

GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER
INSTITUT FÜR INFORMATIONSVERARBEITUNG

Bachelorarbeit

**Zeitreihen-zu-Bild-Encoding für
Maschinenzustandsüberwachung mittels Deep
Vision Neural Networks**

Anh Khoa Pham

Matrikelnr. 10022723

Betreuer: M. Sc. Quy Le Xuan
Erstprüfer: Prof. Dr.-Ing. J. Ostermann
Zweitprüfer: Prof. Dr.-Ing. B. Rosenhahn

Hannover, September 2023

Die Richtlinien im *Merkblatt zur Ausführung von Abschlussarbeiten am Institut für Informationsverarbeitung* sind Bestandteil der Aufgabenstellung. Eine anderweitige Verwendung der Arbeit, insbesondere die Bekanntgabe an Dritte, bedarf der Zustimmung des Erstprüfers.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie die aus fremden Quellen indirekt oder direkt übernommenen Inhalte und Gedanken als solche kenntlich gemacht habe. Dies schließt die Verwendung von elektronischen Medien sowie text- oder andere inhaltsgenerierenden IT-Werkzeuge wie ChatGPT oder GitHub Copilot ein. Ich versichere, dass alle abgegebenen Fassungen der Arbeit, im Besonderen gedruckte und elektronische Ausführungen, vollständig übereinstimmen und die Arbeit nicht zum Erwerb eines anderen Leistungsnachweises eingereicht wurde. Ich stimme der Verwertung meiner Arbeit durch das Institut für Informationsverarbeitung in den folgenden Punkten zu:

- Aufnahme der gedruckten und der elektronischen Fassung der Arbeit in die Institutsbibliothek,
- Übermittlung meiner Arbeit auch an externe Dienste zur Plagiatsprüfung,
- Vervielfältigung der gesamten Arbeit oder von Auszügen für Lehrzwecke und
- Wiedergabe der Arbeit durch Bild- und Tonträger.

Hannover, den 25. September 2023


Pham Anh Khoa

Inhaltsverzeichnis

1. Einleitung	3
2. Grundlagen	5
2.1. Deep Learning	5
2.2. Vorverarbeitung	5
2.3. Künstliche neuronale Netze	7
2.3.1. Fully-connected Layer	8
2.3.2. Aktivierungsfunktion	9
2.3.3. Convolutional Layer	10
2.3.4. Pooling Layer	11
2.3.5. Residual-Block	12
2.3.6. Verlustfunktion	13
2.3.7. Gradient Descent	14
2.4. Regulierung	16
2.4.1. L1-Regulierung	16
2.4.2. L2-Regulierung	16
2.4.3. Dropout	17
2.4.4. Batch-Normalisierung	18
2.5. Kreuzvalidierung	18
2.6. Transfer Learning	19
2.6.1. Fine-Tuning	19
3. Ansatzdetails	21
3.1. Fensterung	21
3.2. Piecewise Aggregate Approximation	22
3.3. Zeitreihen-zu-Bild-Encoding	23
3.3.1. Kurzzeit-Fourier-Transformation	23
3.3.2. Log-Mel-Spektrogramm	26
3.3.3. Kontinuierliche Wavelet-Transformation	29
3.3.4. Rekurrenzplot	34
3.3.5. Gramian-Winkelfeld	35
3.3.6. Markov-Übergangsfeld	36
3.4. Temporal Cycle-Consistency Learning	39
3.4.1. Zykluskonsistenz und Optimierung	40
3.4.2. Zykluskonsistenz von Zeitreihen	42
3.5. Hyperparameter-Tuning	45

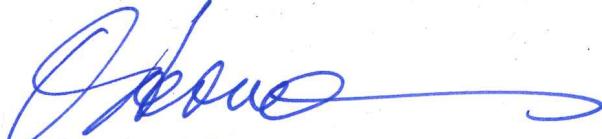
4. Experimente und Ergebnisse	47
4.1. Datensätze	47
4.1.1. Score	49
4.1.2. Verwandte Arbeiten	50
4.2. Vorbearbeitung der Zeitreihen	50
4.3. Ansätze und Ergebnisse	51
4.3.1. Im_CNN	52
4.3.2. Ts_im_CNN	53
4.3.3. Pann	54
4.3.4. Tcl_im_CNN	54
5. Zusammenfassung	57
Literaturverzeichnis	59
A. Visualisierung	63
B. Modellstruktur	69
C. Ergebnisse	73

23. März 2023

Bachelor Thesis
for
Anh Khoa Pham
10022723**Time Series-to-Image Encoding for Machinery Health Monitoring with
Deep Vision Neural Networks**

Machinery Health Monitoring (MHM) is one of the most essential components of modern maintenance management systems in many industries including manufacturing, aerospace, and transportation. Among others, the primary goal of MHM is to reliably detect as well as predict potential failures, allowing maintenance to be efficiently planned and executed in a proactive manner. The increasing availability of condition data, mostly in the form of time series, and the recent breakthroughs in artificial intelligence are the key factors that make data-driven solutions for MHM extremely promising. Inspired by successful use cases e.g., in medical signal analysis and speech recognition, the idea of applying time series-image encoding as a preprocessing step for further analyses using deep vision models, instead of directly using time series data as inputs, is gaining increasing interest. By doing this, complex and hidden patterns of the raw time series data can be transformed into other forms that might be more apparent and easier to be processed using network architectures dedicated to image processing. In addition, encoding time series data as images also allows us to leverage the power of advanced deep neural network models that were pre-trained on large-scale image datasets for multiple computer vision tasks.

The main objective of this thesis is to investigate the use of time series-to-image encoding techniques in conjunction with deep vision neural networks for the task of MHM. On the basis of literature research, different existing methods and pipelines have to be analyzed, adapted, and/or extended. The most promising ideas have to be implemented and evaluated in comparison with other state-of-the-art methods using available benchmark datasets. In the implementation, adequate attention should be paid to a modular design and good documentation of the software, as well as to the reproducibility of the experimental results.



Prof. Dr.-Ing. J. Ostermann

Abstract

Remaining useful life (RUL) ist eine wichtige Aufgabe bei der Prognose und im Gesundheitsmanagement der Maschine oder Komponente, um Sicherheit zu gewährleisten, Kosten zu reduzieren und Wartung der komplexen Systeme sicherzustellen. Traditionelle Ansätze zur RUL-Bestimmung basieren oft auf handgefertigten Merkmalen und physikbasierten Modellen, was die Genauigkeit einschränken kann. Heutzutage ist die Datenerfassung und -speicherung einfacher geworden und mit der rasanten Entwicklung der Technologie und der herausragenden Rechenleistung von Grafikkarten hat Deep Learning vielversprechende Ergebnisse erzielt. Insbesondere haben rekurrente neuronale Netze (RNN) und faltende neuronale Netze (CNN) vielversprechende Ergebnisse bei der RUL-Prognose gezeigt, indem sie komplexe Muster und Darstellungen aus großen Daten lernen können. Die Aufgabe in dieser Arbeit ist RUL der Lager durch die von den Sensoren gemessenen Zeitreihen der Vibrationen als Inputs des Modells vorherzusagen. Basierend auf der Fähigkeit, vollständigere und vielfältigere Informationen der Bilder als der Zeitreihen zu implementieren, werden verschiedene Zeitreihen-zu-Bild-Konvertierung vorgeschlagen. Komplexe Architekturen von Deep Vision Neural Networks werden verwendet, die Merkmalen der transformierten Bilder zu lernen, und zum Schluss den Output als RUL zu bestimmen.

