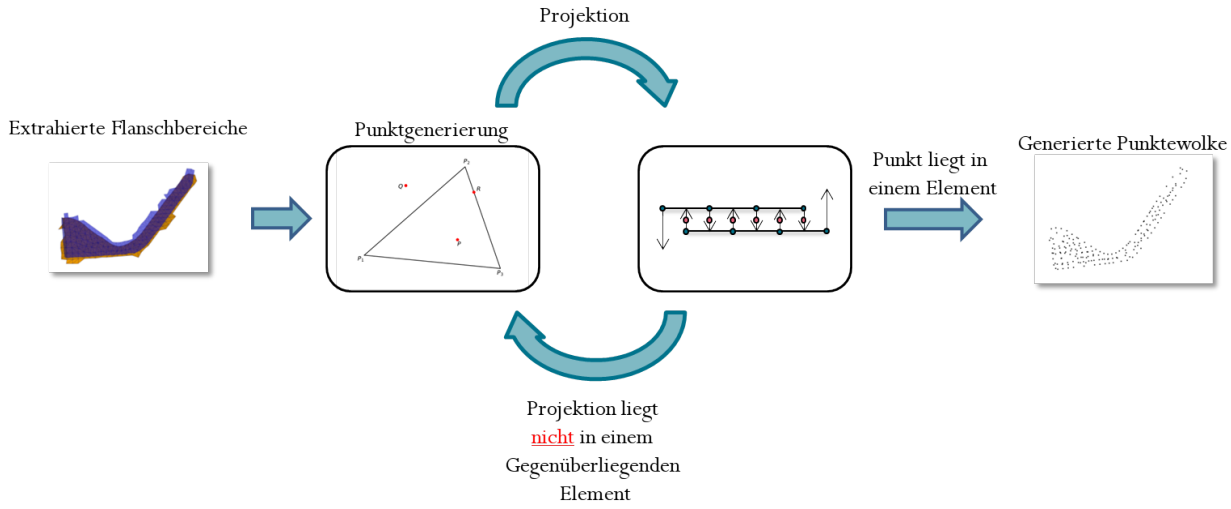


Abb. 43: Erzeugen von gleichverteilten Punkten

5.1.4 Bewertung von Strategie 1

Es wurde eine Methode dargelegt, Schweißpunkte durch bekannte Trainingsdaten auf neuen Flanschflächen über einen Klassifizierungs- und Mappingprozess zu platzieren. Der Nachteil von dieser Strategie ist, dass die Punktewolken aus den Altdatenständen immernoch in einer Datenbank für das Mapping der Schweißpunkte hinterlegt sein müssen. Das Verfahren stößt bei Flanschen, deren Geometrien noch nicht trainiert wurden und dessen Schweißpunktinformationen noch nicht in der Datenbank hinterlegt sind, auf ihre Grenzen. Der Klassifizierungsalgorithmus gibt zwar einen Wahrscheinlichkeitswert aus, mit dem die Geometrie einer anderen zuzuordnen ist, das Mapping der Schweißpunktdaten bringt bei großen Geometrieunterschieden falsche Verteilungen mit sich. In Strategie 2 ist ein Verfahren beschrieben, dass darauf ausgerichtet ist, ohne das Abspeichern von Trainingsdaten die Schweißpunkte auf Flanschgeometrien vorherzusagen.

5.2 Strategie 2

Diese Strategie benötigt durch das Trainieren von neuronalen Netzen mit allen relevanten Informationen keine in Datenbanken hinterlegten Trainingsdaten wie es der Fall in Strategie 1 ist. Die Trainingsdaten sollen in Form von Gewichtungen im neuronalen Netz gespeichert sein. Die Herangehensweise an diese Strategie ist in drei Schritte unterteilt.

- Abstrahieren der Flanschgeometrien
- Test durch einen Klassifizierungsalgorithmus, ob die Architektur des erstellten neuronalen Netzes in der Lage ist, die Abstraktion abzubilden
- Anwendung von Trainingsdaten aus Punktschweißverbindungen auf Flanschgeometrien

5.2.1 Erzeugen der Trainingsdaten

Um die Eingangsdaten so zu gestalten, dass die geometrischen Informationen sowie die Informationen über die Schweißpunktplatzierung für CNN's lesbar sind, ist eine Abstraktion notwendig. Aus den zuvor beschriebenen Prozessen liegen an dieser Stelle die Flanschgeometriedaten als Netz [5] vor.

Als Beispiel für das Abbild einer Steifigkeitsmatriz ist in FreeCAD eine einfache 2D - Geometrie erstellt und mit dem internen Meshing-Tool von FreeCAD vernetzt. Die diskretisierten Elemente sind in Abb. 44 dargestellt. Es handelt sich um ein 2D - Rechteck in der xy-Ebene. Für diese Geometrie soll im Folgenden die Steifigkeitsmatrix ermittelt werden.

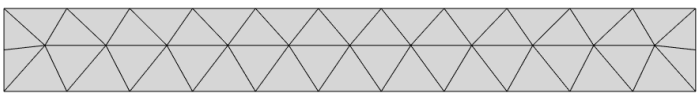


Abb. 44: Netz eines ebenen Rechtecks

Der Algorithmus zur Erstellung der Steifigkeitsmatrizen extrahiert zunächst die Verbindungen der Punkte, sodass aus dem Netz ein Fachwerk entsteht. Das Fachwerk ist in Abb. 45 dargestellt.

[5] Netz = Elemente und Kanten

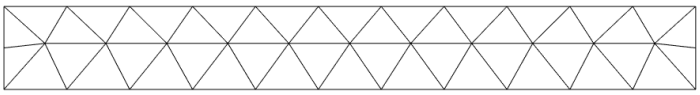


Abb. 45: Fachwerk eines ebenen Rechtecks

Für den Zweck der Abstraktion ist es nicht notwendig, die kompletten Steifigkeitsmatrizen zu errechnen. Es genügt, die Verbindungen der Stäbe und somit die geometrischen Randbedingungen zu berücksichtigen. Aus den Elementen des Fachwerks werden über den zuvor beschriebenen Algorithmus die Gesamtsteifigkeitsmatrix erstellt. Die Einzelsteifigkeitsmatrix ist dabei lediglich mit der Länge l des Stabes in mm durch die Elementmatrix $\begin{bmatrix} l & -l \\ -l & l \end{bmatrix}$ berechnet. Selbst die negativen Einträge können bei der Erstellung der modifizierten Steifigkeitsmatrix vernachlässigt werden, da die absoluten Werte der Matrixeinträge nicht für die Abbildung in neuronalen Netzen erforderlich ist.

Der Algorithmus erstellt eine Gesamtsteifigkeitsmatrix, die über die Python-Bibliothek Matplotlib[6] in RGB-Werte konvertiert wird. Das daraus entstehende Bild ist in Abb. 46 dargestellt. Die Abszisse sowie die Ordinate des Diagramms stellen die Knotennummern dar. Aus Gleichung (7) geht hervor, dass die Hauptdiagonale der Matrix mit den höchsten Werten besetzt ist, da die Einträge der Elementsteifigkeitsmatrizen in der Hauptdiagonalen addiert werden. Die Höhe dieser Werte gibt deshalb an, wie oft ein Knoten mit einem Stab verbunden ist. In den Nebendiagonalen sind die Verbindungen der Knoten nur einmalig hinterlegt, weshalb diese einen im Vergleich zur Hauptdiagonalen geringen Wert besitzen.

[6]Matplotlib ist eine Open-Source-Anwendungen für Python zur Darstellung von Graphen und Diagrammen

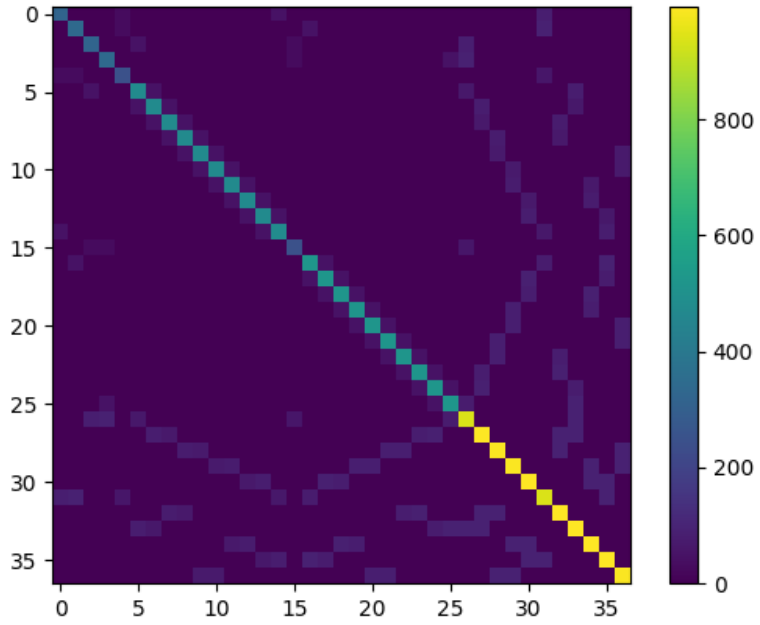


Abb. 46: Steifigkeitsmatrix des Beispielfachwerks

Das Problem bei diesem Vorgehen ist, dass jedes Meshing-Tool die Knoten bei der Definition der Elemente unterschiedlich nummeriert. Diese Nummerierung ist nicht genormt. Jedes Tool hat dazu einen eigenen Standard, die Punkte zu nummerieren. In Abb. 47 wird dies deutlich, da die Gesamtsteifigkeitsmatrix bei einer anderen Nummerierung der Knoten eine andere Gestalt annimmt. In der Abbildung der Gesamtsteifigkeitsmatrix ist zu sehen, dass die Knoten 26 - 36 am häufigsten mit Stäben verbunden sind. Folglich befinden sich diese Knoten in der Mitte des Rechtecks, die jeweils mit 6 Stäben verbunden sind. Entscheidet sich der Algorithmus des Meshing-Tools dafür, dort die Knotennummerierungen 1 - 11 zu verwenden, ändert sich die Gestalt der Steifigkeitsmatrix dementsprechend.