1. Einleitung

Das letzte Jahrzehnt war die Zeit, in der faltende neuronale Netze (CNN oder englisch convolutional neural network) explodierten. Der Anfang war die Einführung von Alexnet im Jahr 2012 [1], der auch als state-of-the-art galt und im gleichen Jahr ImageNet Challenge gewann. Mit dem Training auf großen Datensätzen wie ImageNet sowie der Möglichkeit, die Rechenleistung der Grafikkarte voll auszunutzen, zeigen CNNs herausragende Leistungsfähigkeit bei der Bildklassifizierung. Was CNNs so erfolgreich machen können, ist, dass sie in der Lage sind, Merkmale eines Bildes mithilfe der gelernten Kernel zu erkennen, zum Beispiel vertikale Kanten, horizontale Kanten usw. (siehe Abbildung 1.1). Seitdem gibt es viele weitere vortrainierte Modelle, die bessere Ergebnisse

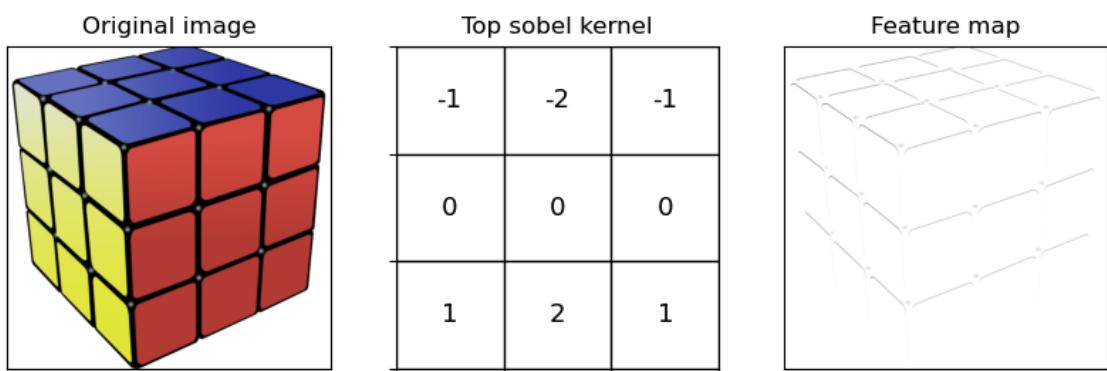


Abbildung 1.1.: Ein Top-Sobel-Kernel kann die oberen horizontalen Kanten erkennen[2].

bei ImageNet-Datensätzen mit weniger Parametern erreichen, z.B., GoogleNet [3], ResNet [4], EfficientNet [5] usw. Dadurch ist die Bildklassifizierung durch Fine-Tuning und Transfer Learning viel zugänglicher geworden.

Um ein Modell zu trainieren, die remaining useful life (RUL) vorherzusagen, nehmen die Modelle in einigen Fällen die Zeitreihen als Inputs, die z.B. von den Sensoren gemessenen Werte von Temperatur, Vibration, Geräusch, Geschwindigkeit, usw. der Maschine sein können. Ihre Eigenschaften können normalerweise nur entlang des Zeitintervalls dargestellt werden, was manchmal nicht genug ist, um zu schätzen, ob eine Maschine noch benutzen lässt.

Das Ziel dieser Arbeit besteht darin, Zeitreihen, die den Gesundheitszustand beschreiben, mithilfe einiger vorgeschlagener Konvertierungen zu Bildern umzuwandeln, wodurch bessere Ergebnisse bei der RUL-Vorhersage erwartet werden. Der Grund für diesen An-

satz besteht darin, die überlegenen Bildverarbeitungsfähigkeiten von CNNs auszunutzen und Bilder können nicht nur mehr Informationen liefern, sondern auch den Zustand von Maschinen besser beschreiben als Zeitreihen. Ein Beispiel hierfür kann durch Abbildung 1.2 dargestellt werden, das Bild also Spektrogramm beschreibt ein Signal sowohl im Zeit- als auch im Frequenzbereich.

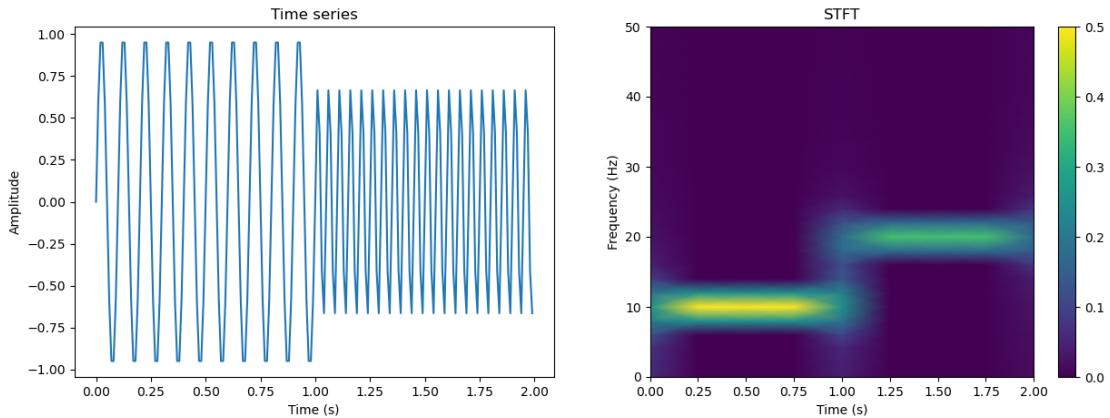


Abbildung 1.2.: Zeit-Frequenz-Darstellung eines Zeitreihens

Allerdings bringt die Konvertierung zu Bildern auch gewisse Herausforderungen mit sich, wie etwa die richtige Auswahl der Hyperparameter, das Problem des Rechenaufwands und die Struktur des Modells. Die Hyperparameter der Transformation bestimmen, wie die konvertierten Bilder aussehen und ob sie die charakteristischen Merkmale des Bildes zeigen können, z. B. den Unterschied zwischen einer gesunden und nicht benutzbaren Maschine. Die Hyperparameter des Modells wirken sich direkt auf die Leistung aus. Ihre Auswahl ist immer ein großes Problem von CNN Modellen im Besonderen und von Deep-Learning-Algorithmen im Allgemeinen. Es ist auch wichtig zu beachten, dass Modelle nur dann leistungsstark werden können, wenn sie aus großen Datenmengen trainiert werden.

Im nächsten Kapitel dieser Arbeit werden die Grundprinzipien von Deep Learning, der Aufbau, die Komponenten und das Training künstlicher neuronaler Netze vorgestellt. In Kapitel 3 geht es um die speziellen Methoden zur Lösung des Problems, insbesondere die Konvertierung von Zeitreihen zu Bildern, und deren Verwendung. Der in dieser Arbeit verwendete Datensatz, die Ansätze und die Ergebnisse werden in Kapitel 4 ausführlich vorgestellt. In Kapitel 5 erfolgt eine Synthese der gesamten Arbeit sowie der bestehenden Herausforderungen.

2. Grundlagen

In diesem Kapitel werden die Grundlagen neuronaler Netze und Datenvorverarbeitungsmethoden vorgestellt, die in dieser Arbeit verwendet werden.

2.1. Deep Learning

Deep Learning (DL) ist ein Teilgebiet des maschinellen Lernens (ML oder englisch machine learning) und der künstlichen Intelligenz (AI oder englisch artificial intelligence). DL-Algorithmen konzentrieren sich häufig auf das Training der Layer tiefer neuronaler Netze, die vom intelligenten menschlichen Gehirn selbst inspiriert sind und Informationen mithilfe eng verbundener Zellen verarbeiten können [6].

Heutzutage werden DL-Algorithmen häufig bei der Verarbeitung komplexer Informationen wie Bilder, Stimmen, Schalle und natürliche Sprachen eingesetzt, wodurch die AI den Menschen nahe kommt und ihnen bei sowohl einfachen als auch komplizierten Aufgaben unterstützen kann. Zu den heute beliebten AI gehören AlphaGo, Deep Blue, ChatGPT, Alexa, Google Assistance, Siri, Contana usw.

Zu den Arten von Algorithmen für ML und DL gehören:

- **Überwachtes Lernen** (englisch supervised learning) ist ein Ansatz in ML, bei dem ein Modell anhand gekennzeichneter Daten trainiert wird. Das heißt, für jeder Input gibt es einen entsprechenden Output. Die Aufgabe des Modells besteht darin, die Zuordnungen zwischen dem Input und Output zu lernen und zu versuchen, die Labels der unsichtbaren Daten vorherzusagen [7].
- **Unüberwachtes Lernen** (englisch unsupervised learning) ist ein Ansatz, bei dem das Modell nur die Daten ohne Labels als Input nimmt. Das Hauptziel des Modells besteht darin, die gemeinsamen Merkmale, Strukturen und Beziehungen der Inputs zu ermitteln [7].

2.2. Vorverarbeitung

Bevor Daten in das Modell aufgenommen werden, ist die Vorverarbeitung der Daten unerlässlich und wirkt sich direkt auf die Leistung des Modells aus. Rohdaten können viel Rauschen enthalten, unvollständig oder zwischen den Klassen unausgeglichen sein. In

diesem Abschnitt werden nur grundlegende Datenvorverarbeitungsmethoden vorgestellt. Spezielle Vorverarbeitungsmethoden für diese Arbeit werden im Kapitel 3 ausführlicher behandelt.

Min-Max-Normalisierung

Die Werte der Rohdaten weisen häufig unterschiedliche Verteilungen auf und auch die Bereiche ihrer Werte sind unterschiedlich. Wenn diese Werte direkt auf das Netz angewendet werden, wird das Training instabil, da einige große Werte die kleineren überfordern. Damit das Modell die Features vom Rohdatensatz als fair und gleichwertig betrachten kann, ist daher eine Normalisierung erforderlich.

Min-Max-Normalisierung (englisch min-max normalization oder rescaling) ist eine Vorverarbeitungsmethode, die den Wertebereich innerhalb eines bestimmten Intervalls ändert und durch Formel 2.1 dargestellt wird [8].

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \times (max_{\text{new}} - min_{\text{new}}) + min_{\text{new}} \quad (2.1)$$

Dabei gilt:

- x_{\min} und x_{\max} sind jeweils der minimale und maximale Wert des Features vom Input.
- min_{new} und max_{new} sind jeweils der minimale und maximale Wert vom skalierten Wertebereich.

Diese Normalisierungsmethode eignet sich, wenn die Datenverteilung gleichmäßig verteilt ist, das heißt, die Wahrscheinlichkeit des Auftretens jedes Werts im Datensatz ist gleich. Ein typisches Beispiel sind Bilder, bei denen die Maximal- und Minimalwerte jedes Pixels im Voraus mit 0 und 256 bekannt sind. Der normalisierte Bereich ist normalerweise $[0,1]$ oder $[-1,1]$.

Standardisierung

Wenn der Datensatz eine Gauß-Verteilung aufweist, ist die Standardisierung (englisch standardize) die empfohlene Wahl. Wenn diese Normalisierungsmethode angewendet wird, haben die Daten einen Mittelwert von 0 und eine Standardabweichung von 1 [8].

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma} \quad (2.2)$$

Dabei gilt:

- μ ist der Mittelwert der Inputs.
- σ ist die Varianz der Inputs.

Datenreinigung

Bei der Datenreinigung (englisch data cleaning) handelt es sich um eine Methode zum Entfernen ungeeigneter Werte oder Features, die nur konstante Werte enthalten, die für das Training des Modells nicht nützlich sind.

Datenimputation

Datenimputation ist eine Methode zum Auffüllen fehlender Werte in einem Datensatz mithilfe des Mittelwerts oder Medians.

Ausreißererkennung

Bei der Ausreißererkennung (englisch outlier detection) geht es darum, Werte zu identifizieren, die von den übrigen Werten abweichen oder anomale sind, und diese zu ändern. Dies kann passieren, wenn ein Problem mit Datenerfassungstools wie Sensoren vorliegt.

Labelscodierung

Bei der LabelsCodierung (englisch label encoding) handelt es sich um die Umwandlung kategorialer Labels in numerische Werte.

2.3. Künstliche neuronale Netze

Ein künstliches neuronales Netz (ANN oder englisch artificial neural network) besteht aus drei Hauptkomponenten, die Input Layer, Hidden Layer und Output Layer sind. Nachdem die Rohdaten den Input Layer durchlaufen haben, werden sie mit den Gewichten (englisch weights) und Bias in diesen Layern berechnet, die während des Trainings gelernt werden, um Ergebnisse im Output Layer zu erstellen [9]. Die Anzahl der Gewichte und Bias ist abhängig von der Komplexität vom Hidden Layer. Abbildung 2.1 zeigt ein grundlegendes neuronales Netz mit nur einem fully-connected Layer, der später im Abschnitt 2.3.1 detaillierter erläutert wird. Die Auswahl der Anzahl sowie der Art des Layers kommt auf die Aufgabe und den Input an. Mit vielen Parametern kann das Modell auf Trainingsdatensatz perfekte Ergebnisse erzielen, garantiert jedoch nicht die gleiche Leistung auf Testdatensatz. Dies wird auch als „overfitting“ bezeichnet [10]. Wenn zu wenige Parameter vorhanden sind, kann das Modell die Eigenschaften sowie die Beziehung zwischen den Inputs und Labels während des Trainings kaum erlernen, was „underfitting“ genannt wird [11]. In diesem Abschnitt werden erklärt, was in einem Netz gibt, und wie ein Modell seine Outputs näher an die Labels verbessern kann.

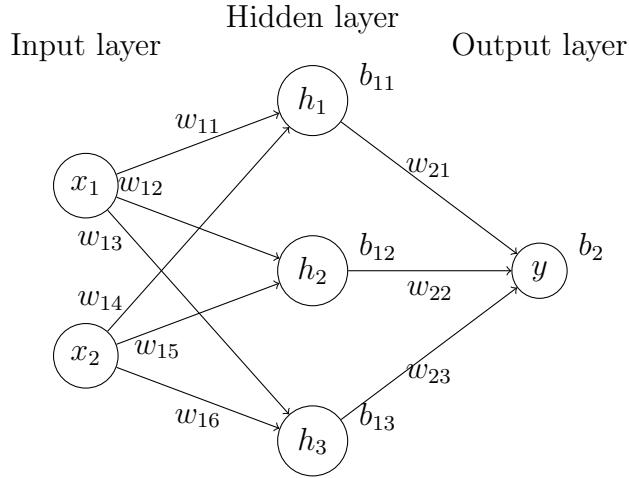


Abbildung 2.1.: Ein künstliches neuronales Netz

2.3.1. Fully-connected Layer

Fully-connected Layer oder linearer Layer ist der grundlegendste Layer eines Modells, aber spielt eine große Rolle, das Modell zu unterstützen, komplexe Darstellung und Beziehung in Daten zu verstehen. Ein linearer Layer kann aus einem (siehe Abbildung 2.2) oder mehreren (siehe Abbildung 2.1) Inputs und Outputs bestehen.