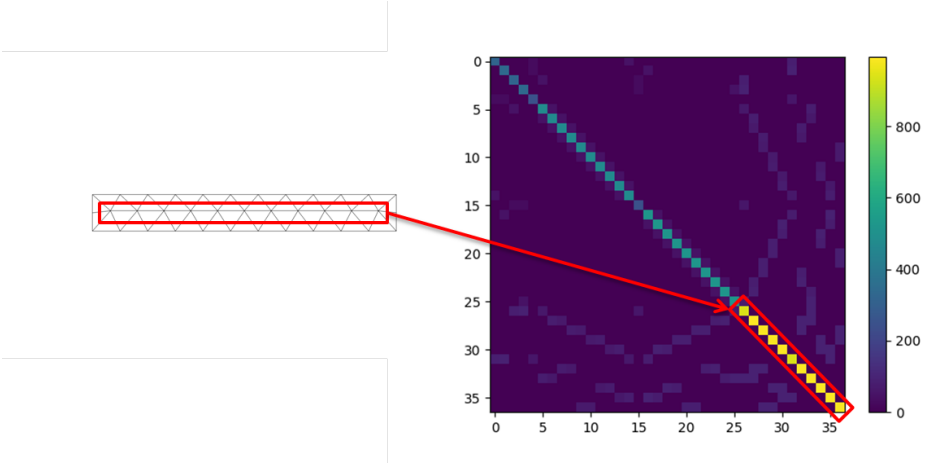


Abb. 47: Höhere Konnektivität

Die Abhängigkeit durch die Knotennummerierung wird anhand des folgenden Beispiels deutlich. In Abb. 48 sind Steifigkeitsmatrizen des selben Netzes mit unterschiedlichen Knotennummerierungen dargestellt. Die linke Darstellung stellt die Steifigkeitsmatrix einer fiktiven Geometrie dar. Die Nummerierung der Knoten wird durch einen zufällig veränderten Permutationsvektor dessen Länge gleich der Anzahl an vorhandenen Knoten ist. Die Neusortierung der Knotenanordnung ergibt eine veränderte Gestalt der Steifigkeitsmatrix, die in Abb. 48 auf der rechten Seite dargestellt ist.

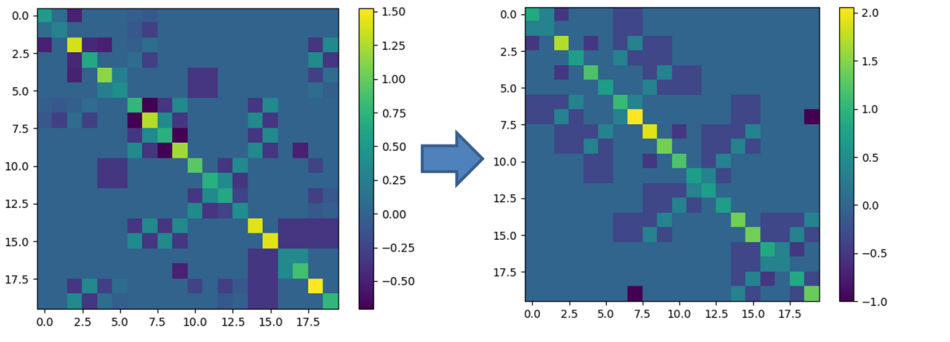


Abb. 48: Tausch der Knotennummerierung

Um die geometrischen Informationen der Nebendiagonalen für den Eingang in das neuronale Netz weiter in den Vordergrund zu heben, sind im Folgenden die Werte dieser um den Faktor 2 erhöht und aus Gründen von geringerem Speicherbedarf auf 1 normiert. Dazu wurde die Elementsteifigkeitsmatrix zu der in Gleichung (33) verändert und auf dieselbe Geometrie angewandt. Das Ergebnis dieser Änderung ist in Abb. 49 dargestellt. Es ist zu erkennen, dass die Nebendiagonalen im Verglichen zur Hauptdiagonalen in ihrem Wert gestiegen ist.

$$K = \frac{1}{l_{max}} \begin{bmatrix} l & 2l \\ 2l & l \end{bmatrix} \tag{33}$$

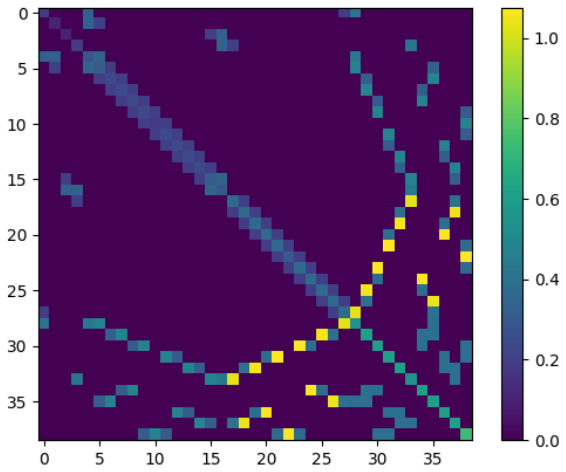


Abb. 49: Faktorisierte Steifigkeitsmatrix des Fachwerks

Durch die in Gleichung (33) dargestellte Matrix werden keine Steifigkeitsmatrizen sondern eine abgewandelte Form dieser berechnet. FÜr die dadurch ermittelten Matrizen wird deshalb im weiteren Verlauf der Arbeit der Begriff $modifizierteSteifigkeitsmatrizen$ verwendet. Das zu trainierende neuronale Netz muss Ähnlichkeiten zwischen den Trainingsdaten und den Testdaten erkennen. Dazu müssen die modifizierten Steifigkeitmatrizen in diesem Fall von dem Einfluss der Knotennummerierung entkoppelt werden. Für diesen Schritt ist ein einheitliches Vorgehen bei der Knotennummerierung notwendig.

5.2.2 Einheitliches Verfahren zur Knotennummerierung

Im Folgenden sind zwei Methoden beschrieben, die ein Einheitliches vorgehen bei der Knotennummerierung beschreiben.
Methode 1 zur Vereinheitlichung der Knotennummerierung:
Diese Methode wird direkt mit den Daten der Punktewolke angewandt und basiert auf der Theorie einer geometrischen Reihenfolge. Über die Methodik aus Abschnitt 5.1.1 kann zunächst der geometrische Schwerpunkt der Punktewolke ermittelt werden. Die Punktewolke kann dann anhand des Schwerpunktes in den Ursprung des Koordinatensystems transformiert werden. Dieser Schritt ist notwendig, um die Lage des jeweiligen Bauteils zu relativieren. Der nächste Schritt ist das Abtasten des Raumes mittels einer Kugel, die sich einen Ursprung mit dem der Punktewolke teilt. Der Algorithmus beginnt mit einem Radius von 0 mm und erhöht diesen stetig, bis ein Punkt der Punktewolke innerhalb der Kugel liegt. Dieser Punkt bekommt die erste Node-ID. Der Radius der Kugel wird solange erhöht, und die in die Kugel eintretenden Punkte der Reihe nach nummeriert, bis alle der Punkte nummeriert sind. In Abb. 50 ist diese Methode zum Sortieren der Knoten dargestellt.

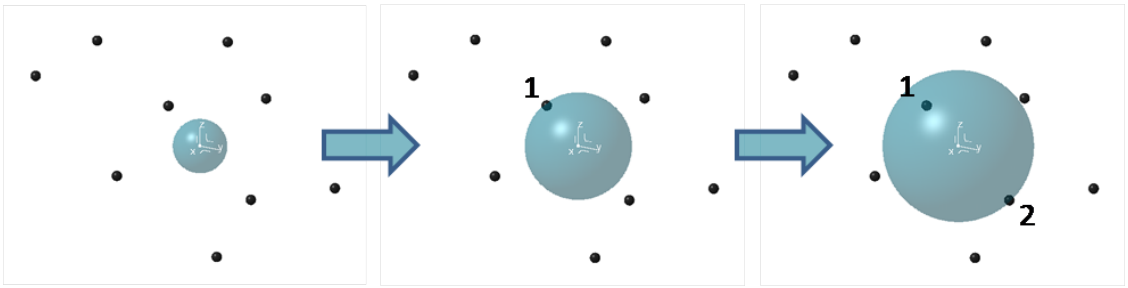


Abb. 50: Geometrische Knotennummerierung

Der Nachteil dieser Methode ist, dass gleiche Flanschgeometrien mit unterschiedlichen Netzen andere Knotennummerierungen aufweisen und somit zu unterschiedlichen Gestalten der führen können. Diese Methode ist somit nicht zur Vereinheitlichung der Knotennummerierung geeignet.