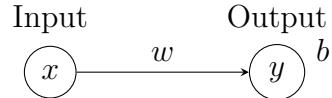


Abbildung 2.2.: Ein fully connected Layer mit einem Input und Output

Der Output eines linearen Layers, der auch als Einheit (englisch unit) oder Neuron genannt wird, ist eine Berechnung, die den Input empfängt, kann durch Formel 2.3 beschrieben werden.

$$y = xw^T + b \quad (2.3)$$

Die Berechnung des Outputs des Netzes in Abbildung 2.1 lässt sich einfach wiederum durch Formel 2.3 bestimmen.

$$\mathbf{h} = [h_1 \ h_2 \ h_3] = \mathbf{x}\mathbf{w}_1^T + \mathbf{b}_1 = [x_1 \ x_2] \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{14} & w_{15} & w_{16} \end{bmatrix} + [b_{11} \ b_{12} \ b_{13}] \quad (2.4)$$

$$\mathbf{h} = [x_1w_{11} + x_2w_{14} + b_{11} \ x_1w_{12} + x_2w_{15} + b_{12} \ x_1w_{13} + x_2w_{16} + b_{13}] \quad (2.5)$$

$$\mathbf{y} = \mathbf{h}\mathbf{w}_2^T + \mathbf{b}_2 = [h_1 \ h_2 \ h_3] \begin{bmatrix} w_{21} \\ w_{22} \\ w_{23} \end{bmatrix} + [b_2] = h_1w_{21} + h_2w_{22} + h_3w_{23} + b_2 \quad (2.6)$$

$$y = (x_1w_{11} + x_2w_{14} + b_{11})w_{21} + (x_1w_{12} + x_2w_{15} + b_{12})w_{22} + (x_1w_{13} + x_2w_{16} + b_{13})w_{23} + b_2 \quad (2.7)$$

$$y = x_1(w_{11}w_{21} + w_{12}w_{22} + w_{13}w_{23}) + x_2(w_{14}w_{21} + w_{15}w_{22} + w_{16}w_{23}) + (w_{21}b_{11} + w_{22}b_{12} + w_{23}b_{13} + b_2) \quad (2.8)$$

$$\mathbf{y} = [x_1 \ x_2] \underbrace{\begin{bmatrix} w_{11}w_{21} + w_{12}w_{22} + w_{13}w_{23} \\ w_{14}w_{21} + w_{15}w_{22} + w_{16}w_{23} \end{bmatrix}}_{\mathbf{w}'^T} + \underbrace{\begin{bmatrix} w_{21}b_{11} + w_{22}b_{12} + w_{23}b_{13} + b_2 \end{bmatrix}}_{\mathbf{b}'} \quad (2.9)$$

$$\mathbf{y} = \mathbf{x}\mathbf{w}'^T + \mathbf{b}' \quad (2.10)$$

2.3.2. Aktivierungsfunktion

Formel 2.10 des Beispiels im Abbildung 2.1 hat gezeigt, dass die Outputs eines Netzes linear sind, ohne Berücksichtigung, wie viele verborgene Neuronen und verborgene Layer verfügbar sind, was in der Tat sehr unpraktisch ist. Eigentlich sind die Labels im Raum sehr unordentlich. Unabhängig davon, ob die Aufgabe einfach oder komplex ist, ist daher immer eine nichtlineare Darstellung des Modells erforderlich. Um dieses Problem zu lösen, ist es immer empfehlenswert, eine sogenannte Aktivierungsfunktion direkt nach einem, jedem oder einigen Layern zu hinzufügen. Diese Funktionen ändern die Werte der Neuronen im Netzwerk abhängig von ihren vorherigen Werten, wodurch die Outputs des Netzes ihre Linearität verlieren. Abbildung 2.3 zeigt einige häufige benutzte Aktivierungsfunktionen und ihre Formeln sind alle in [12] referenziert.

ReLU (siehe Formel 2.11) liefert nicht nur Nichtlinearität, sondern ist auch eine einfache und recheneffiziente Aktivierungsfunktion, die in der Lage ist, Neuronen mit negativen Werten zu deaktivieren, oder mit anderen Worten, sie auf null zu setzen, kann durch Formel 2.11 dargestellt werden.

$$ReLU(x) = \max(0, x) \quad (2.11)$$

Allerdings ist ReLU nicht ohne Schwächen, eine davon kann als „dying ReLU“ erwähnt werden. Dieses Problem tritt auf, wenn alle Neuronen einen Null-Output haben, also alle zuvor negative Werte hatten. Dies führt dazu, dass die Gewichte nicht über Gradient Descent und Backpropagation aktualisiert werden, das heißt, das Netzwerk ist nicht mehr lernfähig. Eine Möglichkeit, dieses Problem zu lösen, ist Leaky ReLU (siehe Formel 2.12). Diese Funktion nimmt den negativen Wert von αx oder positiven Wert von x an, wobei α eine sehr kleine Konstante ist, wodurch vermieden wird, dass negative Neuronen unterdrückt werden und dennoch zum Lernprozess beitragen können.

$$Leaky_ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} \quad (2.12)$$

Eine weitere Aktivierungsfunktion ist Sigmoid (Formel 2.13), die ebenfalls die Fähigkeit hat, die Linearität zu verlieren. Sie bildet die Werte der Neuronen zwischen 0 und 1 ab. Aufgrund dieser besonderen Eigenschaft wird die Sigmoidfunktion auch bei binären

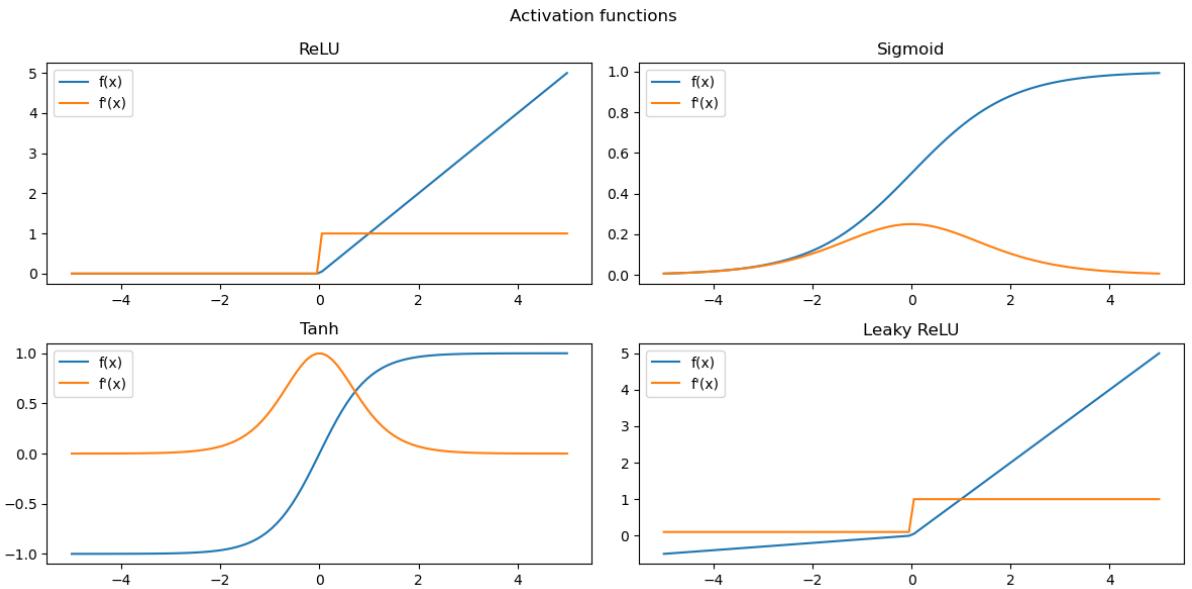


Abbildung 2.3.: Aktivierungsfunktionen

Klassifizierungsaufgaben angewendet.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.13)$$

Ganz ähnlich wie bei Sigmoid wird der Output der Tanh-Funktion (siehe Formel 2.14) über einen bestimmten Bereich abgebildet, jedoch breiter von -1 bis 1.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.14)$$

2.3.3. Convolutional Layer

Wie schon im Kapitel 1 erwähnt, werden in dieser Arbeit die Zeitreihen verarbeitet, zu Bildern zu konvertieren. Um die Merkmale eines Bildes wie ein Beispiel in Abbildung 1.1 zu lernen, wird ein sogenannter faltender Layer (englisch convolutional layer) benutzt. Dieser Layer besteht aus Kernel, die entsprechend der Bildtiefe übereinander gelegt werden, und trainierbare Parameter enthalten. Dann lassen sie auf den Input anwenden, indem sie über das Bild geschoben werden. An jedem Pixel wird die elementweise Multiplikation zwischen den Parametern des Kernels und dem entsprechenden Teil des Bildes durchgeführt. Formel 2.15 zeigt ein Beispiel der Berechnung der zwei ersten Pixel von Feature-Map für den Fall ohne Padding, Schrittweite und Dilation gleich ein

in der Abbildung 2.4, wobei w_i und b_i erlernbare Parameter sind.

$$\begin{aligned} y_{11} &= x_{11}w_{11} + x_{12}w_{12} + x_{13}w_{13} + x_{21}w_{21} + x_{22}w_{22} + x_{23}w_{23} \\ &\quad + x_{31}w_{31} + x_{32}w_{32} + x_{33}w_{33} + b \quad (2.15) \\ y_{12} &= x_{12}w_{11} + x_{13}w_{12} + x_{14}w_{13} + x_{22}w_{21} + x_{23}w_{22} + x_{24}w_{23} \\ &\quad + x_{32}w_{31} + x_{33}w_{32} + x_{34}w_{33} + b \end{aligned}$$

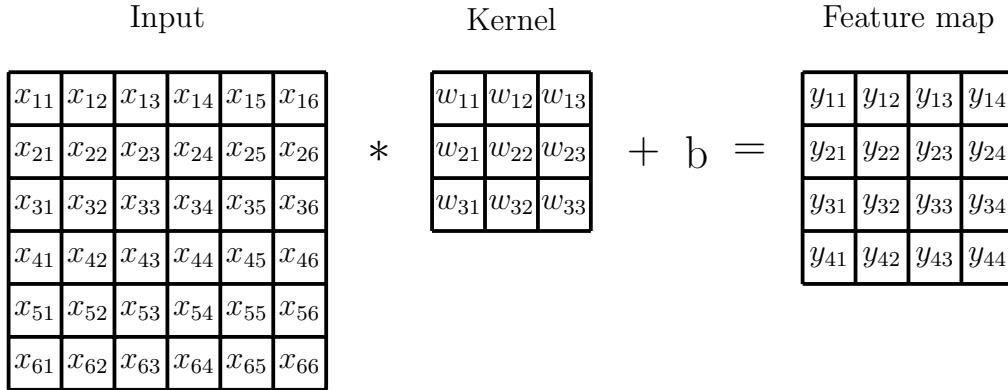


Abbildung 2.4.: 2D-Faltung zwischen 6x6-Bild und 3x3-Kernel mit Schrittweite, Dilation gleich eins und ohne Padding

Beim faltenden Layer gibt es einige Hyperparameter [13], die sind:

- **Kernelgröße** beschreibt die Abmessung der Höhe und Breite des Kernels, und wird normalerweise als 3x3, 5x5 oder 7x7 genommen.
- **Schrittweite** gibt an, wie sich der Kernel über den Input bewegt oder gleitet.
- **Padding** ist das Hinzufügen von konstanten Werten an den Rändern von Bildern, um sicherzustellen, dass die Informationen an dieser Position während der Faltung zwischen dem Kernel und dem Bild nicht verloren gehen.
- **Dilation** bezieht sich auf den Abstand zwischen den Werten im Kernel während der Faltung an dem Input.

Der Output des faltenden Layers wird als Feature-Map bezeichnet und hat die Größe mithilfe der Formel 2.16 berechnet werden [14].

$$y_{\text{shape}} = \left\lfloor \frac{x_{\text{shape}} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor \quad (2.16)$$

2.3.4. Pooling Layer

Nachdem die Feature-Maps durch den faltenden Layer erstellt wurden, werden sie nun in einen Layer eingefügt, der den Effekt hat, die Größe der Feature-Maps zu reduzieren,

um den Berechnungsprozess zu optimieren, aber dennoch die Pixel beizubehalten, die die Eigenschaften des Bildes zeigen. Formel 2.17 und Abbildung 2.5 beschreiben die

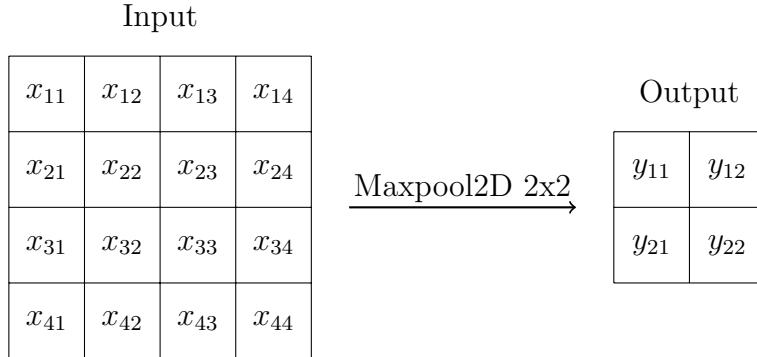


Abbildung 2.5.: Pooling Layer reduziert die Größe von Feature-Map.

Berechnung des Outputs vom Maxpool2D-Layer mit der Kernelgröße gleich 2x2 für den Fall ohne Padding, Dilation gleich ein und Schrittweite gleich zwei.

$$\begin{aligned}
 y_{11} &= \max(x_{11}, x_{12}, x_{21}, x_{22}) \\
 y_{12} &= \max(x_{13}, x_{14}, x_{23}, x_{24}) \\
 y_{21} &= \max(x_{31}, x_{32}, x_{41}, x_{42}) \\
 y_{22} &= \max(x_{33}, x_{34}, x_{43}, x_{44})
 \end{aligned} \tag{2.17}$$

Genauso wie faltender Layer hat pooling Layer die Hyperparameter wie Kernelgröße, Schrittweite, Padding und Dilation. Außerdem hat der Output die Größe wie in Formel 2.16.

2.3.5. Residual-Block

Residual-Block wurde erstmals 2015 mit ResNet eingeführt. Die Grundidee besteht darin, dass der Output eines Layers durch eine sogenannte „Skip-Verbindung“ also „Residual-Verbindung“ zu einem anderen Layer tiefer im neuronalen Netzwerk hinzugefügt wird. Dies kann durch Formel 2.18 und Abbildung 2.6 links gezeigt werden [4].