Methode 2 zur Vereinheitlichung der Knotennummerierung:
Diese Methode zur Vereinheitlichung der Knotennummerierung wird nach der Erstellung der modifizierten Steifigkeismatrix angewandt und sortiert die Zeilen und Spalten der Matrizen. Zur Übertragung der Matrizen eigenen sich Graphen. In der Graphentheorie ist ein Graph eine abstrakte Struktur, die eine Menge von Objekten und deren Verbindgungen beinhaltet. Ein Graph G ist ein geordnetes Paar (V, E) wobei V eine Menge von Knoten und $E \subseteq [V]^2$ eine Menge von Kanten ist. Kanten stellen Beziehungen zwischen Knoten dar. Grafen werden

zur Abbildung von Beziehungen eingesetzt.[26] Der Speicherbedarf von Grafen ist bei der Abbildung von dünn besetzten Matrizen geringer als der einer gespeicherten Gesamtmatrix. Ein Beispiel eines ungerichteten Graphen ist in Abb. 51 dargestellt.

- $V = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$
- $E = \{\ \{1,2\},\ \{2,3\},\ \{2,5\},\ \{2,6\},\ \{3,6\}\ \}$

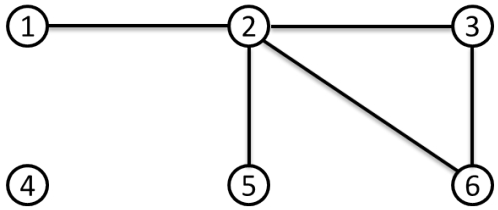


Abb. 51: Graph mit Knoten und Verbindungen

Für die Berechnung der Gleichungssysteme aus der FEM-Theorie ist es notwendig, Matrizen wie zum Beispiel die Steifgkeitsmatrix K oder auch die Massenmatrix M, zu invertieren. Das Invertieren ist der zeitintensivste Schritt bei der Lösung von Gleichungssystemen. Die Hintergründe dafür sind in [27] und [28] dargestellt. Die Effizienz für Rechenoperationen mit Matrizen hängt von der Bandbreite B der Matrix ab. Der Wert B ist durch k_l und k_r ermittelbar. Dabei sei eine Matrix A mit den Einträgen α_{ij} für $j > i+k_l$ und $i > j+k_r$ gegeben. Die linksseitige Bandbreite sei k_l und die rechtsseitige Bandbreite sei k_r. Die Bandbreite einer Matrix kann dann über Gleichung (34) ermittelt werden. Für symmetrische Matrizen gilt immer $k_l = k_r$.

$$B = k_l + k_r + 1 \tag{34}$$

Durch ein Tauschen der Anordnungen der Gleichungen im Gleichungssystem kann die Bandbreite beeinflusst werden. Im Falle von den modifizierten Steifigkeitsmatrizen bedeutet das Reduzieren von B ein Tauschen der Knotennummerierung. Für jede quadratisch symmetrische Matrix gibt es eine Permutation in Form eines Permutationsvektors p, die die Matrix in eine Matrix minimaler Bandbreite überführt. Dabei beinhaltet der Permutationsvektor die

Reihenfolge in natürlichen Zahlen, in der die Knoten angeordnet sind. In der Regel ist die Permutation nicht eindeutig ermittelbar, weshalb die meisten Algorithmen eine Näherung daran anstreben[29]. Approximative Algorithmen zur Bestimmung dieses Permutationsvektors sind zum Beispiel der Algorithmus nach Cuthill-McKee (CM) 1969, der Reverse Cuthill-McKnee (RCM) [30] sowie der Algorithmus von Gibbs-Poole-Stockmeyer (GPS) 1976 [30]. Aufgrund des approximativen Verfahrens der Algorithmen ist ein Finden der gleichen Bandbreitenmatrix für ähnliche Flanschgeometrien ohne Toleranz nicht immer möglich.

Der CM-Algorithmus basiert auf der Umnummerierung der Knotennummern einer Laplace-Matrix L, die Beziehungen von Knoten und Kanten eines Graphen, der alle Abhängigkeiten des Netzes beinhaltet, repräsentiert. Die Laplace-Matrix L eines Graphen mit der Knotenmenge K ist eine $(K \times K)$ - Matrix und ist definiert durch $L = D - A$. Wobei D die Gradmatrix und A die Adjazenzmatrix des Graphen darstellt.
Der CM-Algorithmus führt folgende Schritte aus:

- Wählen eines Startknotens i mit minimalem Grad aus D. Bei mehreren Knoten mit minimalem Grad wird der Startknoten per Zufall aus der Menge der Knoten mit minimalem Grad gewählt. Der Startknoten wird in eine temporäre Ebenentabelle E in die erste Spalte E_1 eingetragen.
- Alle mit dem Startknoten verknüpften Knoten werden ihrem Grad nach aufsteigend in E_2 eingetragen. Knoten werden nur dann eingetragen, wenn sie noch nicht in E vorkommen
- In Spalte E_3 werden die Knoten eingetragen, die mit dem Knoten minimalen Grades aus E_2 verknüpft sind. Bei mehreren Knoten mit minimalem Grad wird einer dieser Knoten zufällig ausgewählt.
- Schritt 2 und 3 wiederholen sich so oft, bis alle Knoten in E eingetragen sind.
- Der Permutationsvektor besteht jetzt aus den Einträgen der Ebenentabelle aneinandergereiht.
- Besitzen zu Beginn mehrere Knoten den minimalen Grad, kann eine Iteration des beschriebenen Verfahrens herangezogen werden. Damit dieses Verfahren als Normierung dienen kann, muss garantiert werden, dass ein Minimum der Bandbreite gefunden wird. Dazu ist ein Optimierungsverfahren unerlässlich.

Eine Vereinheitlichung der Knotennummerierung ist nur dann gegeben, wenn das absolute Minimum der Bandbreite B durch einen Optimierungsalgorithmus für jede Matrix gefunden wird.
In Abb. 52 ist das Ergebnis des CM-Algorithmus auf das in Abb. 45 dargestellte Fachwerk zu sehen. Die Bandbreite der modifizierten Steifigkeitsmatrix, die in Abb. 52 (links) dargestellt ist, konnte von $B = 37$ auf eine Bandbreite von $B = 14$ in Abb. 52 (rechts) reduziert werden. Zum Vergleich ist die nicht reduzierte modifizierte Steifigkeitsmatrix in Abb. 52(rechts) schwach im Hintergrund dargestellt.

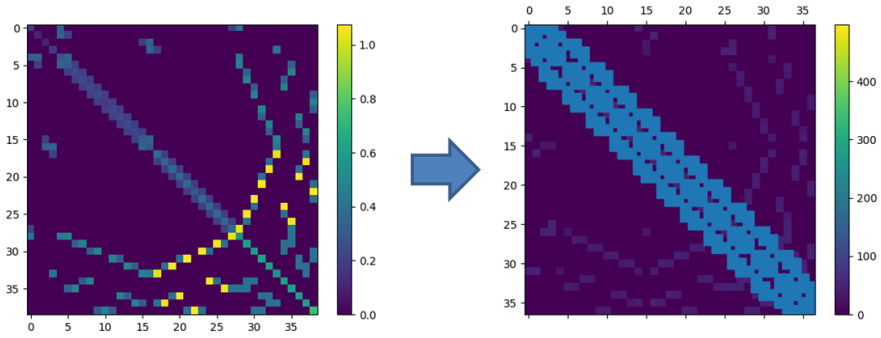


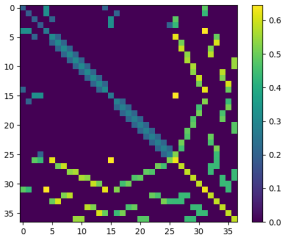
Abb. 52: Bandbreitenreduzierte Steifigkeitsmatrix

Nach diesem Schritt sind die modifizierten Steifigkeitsmatrizen in bandbreitenreduzierte Matrizen überführt und werden im weiteren Verlauf der Arbeit als solche bezeichnet.