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x} \tag{2.18}$$

Eine weitere Variante des Residual-Blocks ist der Bottleneck-Block, wie in Abbildung 2.6 dargestellt, der über faltende Layer mit der Filtergröße 1x1 verfügt, um die Tiefe oder Anzahl der Kanäle zu reduzieren und sie dann wieder zu erhöhen. Ein faltender Layer hat dazwischen eine Filtergröße 3x3, der deshalb den Input und Output weniger Dimensionen hat [4]. Sowohl der Residual- als auch der Bottleneck-Block werden häufig in CNN verwendet, da sie dazu beitragen, das Explodieren und Verschwinden vom Gradient zu vermeiden. Darüber hinaus tragen sie dazu bei, dass das Modell schneller konvergiert und nicht so viele Ressourcen der Rechen verbraucht.

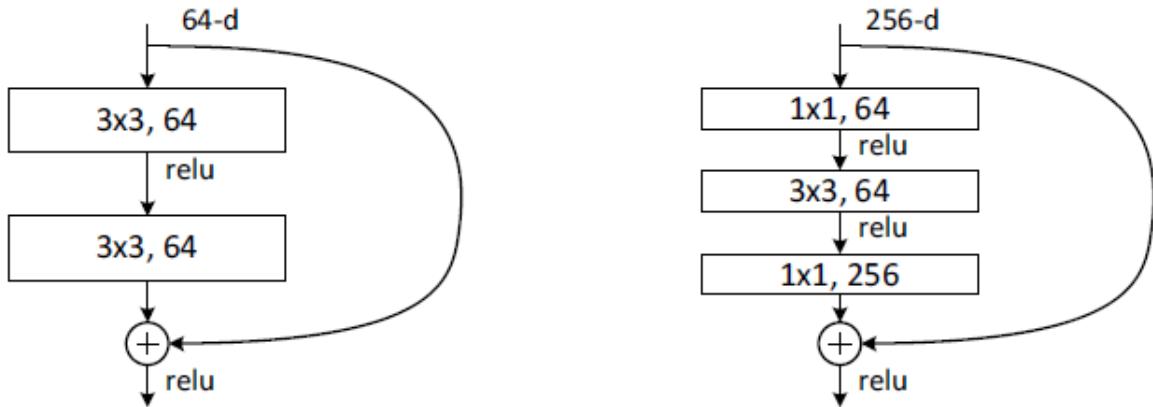


Abbildung 2.6.: Residual- und Bottleneck-Block [4]

2.3.6. Verlustfunktion

Verlustfunktion (englisch loss function) spielt eine große Rolle im Trainingsprozess, indem sie angibt, wie falsch die Vorhersagen im Vergleich zum Label sind. Für unterschiedliche Probleme erfordern verschiedenen Verlustfunktionen.

Lineare Regression

Lineare Regression ist ein Problem, bei dem das Modell Werte basierend auf einem oder mehreren Inputs generiert. Häufig werden mittlere quadratische Abweichung (MSE oder englisch mean squared error), mittlere absolute Fehler (MAE oder englisch mean absolute error) und Wurzel des mittleren quadratischen Fehlers (RMSE oder englisch root mean squared error) verwendet, indem sie den Durchschnitt der quadrierten Abstände zwischen den vorhersagten und wahren Werten. Die drei Formeln 2.19, 2.20 und 2.21 sind die drei schon erwähnten Verlustfunktionen [15], wobei N die Anzahl der Proben ist. y_i und \hat{y}_i sind jeweils wahre und vorhersagte Werte.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.19)$$

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.20)$$

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (2.21)$$

Klassifizierung

Binäre Klassifizierung ist ein Problem, bei dem das Netz versucht, Daten in eine von zwei Klassen oder Kategorien zu klassifizieren. Die Verlustfunktionen sind in diesem Fall binäre Kreuzentropie. Wenn die Klassifizierungsaufgabe mit mehr Klassen ist, wird Kreuzentropie verwendet. Die zwei Verlustfunktionen werden jeweils durch die Formeln 2.22 und 2.23 dargestellt [15]. C ist in diesem Fall der Anzahl der Klassen.

$$BCE = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \quad (2.22)$$

$$CE = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij}) \quad (2.23)$$

2.3.7. Gradient Descent

Das Modell lernt mithilfe eines Optimierungsalgorithmus namens Stochastic Gradient Descent (SGD). Die Hauptidee besteht darin, die Parameter des Modells so zu aktualisieren, dass die Outputs des Modells nahe an den Labels liegt, oder mit anderen Worten, die Verlustfunktion hat einen Mindestwert. Zunächst werden die Verlustfunktion und ihr Gradient aus den Inputs und den im Modell vorhandenen Gewichten berechnet. Anschließend wird sie mit einem Hyperparameter namens Lernrate multipliziert, um die Größe des Sprungs im Lernprozess zu bestimmen, der einen großen Einfluss auf die Konvergenz des Algorithmus hat. Die Gewichte des Modells werden aktualisiert, indem die Sprunggröße vom Wert der aktuellen Gewichtung subtrahiert wird. Die Sprunggröße ist die Multiplikation zwischen Gradient und Lernrate [16]. Dieser Vorgang wird wiederholt, bis die Verlustfunktion den Minimalwert erreicht, das heißt, ihr Gradient ist jetzt null. Abbildung 2.7 zeigt, wie man das Minimum einer Verlustfunktion mit unterschiedlichen Lernraten nach vielen Iterationen mithilfe der Aktualisierungsregel (siehe Formel 2.24) von SGD ermittelt. Die Lernrate ist ein äußerst wichtiger Faktor, der bestimmt, ob das Modell gut lernen kann oder nicht. Eine zu große Lernrate kann dazu führen, dass der Algorithmus einen zu großen Sprung macht und möglicherweise über den Extrempunkt springt. Wenn sie zu klein ist, wird das Modell im Optimierungsprozess langsam. Da die Optimierung von Verlustfunktionen in der Praxis ein sehr komplexes Problem darstellt, empfiehlt es sich immer, verschiedene Lernraten auszuprobieren.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t) \quad (2.24)$$

Dabei gilt:

- θ_t ist der aktuelle Parameter.
- θ_{t+1} ist der aktualisierte Parameter.
- η ist die Lernrate oder Schrittweite (englisch learning rate).

- $J(\theta_t)$ ist die Verlustfunktion in Bezug auf den aktuellen Parameter.
- $\nabla J(\theta_t)$ ist der Gradient der Verlustfunktion in Bezug auf den aktuellen Parameter.

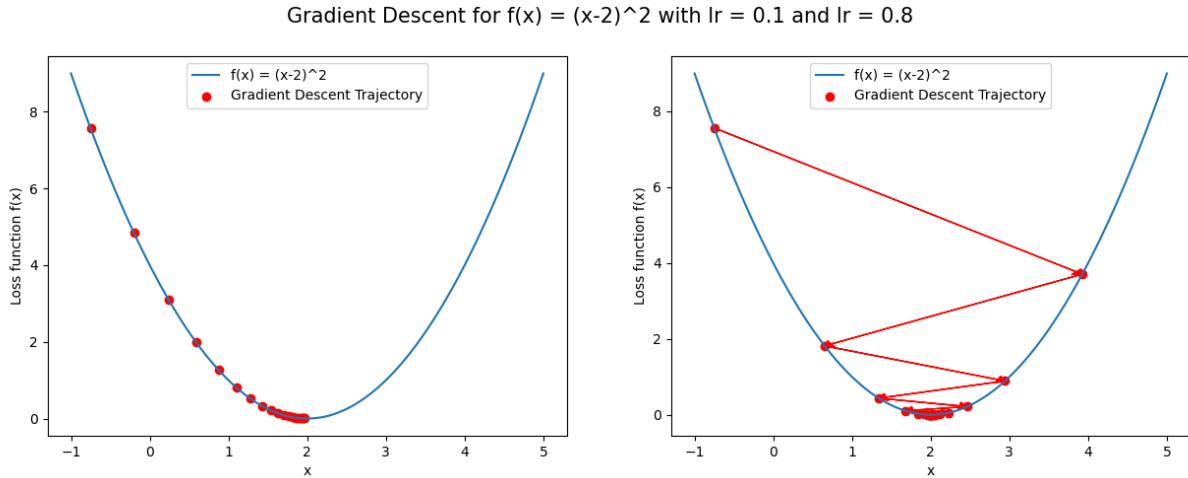


Abbildung 2.7.: Bestimmung des Minimum einer Verlustfunktion mit unterschiedlichen Lernraten mithilfe von SGD