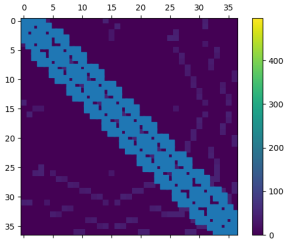
5.2.3 Klassifizierungs-Algorithmus

Im Folgenden soll anhand eines Klassifizierungsproblems ermittelt werden, ob CNN's in der Lage sind, die banbreitenreduzierten Matrizen abzubilden. Das Klassifizierungsproblem ist anhand von 4 unterschiedlichen Geometrien dargestellt. Dabei definiert jede der 4 Geometrien jeweils eine Klasse. Innerhalb der Klasse ist die Geometrie jeweils mit einer Netzgröße von 10 mm, 15 mm, 20 mm, 25 mm und 30 mm hinterlegt.
Die Geometrien sind in FreeCAD konstruiert und nach der in Abschnitt 5.2.1 beschriebenen Methode zu bandbreitenreduzierten Matrizen überführt. In Abb. 53 ist jeweils eine Geometrie, die modifizierte Steifigkeitsmatrix mit der Netzgröße 20 mm und ihre bandbreitenreduzierten Matrizen von jeder Klasse dargestellt.

(a) Geometrie: Klasse-1

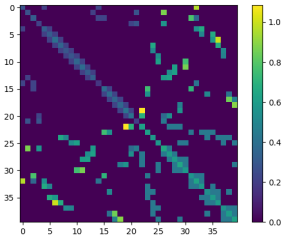


(b) mod. Steifigkeitsmatrix

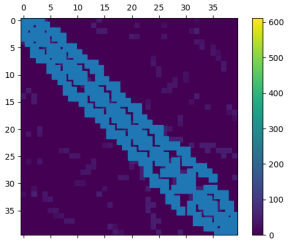


(c) Reduzierte-Bandmatrix

(d) Geometrie: Klasse-2

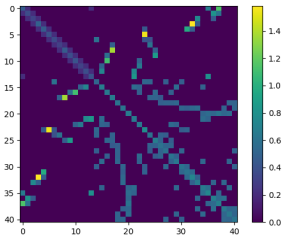


(e) mod. Steifigkeitsmatrix

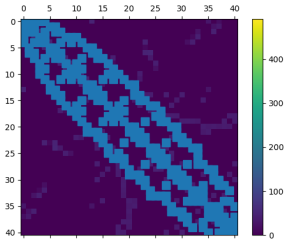


(f) Reduzierte-Bandmatrix

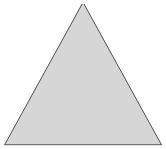
(g) Geometrie: Klasse-3



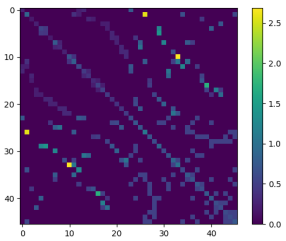
(h) mod. Steifigkeitsmatrix



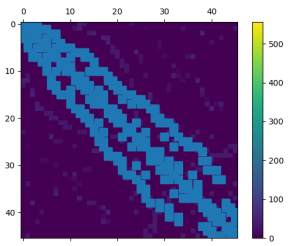
(i) Reduzierte-Bandmatrix



(j) Geometrie: Klasse-4



(k) mod. Steifigkeitsmatrix



(l) Reduzierte-Bandmatrix

Abb. 53: Klassifizierungsdaten

Die modifizierten Steifigkeismatrizen und somit auch die bandbreitenreduzierten Matrizen besitzen eine unterschiedliche Anzahl an Zeilen, da die Zeilenanzahl der Matrix äquivalent zu der Anzahl der Knoten der vernetzten Geometrie ist. Ein neuronales Netz kann nur eine fixe Anzahl an Eingangs- und Ausgangsneuronen besitzen, die am Anfang definiert ist. Das führt zu einer einheitlichen Größe der Matrizen. Um einen Informationsverlust zu vermeiden, sind alle Matrizen auf die Größe der größten Matriz erweitert. Die Erweiterung ist durch das virtuelle Hinzufügen von Knoten ohne Verbindungen untereinander an den Graph der bandbreitenreduzierten Matrizen erfolgt. Dies ist beispielhaft an einer fiktiven bandreitenreduzierten Matrix in Abb. 54 dargestellt. Der rotumrandete Bereich stellt die Erweiterung des Graphen um die nichtverbundenen Knoten dar.

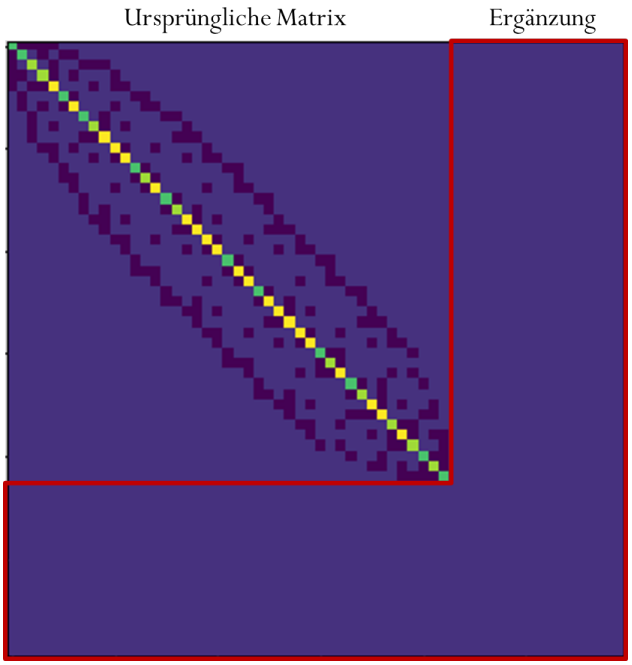


Abb. 54: Erweitern der Matrizen auf eine Einheitsgröße

Der Nachteil der Matrixerweiterung ist die Tatsache, dass die Matrizengröße und somit der Speicherbedarf steigt. Die Matrixerweiterung ist deshalb erst direkt vor dem Einlesen in die Eingangsebene des neuronalen Netzes durchzuführen. Tensorflow[7] stellt hierfür eine *Zeropadding* - Anwendung bereit, die unmittelbar vor der Eingangsebene des neuronalen Netzes platziert werden kann.
Für den Aufbau und das Training des neuronalen Netzes wurde der Framework *Keras*2.3.0 von Tensorflow 1.14 verwendet. Die Trainingsdaten setzen sich aus den erweiterten Matrizen der Geometrieklassen mit den Netzgrößen 10 mm, 15 mm, 25 mm und 30 mm zusammen. Die

[7]Tensorflow ist eine Open-Source-Software für die Anwendungen von maschinellem Lernalgorithmen

erweiterten Matrizen der Geometrieklassen mit der Netzgröße 20 mm stellen die Testdaten dar. Die Trainings- und Testdaten sind in Form eines 4D-Tensors in die Eingangsebene des CNN's zu übergeben. Die Trainingsdaten werden durch einen 4D-Tensor der Form (batch, width, height,channels) abgebildet.

- batch = Rang des Tensors
- width = Breite der Matrix
- height = Höhe der Matrix
- channels = Dimension der Matrixeinträge

Der Trainingsdatentensor hat die Form (16, 238, 238, 1), während der Testdatentensor die Form (4, 238, 238, 1) besitzt. Als Basis für das verwendete CNN wurde ein kommerzielles CNN von Keras verwendet, das in [31] beschrieben ist.
Für die Architektur des neuronalen Netzes wurde ein kommerzielles CNN's von Keras zur Klassifizierung von handgeschriebenen Zahlen als Basis verwendet und modifiziert. Die Modifikation beinhaltet das Einfügen eines MaxPooling2D - Layer nach dem ersten Convolutional2D - Layer, um die zweite und dritte Dimension der Datentensoren zu verringern. Außerdem sind die Filtergrößen der beiden Convolutional2D - Layer auf die Größe (2×2) verringert. Die verwendete Architektur ist in **??** dargestellt.

Für die Layer $conv2d_1$, $conv2d_2$, $dense_1$ und $dense_2$ wurde die Aktivierungsfunktion ReLu mit $y = 0$ für $0 \leq x$ und $y = x$ für $0 > x$ für die Klassifizierung verwendet. Die Trainingsmatrizen besitzen keine negativen Einträge, weshalb in diesem Fall der Ausgangswert des einen Neurons gleich dem Eingangswert des nächsten Neurons ist $y = x$. Dem Layer $Out - Layer$ ist die Softmax-Aktivierungsfunktion zugeordnet, die in Gleichung (14) dargestellt wurde, da sich diese Funktion für Klassifizierungsprobleme eignet, siehe Abschnitt 2.3.1. Der Trainingslauf wurde mit folgenden Parametereinstellungen durchlaufen:

- Trainings Eingangstensor = (16, 238, 238, 1)
- Test Eingangstensor = (4, 238, 238, 1)
- batch size = 2
- epochs = 60

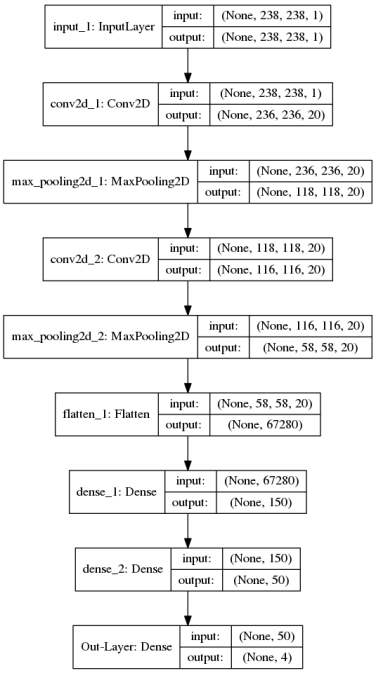
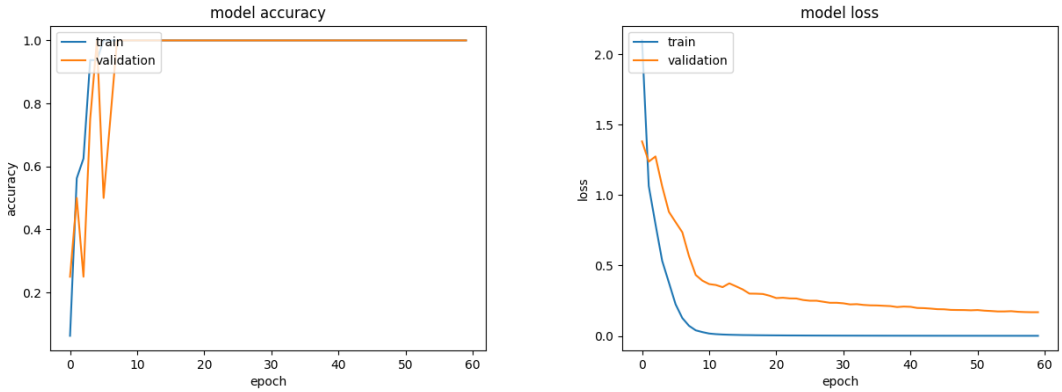


Abb. 55: Architektur des Klassifikationsnetzes

- optimizer = adam
- aktivation 'relu' = $conv2d\_1$, $conv2d\_2$, $dense\_1$ und $dense\_2$
- aktivation 'softmax' = $Out - Layer$

Das Ergebnis des Trainings und der Testläufe ist in Abb. 56 dargestellt.

Die Trainings- und Testgenauigkeit liegt für diesen Versuch der Abbildung von bandbreitenreduzierten Matrizen über neuronale Netze bei 100%. Das bedeutet, das neuronale Netz mit der beschriebenen Architektur ist in der Lage die Trainingsdaten abzubilden und noch nicht trainierte Daten zu Klassifizieren. Aus dem Versuch geht ebenfalls hervor, dass das neuronale Netz in der Lage ist, innerhalb dieses Versuchs Affinitäten in den modifizierten Steifigkeitsmatrizen bei unterschiedlichen Vernetzungsgrößen zu detektieren und die 4 Testdaten richtig zu klassifizieren.



(a) Trainings- und Testgenauigkeit

(b) Trainings- und Testverlust

Abb. 56: Genauigkeit und Verslust des Trainings

5.2.4 Schweißpunkvorhersage

Im Folgenden soll eine Methode beschrieben werden, mit der es möglich ist, Schweißpunktplazierungen auf Flanschgeometrien für neuronale Netze trainierbar zu abstrahieren, sodass Schweißpunktplatzierungen auf Flanschgeometrien automatisiert vorhergesagt werden können. Dabei sind die Informationen der Platzierung in Form von den Gewichtungen des neuronalen Netzes hinterlegt.

Das neuronale Netz soll die Schweißpunktplatzierung über seinen Ausgabevektor bestimmen. Für das Trainieren und später Interpretieren des Ausgabevektors, müssen die Schweißpunktverteilungen abstrahiert auf die Flanschgeometrie abgebildet werden. Da die bandbreitenreduzierten Matrizen die einzige Verknüpfung zur Geometrie über die Knoten besitzt, muss sich auch bei der Erstellung der Ausgabevektoren auf die gleichen Knoten bezogen werden. Da Schweißpunkte auf Flanschgeometrien platziert werden müssen, die noch nicht explizit trainiert wurden, ist es notwendig, größere Bereiche zu definieren, innerhalb denen ein Schweißpunkt platziert werden kann. Dazu wird jedem Knoten eine Wahrscheinlichkeit zugewiesen, mit der ein Schweißpunkt auf diesem Knoten liegt. Mit größer werdendem Abstand vom Schweißpunkt, wird die Wahrscheinlichkeit kleiner. Die Wahrscheinlichkeitsverteilung auf die Knoten darum herum ist über eine Standardabweichung definiert, die in Gleichung (35) dargestellt ist. Die Wahrscheinlichkeit an Knoten x ist durch $\rho(x)$ definiert. Der Wert x_0 gibt den Scheitelpunkt der Gaußglocke und σ die Streuung der Wahrscheinlichkeitsdichte um den Scheitelpunkt an.

$$\rho(x) = e^{-\frac{1}{2}\cdot\frac{(x-x_0)^2}{\sigma^2}} \tag{35}$$

Die Wahrscheinlichkeitswerte für jeden Knoten sind in einem Vektor V_w mit der Länge n dargestellt, wobei n gleich der Anzahl der Knoten ist. Dabei wird der Wert der höchsten Wahrscheinlichkeit eingetragen, wenn ein Knoten im Einflussbereich von zwei Knoten liegt. Die Parameter der Gleichung (35) sind $x_0 = 0$ und $\sigma = 6$ Dieser Vektor mit den Wahrscheinlichkeiten ist der Ausgangsvektor, auf den das neuronale Netz referenziert werden soll. Zur Verdeutlichung des beschriebenen Verfahrens zur Erstellung der Wahrscheinlichkeitsvektoren werden die Flanschnetze mit den Wahrscheinlichkeitsvektoren in dem Programm ParaView[8] dargestellt. Im CAE-Bereich ist eine Visualisierung der Ergebnisdaten im Postprozess ein notwendiges Instrument zur Analyse von Berechnungen. Dabei werden die Geometrien in sich unterscheidende Farben eingeteilt, um zum Beispiel Kraftverteilungen zu visualisieren.

[8]ParaView ist eine Open-Source-Software um abstrakte Daten und Zusammenhänge in eine graphische bzw. visuell erfassbare Form zu bringen [32]

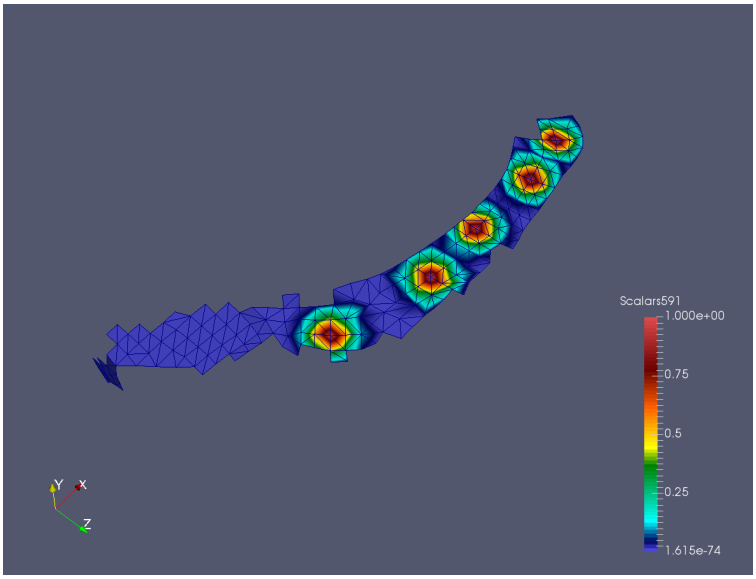


Abb. 57: Schweißpunktwahrscheinlichkeiten über eine Flanschgeometrie

Für die Übermittlung dieser Daten gibt es Austauschformate wie zum Beispiel das VTK (Visualization ToolKit)-Format. VTK ist eine Open-Source Software für die Übermittlung von 3D-Grafiken und besitzt einen Python-Interpreter, weshalb das Format über eine direkte VTK-Applikationen in Python verwendet werden kann. Über die Klasse *vtkData* können der Datei direkt die Geometrie darstellenden Elemente übermittelt werden, die von der Applikation selbstständig in NID's und EID's umgewandelt und abgespeichert werden. Den Knoten kann dann über die Funktion *PointData* ein Vektor dessen Länge gleich der Anzahl an Knoten ist zugewiesen werden. Dieser Vektor enthält die Eigenschaften der Knoten nach der Berechnung wie zum Beispiel einen Kraftwerte oder Verschiebungen. Die Eigenschaften der Knoten für die Visualisierung der Schweißpunktwahrscheinlichkeit ist V_w. In **??** ist eine Flanschgeometrie mit den Schweißpunktwahrscheinlichkeiten dargestellt. Die roten Bereiche deuten auf eine Wahrscheinlichkeit von 100 % für einen Schweißpunkt hin, die dann mit den Farben in der Skala abnimmt.