Backpropagation

In einem neuronalen Netzwerk durchläuft der Eingabewert Hidden Layer und führt in diesen Layern Berechnungen mit Gewichten und Bias durch. Die vorhergesagte Werte im Output Layer hängen vollständig von allen diesen Parametern ab, was schon durch die Formel 2.9 dargestellt wird. Formel 2.24 wird die Gewichte gleichzeitig auf dem Laufenden halten. Dieser Vorgang wird Backpropagation genannt. In der Praxis kann es jedoch bei diesem Prozess zu Problemen kommen, die sind:

- Die Gradientenexplosion (englisch exploding gradient) stellt ein Problem dar, wenn der Gradient der Verlustfunktion während der Backpropagation zu groß ist. Dies führt dazu, dass die Parameter auch bei großen Werten aktualisiert werden und das Training instabil wird.
- Verschwindender Gradient (englisch vanishing gradient) ist das entgegengesetzte Konzept zum explodierenden Gradienten, wenn der Gradient der Verlustfunktion zu klein ist, was dazu führt, dass die Gewichte mit zu kleinen Werten aktualisiert werden, was dazu führt, dass das Modell kaum lernt.
- Lokales Minimumproblem ist eine Optimierungsherausforderung, bei der die Verlustfunktion an einem Punkt einen kleineren Wert als die angrenzenden Punkte erreicht hat, aber darin feststeckt und den Extrempunkt an einem anderen Ort nicht bestimmen kann.

Die Tatsache, dass das Modell mit dem Trainingsdatensatz gut lernen kann, garantiert

nicht, dass es mit dem Testdatensatz gut abschneidet. Overfitting ist schon immer ein häufiges Problem beim Deep Learning.

2.4. Regulierung

Bei der Regulierung (englisch regularization) handelt es sich um eine Reihe von Techniken, die im Abschnitt 2.3.7 erwähnten Probleme zu lösen.

2.4.1. L1-Regulierung

Die L1-Regulierung, auch Lasso-Regression genannt, fördert die Reduzierung von Gewichten auf null und wird durch Formeln 2.25 ausgedrückt.

$$\mathcal{L} = \mathcal{L}(y_i, \hat{y}_i) + \lambda \sum_{j=1}^M |w_j| \quad (2.25)$$

Dabei gilt:

- \mathcal{L} ist die gesamte Verlustfunktion.
- $\mathcal{L}(y_i, \hat{y}_i)$ ist die Verlustfunktion zwischen den vorhergesagten und wahren Werten.
- M ist die Anzahl der Gewichte.
- $|w_j|$ ist der Betrag der Gewichte.
- λ ist eine Konstante.

Die Summe vom Betrag von allen Gewichten des Modells wird zur Verlustfunktion addiert, wobei die Konstante λ ein Hyperparameter ist und sich entscheidet, wie sich diese Summe auf die Verlustfunktion auswirkt. Die Hauptidee der L1-Regulierung besteht darin, die Gewichte zu entfernen, die für die Beschreibung der Merkmale verantwortlich sind, die nicht wichtig sind oder keinen großen Einfluss auf die Bestimmung der Outputs haben.

2.4.2. L2-Regulierung

Bei Inputs mit vielen verwandten Merkmalen, die alle zur Vorhersage der Outputs beitragen, führt das versehentliche Entfernen einiger Gewichte jedoch dazu, dass das Modell schwieriger zu trainieren ist. Die L2-Regulierung, auch Ridge-Regression oder "weight decay," genannt, ist die Lösung für dieses Problem.

$$\mathcal{L} = \mathcal{L}(y_i, \hat{y}_i) + \lambda \sum_{j=1}^M w_j^2 \quad (2.26)$$

Anders als L1 addiert L2-Regulierung die Summe der Quadrate aller Gewichte zur Verlustfunktion (siehe Formel 2.26). Dadurch werden nicht nur die Werte der Gewichte nicht auf null gesetzt, sondern das Modell wird auch dazu ermutigt, Gewichte mit kleinen Werten zu wählen, wodurch das Training einfacher und stabiler wird. Kleine Gewichte machen das Modell außerdem weniger empfindlich gegenüber kleinen Änderungen in den Inputs, was für Daten, die das Modell noch nie zuvor gesehen hat, wirklich von Vorteil ist.

2.4.3. Dropout

Dropout ist eine Methode, die verhindert, dass das Modell zu sehr auf bestimmte Trainingsdaten spezialisiert wird [17]. Dieser Layer stört den Trainingsprozess, indem sie einige der Neuronen im Netzwerk auf null setzt und der Rest auf eine Skala skaliert. Formel 2.27 beschreibt, wie Dropout funktioniert.

$$\text{mask}_{ij} = \begin{cases} 0 & \text{with probability } p \\ \frac{1}{1-p} & \text{otherwise} \end{cases} \quad (2.27)$$

$$y = x \times \text{mask}$$

Dabei gilt:

- x ist Input.
- i und j sind die Zeilen- bzw. Spaltenindizes von mask .
- mask ist eine Matrix mit der gleichen Größe des Inputs.
- p ist Dropoutrate und auch ein Hyperparameter.
- \times ist Hadamard-Produkt.

In der obigen Formel wird die Anzahl der Nullwerte in der Maskenmatrix durch Stichprobeneziehung aus einer Bernoulli-Verteilung mit der Wahrscheinlichkeit p generiert [17]. Die restlichen Werte sind gleich $\frac{1}{1-p}$. Dropout wird nur während des Trainings verwendet und bei jeder Iteration ist der Output das Ergebnis zwischen dem Input und einer anderen zufälligen Maskmatrix.

Darüber hinaus reduziert diese Methode die Co-Adaption an das Modell und bringt auch Aggregationseffekte mit sich. Das heißt, durch die Deaktivierung einiger Neuronen verhindert Dropout, dass ein Neuron zu stark von anderen spezifischen Neuronen abhängig ist, und erstellt leichtere Versionen des Modells. Dieser Effekt trägt zu einer verbesserten Leistung durch Training bei und erzeugt die robusteste Version, bevor sie im Testdatensatz ausgewertet wird.

Dropout bringt auch Kostenvorteile mit sich. Je weniger Neuronen verfügbar sind, desto weniger Rechenzeit wird benötigt.