Für das Training der neuronalen Netzen mit den Flanschgeometrien sind von allen Flanschen des Toyota Yaris die bandbreitenreduzierten Matrizen aufgestellt. Es werden aus Gründen der Rechenzeit nur Flanschgeometrien mit weniger als 1.000 Knoten verwendet. Alle bandbreitenreduzierten Matrizen sind deshalb mit nicht verbundenen Knoten auf die Größe (1.000×1.000) erweitert. Damit die Knotennummerierung der bandbreitenreduzierten Matrizen mit den Vektoren V_w übereinstimmen, müssen die Knoten der Vektoren V_w durch den selben Permutationsvektor sortiert werden, der durch den CM-Algorithmus ermittelt wurde. Auch die Vektoren V_w müssen auf die selbe Größe wie die bandbreitenreduzierten Matrizen gebracht werden, in dem die Länge der Vektoren durch das Auffüllen mit Nullen auf 1.000 erweitert wird.
Das neuronale Netz für diesen Versuch ist in Abb. 58 dargestellt.

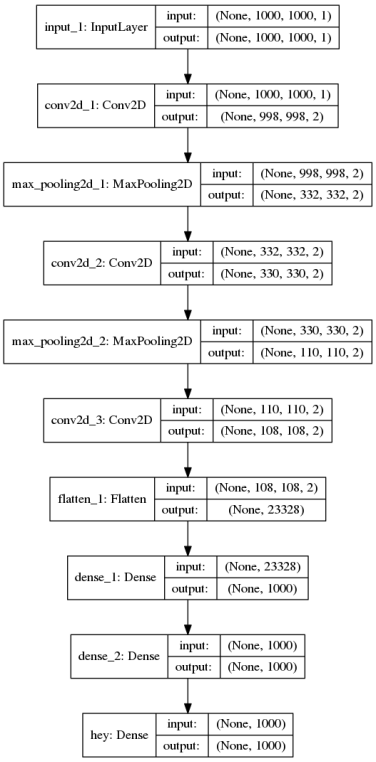


Abb. 58: Architektur des Schweißpunktvorhersage Netzes

Für das Training wurden die folgenden Parameter verwendet:

- Trainings und Test Eingangstensor = (800, 1000, 1000, 1)
- Trainings und Test Ausgangstensor = (800, 1000, 1)
- batch size = 200
- epochs = 100
- optimizer = adam
- aktivation 'relu' = $first\_layer\_conv$, $second\_layer\_conv$, $third\_layer\_conv$, $dense\_51$ und $dense\_52$
- aktivation 'relu' = out

Damit die Ausgabevektoren des neuronalen Netzes wieder den Geometrien zugewiesen werden können, müssen die Knoten durch denselben Permutationsvektor wieder der ursprünglichen Reihenfolge nach sortiert werden. In **??** ist ein Schaubild dargestellt, in dem das Vorgehen der Knotensortierung beschrieben ist.

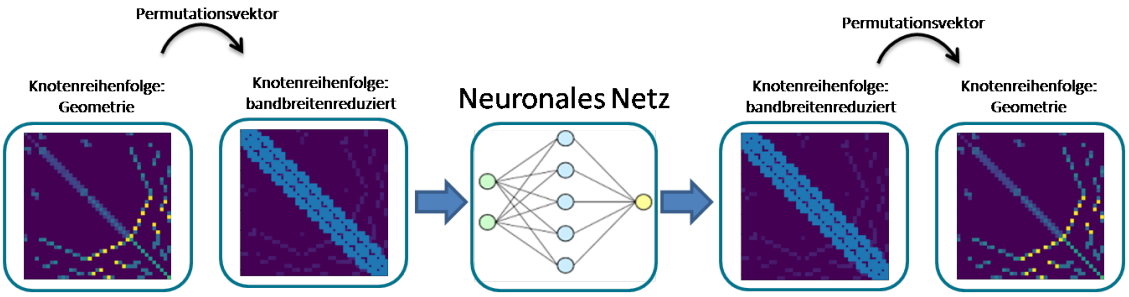
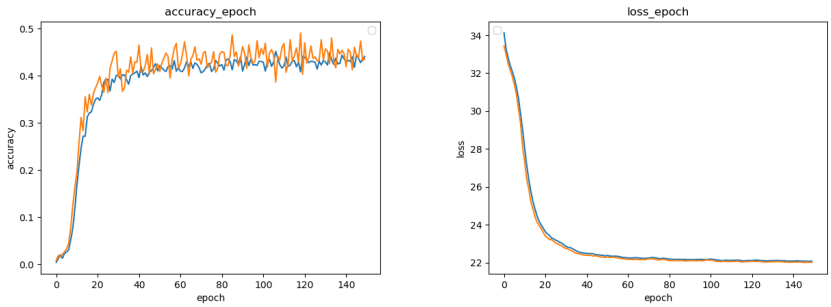


Abb. 59: Mapping der Knotennummerierung

Die Trainingsgenauigkeit und der Verlust beim Training des neuronalen Netzes ist in Abb. 60 dargestellt.

Dabei wurde eine Trainingsgenauigkeit von 44 % ermittelt, woraus sich schließen lässt, dass die in **??** beschriebene Architektur eines neuronalen Netzes die Schweißpunktwahrscheinlichkeiten nur zu 44 % abbilden kann. Die Folgerung daraus ist, dass das beschriebene Netz mit den abstrahierten Daten nicht in der Lage ist, die Informationen über die Schweißpunktwahrscheinlichkeiten zuverlässig wiederzugeben. Teilinformationen wie zum Beispiel



(a) Trainings- und Testgenauigkeit (b) Trainings- und Testverlust

Abb. 60: Genauigkeit und Verslust des Trainings

einzelne Bereiche, in denen die Schweißpunkte bei den Trainingsdaten platziert wurden, konnten jedoch wiedergegeben werden. In **??** ist der Ausgangsvektor des neuronalen Netzes mit der Wahrscheinlichkeitsverteilung über den Permutationsvektor zurück auf die Geometrie projiziert abgebildet. Durch den in Abschnitt 4.1 beschriebenen Point-Picking-Algorithmus können in den ermittelten Bereichen höherer Wahrscheinlichkeit die Schweißpunkte verteilt werden.

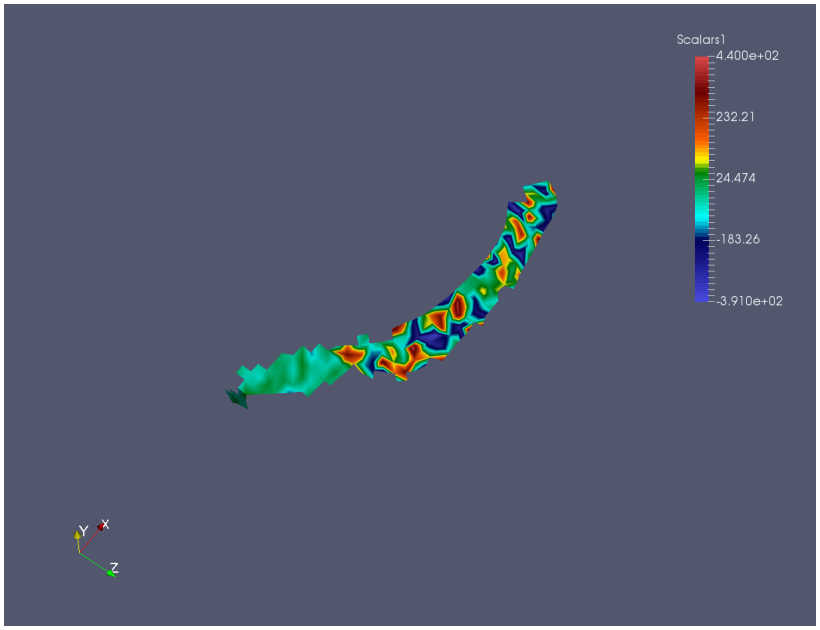


Abb. 61: Mapping des Ausgangsvektors

Um den Rechenaufwand des Trainings für das neuronale Netz zu verkürzen, können die Matrizen

6 Eingliedern der Schweißpunktvorhersage in den Preprozess

In Abb. 62 ist die Eingliederung eines Expertensystems zur automatisierten Schweißpunkterzeugung in CAX-Preprozessen schematisch dargelegt.

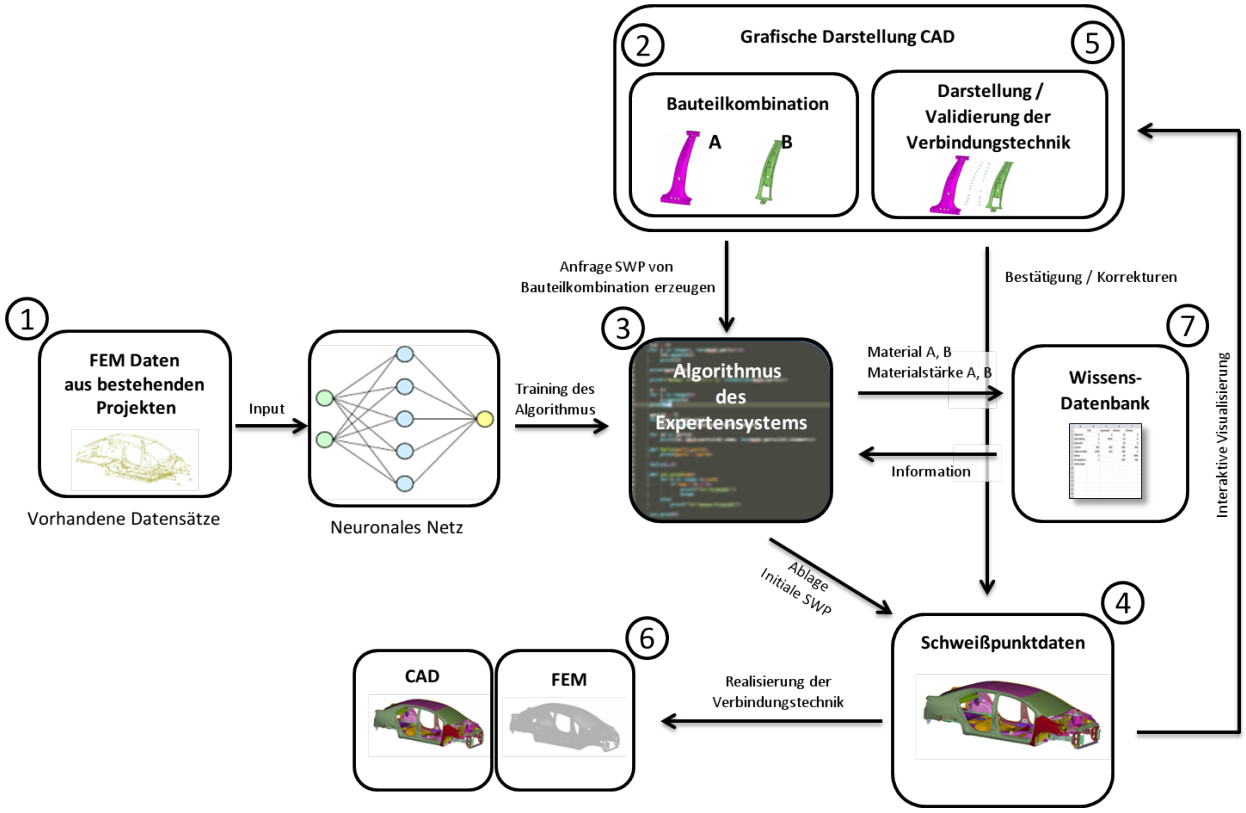


Abb. 62: Einbinden der Strategie in den CAX-Preprozess

Schritt 1: (In Abb. 62 Prozess von 1 - 3)

Der Algorithmus des Expertensystems muss zunächst mit existierenden Flanschgeometrien und den jeweiligen Schweißpunktverbindungen aus vorhandenen Datensätzen mit Produktonsfreigabe trainiert werden. Dabei ist die Form der Trainingsdaten, je nach Strategie des Expertensystems unterschiedlich und müssen dementsprechend bearbeitet oder abstrahiert werden. In dieser Arbeit wurden dazu unter Abschnitt 5 zwei mögliche Strategien dargestellt.

Schritt 2: (In Abb. 62 Prozess von 2 - 3 - 4)

Der Anwender gibt eine aus zwei Bauteilen A und B bestehende Bauteilkombination an, die

automatisiert durch Schweißpunkte verbunden werden soll. Diese Problemstellung wird an den Algorithmus des Expertensystems weitergegeben. Auf Basis der trainierten Daten ermittelt der Algorithmus die Schweißpunktverteilung und gibt diese an eine Schweißpunktdatenbank weiter.

Schritt 3: (In Abb. 62 Prozess von 3 - 7)

Bei einer vorhandenen Wissensdatenbank, die definierte Regeln zur Schweißpunktverteilung bei gegebenen Materialkombinationen oder Materialstärken besitzt, kann diese Information nach einer Abfrage durch den Algorithmus mit in den Schweißpunktplazierungsprozess eingehen.

Schritt 4: (In Abb. 62 Prozess von 4 - 5)

Die Schweißpunktdatenbank gibt zunächst eine interaktive Visualisierung der ermittelten Schweißpunktplatzierungen an den Anwender in Form einer grafischen Darstellung im CAD zurück. Der Anwender hat so die Möglichkeit die Schweißpunktverteilung visuell oder mit vom CAD-Programm bereitgestellten Tools zu prüfen. Beispielsweise zur Prüfung der Randabstände oder der Abstände zueinander.

Schritt 5: (In Abb. 62 Prozess von 5 - 4)

Nach der Prüfung der Schweißpunktverteilung im CAD-Programm kann der Anwender die Verteilung bestätigen oder diese durch ein Versetzung der Schweißpunkte korrigieren. Anschließend werden die Validierten Schweißpunktdaten an die Schweißpunktdatenbank zurückgegeben.

Schritt 6: (In Abb. 62 Prozess von 4 - 6)

Die Schweißpunktdaten werden dann in allen vorgesehenen CAX-Anwendungen realisiert.

7 Zusammenfassung und Ausblick

Im Verlauf dieser Arbeit wurden drei Methoden entwickelt, um Schweißpunkte im Preprozess von CAX-Anwendungen automatisiert auf Karosseriebauteilen zu platzieren.
Zur Vorbereitung der Methodenentwicklung wurde ein Algorithmus erarbeitet um Flanschbereiche und die zugehörigen Schweißpunkte aus vorhandenen diskretisierten Karosseriebauteilen zu extrahieren. Für die Extraktion wurden Kriterien definiert, nach denen Elemente der FE-Netze einem Flansch zugeordnet werden können. Als Beispiel dafür ist dieser Algorithmus an den LS-Dyna-Inputdecks des Toyota Yaris erprobt worden und die Ergebnisse in Form von CAD-Daten dargestellt.
Im Anschluss daran ist eine teilautomatisierte Generierung von Schweißpunkten in CAX-Anwendungen entwickelt worden. Der entwickelte Algorithmus ist in der Lage, eine manuell definierte Anzahl an Schweißpunkten mit möglichst gleichgroßen Abständen auf der Flanschfläche zu verteilen. Die Grundlagen hierzu sind die extrahierten Flanschgeometrien in Form von Punktewolken. Für die Verteilung der Punkte wurde ein Point-Picking-Algorithmus verwendet, der Punkte nach gesetzten Randbedingungen wie zum Beispiel der Abstand zu den Flanschrändern aus der Punktewolke auswählt. An den Positionen der ausgewählten Punkte sind dann in FreeCAD die Schweißpunktverbindungen durch normal zur Fläche ausgerichtete Zylinder visualisiert.
Des Weiteren sind zwei Strategien entwickelt worden, bei denen die Schweißpunktvorhersage durch maschinelle Lernalgorithmen ermöglicht wird. Strategie 1 basiert auf der Klassifizierung von Geometrien und einem anschließenden Mapping der Schweißpunkte. Dazu wurden verschiedene Klassifizierungsmethoden dargelegt. In einer Vorarbeit ist ein Klassifizierungsalgorithmus entstanden, der durch das Trainieren von neuronalen Netzen in der Lage ist, Punktewolken zu klassifizieren. Die Eingangsdaten für den Klassifizierungsalgorithmus sind Punktewolken mit einer definierten Anzahl an Punkten. Deshalb wurde ein Algorithmus dargestellt der eine definierte Anzahl an Punkten auf den extrahierten Flanschgeometrien verteilen kann. Durch hinterlegte Informationen über die Schweißpunktpositionierung auf der alten Flanschgeometrie, können nach der Klassifizierung die Schweißpunkte auf die neue Flanschgeometrie gemappt werden. Das Mapping basiert auf einer Reihe von Matrizenmultiplikationen, die ein Best-Fit von Punktewolken über eine Translation, eine Skalierung und über Rotationen suchen.
Strategie 2 basiert auf der Überlegung, die Schweißpunktinformationen über ein neuronales Netz auf die Geometrie zu projizieren. Dazu ist ein Algorithmus beschrieben, der die extrahierten Flanschgeometrien in Form von modifizierten Steifigkeitsmatrizen für CNN's einlesbar als Trainingseingangsdaten abstrahiert. Da Steifigkeitsmatrizen allgemein von der Knotennummerierung abhängen, wurde ein Verfahren zur Vereinheitlichung der Knotennummerierung entwickelt. Diese vereinheitlichten Steifigkeitsmatrizen sollen über das neuronale

Netz auf einen Vektor gemappt werden, der die Wahrscheinlichkeiten eines Schweißpunktes an den jeweiligen Knoten beinhaltet. Das Training des neuronalen Netzes mit allen extrahierten Flanschen des Toyota Yaris hat eine Trainingsgenauigkeit von 44 % erreicht, was für eine genaue Bestimmung der Schweißpunktlage nicht ausreichend ist. Es konnten lediglich Teilbereiche identifiziert werden, auf denen sich die Schweißpunkte befinden.

Ausblick:
Die Trainingsgenauigkeit des neuronalen Netzes der Strategie 2 von 44 % ist für eine genaue Vorhersage der Schweißpunktplatzierungen nicht ausreichend. Um die Genauigkeit der Schweißpunktvorhersage durch Strategie 2 zu erhöhen, können die Eingangs- und Ausgangsdaten des Trainings hinsichtlich ihres Informationsgehaltes über die Geometrie oder Eigenschaften des Bauteils erweitert werden. Dazu können die Steifigkeitsmatrizen zum Beispiel durch die Steifigkeiten der Scheibenelemente beschrieben werden.
Des Weiteren können die Variablen der Steifigkeitsmatrix E und A aus Gleichung (9) verwendet werden, um Blechdicken oder Blechmaterialien der Fahrzeugteile in den Steifigkeitsmatrizen zu berücksichtigen. Diese Implementierung gibt dem Algorithmus die Möglichkeit, geometrisch ähnliche Flanschgeometrien anhand von weiteren Kriterien zu unterscheiden.
Die Qualität eines neuronalen Netzes steigt mit der Anzahl an Tainingsdatensätzen. Für ein neuronales Netz, das die richtigen Ausgangsvektoren für alle existierenden Flanschgeometrien findet, benötigt das Netz viele Datensätze. Dazu können noch weitere Flanschdaten von Fahrzeugmodellen mit dem in dieser Arbeit beschriebenen Algorithmus extrahiert und das neuronale Netz damit trainiert werden. Um ein Overfitting zu vermeiden, sollten die Daten vor dem Training geprüft werden, damit nicht zu viele gleiche Flanschgeometrien trainiert werden.

8 Literaturverzeichnis

Literatur

[1] Prof. Dr.-Ing. Martin Meywerk: CAE-Methoden in der Fahrzeugtechnik. Springer-Verlag, [Veröffentlicht 2007]

[2] U. Seiffert, Gotthard Rainer (Hrsg.): Virtuelle Produktentstehung für Fahrzeug in Antrieb im Kfz. Vieweg+Teubner, [Veröffentlicht 2008]

[3] 6. LS-DYNA Anwenderforum, Methoden und Prozesse zur Kostensenkung - Ein Status der Wandlungen im Fahrzeugentwicklungsprozess durch CAE-Methoden, Frankenthal 2007.

[4] Daimler AG [online], Available: https://media.daimler.com/marsMediaSite/de/instance/ko/CAx-goes-AIAx-Kuenstliche-Intelligenz-lernt-von-Experten.xhtml?oid=41324165, [Zugriff am 02.08.2019]

[5] LS-DYNA®, KEYWORD USER'S MANUAL Volume I [online], Available: https://www.dynasupport.com/manuals/ls-dyna-manuals/ls-dyna_manual_volume_i_r11.pdf, Californien, [Veröffentlicht am 18.10.2018]

[6] D. Kriesel, Neuronale Netze [online], Available: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-1col-dkrieselcom.pdf, [Zugriff am 23.08.2019]

[7] P. Balzer, Neuronale Netze einfach erklärt [online], Available: https://www.cbcity.de/tutorial-neuronale-netze-einfach-erklaert, Dresden, [Zugriff am 15.08.2019]

[8] K. Backhaus, B. Erichson, R. Weiber: Fortgeschrittene Multivariate Analysemethoden, Springer-Verlag, Heidelberg [Veröffentlicht 2015]

[9] O. Gableske: Multilayer Perzeptron, Vorlesungsunterlage [online], Available: http://www.informatik.uni-ulm.de/ni/Lehre/WS04/ProSemNN/pdf/MLP.pdf, [Veröffentlicht 16.04.2005]

[10] Georgia Institute of Technology: Feedforward Neural Networks [online], Available: https://www.cc.gatech.edu/ san37/post/dlhc-fnn/, [Zugriff am 15.08.2019]

[11] JF. Couchot, R. Couturier: Convolutional Filters [online], Available: https://www.semanticscholar.org/paper/Steganalysis-via-a-Convolutional-Neural-Network-Couchot-Couturier/7e9708d9dc8b0a4ac2fa52eb384d67f52d7cbbe4/figure/0, [Veröffentlicht 2016]

[12] F. Moutarde: http://perso.mines-paristech.fr/fabien.moutarde/ES_MachineLearning /TP_convNets/convnet-notebook.html, [Zugriff am 15.08.2019]

[13] Undergraduate BS Computer Science: https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks, [Veröffentlicht 16.06.2017]

[14] A. Escontrela: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1, [Veröffentlicht 17.06.2018]

[15] O. Kramer: Computational Intelligence, Springer-Verlag, Wiesbaden, ISBN 978-3-540-79739-5, [Veröffentlicht 2009]

[16] M. Salvaris, D. DeanWee, H. Tok: Deep Learning with Azure, Apress, ISBN 978-1-4842-3679-6, [Veröffentlicht 2018]

[17] FreeCAD source documentation [online], Available: https://www.freecadweb.org/api/, [Zugriff am 04.09.2019]

[18] scikit-learn user guide Release 0.21.3, scikit-learn developers, [Veröffentlicht 29.07.2019]

[19] A. Waleed: Traffic Sign Recognition with TensorFlow [online], Available: https://medium.com/@waleedka/traffic-sign-recognition-with-tensorflow-629dffc391a6, [Zugriff am 06.09.2019]

[20] K. Willems: TensorFlow Tutorial [online], Available: https://www.datacamp.com/com munity/tutorials/tensorflow-tutorial,

[Zugriff am 06.09.2019]

[21] S. Saha: A Comprehensive Guide to Convolutional Neural Networks [online], Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 , [Zugriff am 06.09.2019]

[22] C. R. Qi, H. Su, K. Mo, and L. J. Guibas: Pointnet: Deep learning on point sets for 3d classification and segmentation, arXiv :1612.00593, [Veröffentlicht 2016]

[23] Altairuniversity.com: [Online], Available: https://altairuniversity.com/wp-content/uploads/2014/02/meshing.pdf, [Zugriff am 14.04.2018].

[24] F. Rieg, R. Hackenschmidt, B. Alber-Laukant: Finite Elemente Analyse für Ingenieure, Hanser Verlag, München, ISBN: 978-3-446-45639-6, [Veröffentlicht 2012]

[25] G. Silber, F. Steinwender: Bauteilberechnung und Optimierung mit der FEM, B. G. Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, ISBN: 978-3-519-00425-7, [Veröffentlicht 2005]

[26] Prof. S. Felsner, Vorlesung: Algorithmische und Diskrete Mathematik, http://page.math.tu-berlin.de/ felsner/Lehre/GrTh05/Graphentheorie.pdf, Technische Universität Berlin, Institut für Mathematik

[27] H. Goering, H.-G. Roos, and L. Tobiska: Finite–Elemente–Methode, Akademie Verlag, Berlin, dritte edition, [Veröffentlicht 1993]

[28] G. Golub and C. Van Loan: Matrix Computations, Johns Hopkins University Press, second edition, [Veröffentlicht 1989]

[29] W. Neundorf, Bandbreitenreduktion: https://www.db-thueringen.de/servlets/ MCR-FileNodeServlet/dbt_derivate_00008951/IfM_Preprint_M_02_07.pdf, Universität Ilmenau , Institut für Mathematik, [Veröffentlicht 01.09.2002]

[30] Th. Meis, U. Marcowitz: Numerische Behandlung partieller Differentialgleichungen. Springer-Verlag Berlin, [Veröffentlicht 1978]

[31] Keras Documentation [online], Available: https://keras.io/examples/mnist_cnn/, [Zugriff am 22.09.2019]

[32] Kitware Europe [online], Available: https://www.paraview.org/, [Zugriff am 23.09.2019]

[33] S.Jung. Dissertation: Erarbeitung eines wissensbasierten Systems für die Modellierung von Mehrkomponenten-Mehrphasen-Gleichgewichten, Martin-Luther-Universität Halle-Wittenberg, [Veröffentlicht 11.07.2002]

[34] Z. Styczynski, K. Rudion, A. Naumann: Einführung und Grundbegriffe der Expertensysteme, Springer-Vieweg 2017, ISBN 978-3-662-53171-6, [Veröffentlicht 11.04.2017]

[35] F.Puppe: Einführung in Expertensysteme, Springer-Verlag , ISBN 978-3-540-19481-1 [Veröffentlicht 2017]

[36] B. Bohn, J. Garcke, R. Iza-Teran: Analysis of Car Crash Simulation Data withNonlinear Machine Learning Methods, International Conference on Computational Science, ICCS 2013

[37] D.Selvamuthu, D. Das: Introduction to Statistical Methods, Design of Experiments and Statistical Quality Control, Springer Nature Singapore, ISBN 978-981-13-1735-4, [Veröffentlicht 2018]

[38] B. Aunkofer: Maschinelles Lernen mit Entscheidungsbaumverfahren [online] , Available: https://data-science-blog.com/blog/2017/02/13/entscheidungsbaumverfahren-artikelserie/ [Zugrfiff am 26.09.2019]

[39] A. Pillai, Parameter estimation for the spot weld design in automotive construction, 2019

[40] T. DeRose: Coordinate-free geometric programming, Seattle, Washingtion, Technischer Report, [Veröffentlicht 1994]

Anhang