2.4.4. Batch-Normalisierung

Obwohl die Daten vor der Einspeisung ins Netz skaliert wurden, kann nicht garantiert werden, dass das Modell ordnungsgemäß funktioniert. Einer der möglichen Gründe ist, dass ein oder mehrere Neuronen im Netz Werte haben können, die viel größer sind als die anderen. Dies kann zu dem im Abschnitt 2.3.7 erwähnten Problem von der Gradientenexplosion führen. Die Lösung dieses Problems besteht darin, dass der Output eines oder mehrerer Layer im Netz mit einem Mittelwert von 0 und einer Varianz von 1 normalisiert wird [18]. Dieser Vorgang wird als Batch-Normalisierung (BN oder englisch batch normalization) bezeichnet und wird in der Formel 2.28 ausgedrückt.

$$y = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \gamma + \beta \quad (2.28)$$

Dabei gilt:

- $\text{E}[x]$ ist der Mittelwert des Inputs entlang der räumlichen Dimension und des Minibatches.
- $\text{Var}[x]$ ist die Varianz des Inputs entlang der räumlichen Dimension und des Minibatches.
- γ und β sind die erlernbaren Parameter.
- ϵ ist eine sehr kleine Konstante.

Batch-Normalisierung kann die Abhängigkeit von Parametern wie Gewichten und Bias von Anfangswerten verringern und das Netz weniger empfindlich gegenüber der Lernrate machen. Obwohl bei jeder Iteration mehr Rechenaufwand erforderlich ist, kann mit BN der Gradient verbessert werden, sodass das Modell schneller konvergiert. Gamma und Beta werden aktualisiert, um Konsistenz zwischen den in das Modell eingespeisten Teildatenmengen sicherzustellen. Das heißt, sie weisen ungefähr den gleichen Mittelwert und die gleiche Standardabweichung auf. Jede plötzliche Gewichtsänderung des vorherigen Layers wirkt sich so wenig wie möglich auf den nächsten Layer aus.

2.5. Kreuzvalidierung

Kreuzvalidierung (englisch cross-validation) ist eine häufig verwendete Technik beim maschinellen Lernen, die darauf abzielt, die Leistung eines Modells abzuschätzen, wenn es anhand eines Trainingsdatensatzes trainiert und anhand eines Testdatensatzes ausgewertet wird.

Für jeden Versuch wird zunächst ein Modell mit Hyperparametern ausgewählt. Um eine möglichst faire Bewertung abgeben zu können, werden die Trainingsdaten in k Falze (englisch folds) unterteilt. In jedem Splitt wird das Modell auf $(k-1)$ Datenstücken trainiert, diese dienen als Trainingsdaten, der Rest dient als Validierungsdaten [20]. Dieser

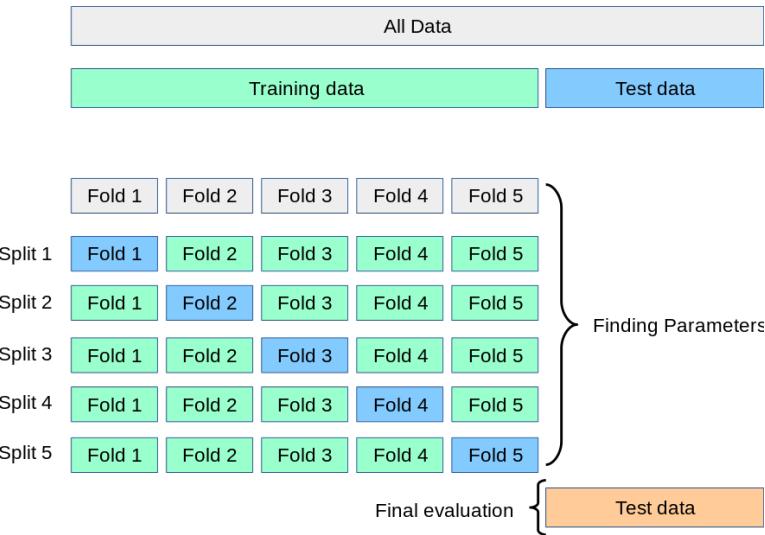


Abbildung 2.8.: Kreuzvalidierung [19]

Vorgang wird k mal durchgeführt. Es ist zu beachten, dass das Modell bei jedem Splitt mit zufälligen Parametern neu initialisiert und neu trainiert wird und eine vom Modell auf Validierungsdaten auswertende Leistungsmetrik wie Verlust oder Genauigkeit liefert. Die Leistung dieses Versuchs entspricht dem Durchschnitt der Leistungsmetriken der gesamten Splitte. Abbildung 2.8 beschreibt, wie Kreuzvalidierung funktioniert. Durch viele Versuche mit unterschiedlichen Architekturen der Modelle und Hyperparametern wird ein bestes Modell mit der höchsten Leistung ausgewählt. k fungiert als Hyperparameter, in der Praxis wird k üblicherweise zwischen 5 und 10 gewählt. Die Kreuzvalidierung ist auch nützlich, um die am besten passenden Hyperparameter zu finden und Overfitting zu vermeiden.

2.6. Transfer Learning

Transfer Learning ist eine Technik des maschinellen Lernens, die darauf abzielt, Wissen und Verständnis aus einem Problem für einen anderen spezifischen Zweck zu nutzen, um die Effizienz und Effektivität zu steigern. In diesem Kapitel werden einige gängige Methoden erwähnt.

2.6.1. Fine-Tuning

In der Praxis ist es nicht einfach und sehr teuer, ein Modell zu erstellen, Parameter auszuwählen und zu testen. Außerdem sind eine große Menge an Daten und alle Labels auf den Trainings- und Testdatensätzen erforderlich, um diese Modelle zu trainieren. Für Deep Learning besteht einer der beliebtesten Ansätze darin, dass künstliche neuronale

Netze, die auf großen Datensätzen trainiert wurden, auf einem spezialisierteren, kleineren Datensatz verfeinert und weiter trainiert werden. Dieser Vorgang wird als Fine-Tuning bezeichnet.

Die Grundidee dieser Methode besteht darin, dass die Struktur des Netzes und die erlernten Gewichte genutzt werden [21]. Diese Gewichte haben die Fähigkeit, die Merkmale wie Kanten oder Formen von den Bildern zu erkennen, indem sie vorher trainiert wurden, und erzielen gute Ergebnisse. Für eine andere bestimmte Aufgabe wird nur der letzte Layer bearbeitet, normalerweise fully-connected Layer, um dem erforderlichen Output zu entsprechen. Das heißt, nur die Parameter im verfeinerten Layer sind für das Lernen verantwortlich und werden durch Gradient Decent und Backpropagation aktualisiert, um den Output zu bestimmen. Die Parameter in den vorherigen Layern, wie zum Beispiel die faltenden Layer, tragen nur zur Berechnung bei und ihre Gewichte und Bias werden nicht geändert.

Fine-Tuning ist nicht ohne Schwächen. Bei komplexeren Aufgaben wird nicht nur der letzte fully-connected Layer geändert, manchmal müssen auch die davor liegenden faltenden Layer neu trainiert werden und auch die Struktur der verfeinerten Layer ist eine Herausforderung. Wenn die Zieldatendomäne nicht die gleiche Verteilung wie die ursprüngliche Datendomäne aufweist, kann Fine-Tuning seine Wirkung nicht entfalten. Fine-Tuning wird üblicherweise auf Deep Vision Networks zur Bildklassifizierung und Modelle für Probleme bei der Verarbeitung natürlicher Sprache (englisch natural language processing) angewendet.

3. Ansatzdetails

In diesem Kapitel werden einige spezielle Methoden erläutert und in dieser Arbeit verwendet.

3.1. Fensterung

Der Input der RUL-Bestimmungsaufgabe sind die Zeitreihen, die eine oder mehrere Maschineneigenschaften beschreiben. Beim überwachten Lernen werden zum Trainieren eines Modells viele Daten mit seinen Labels benötigt. Dazu wird die Zeitreihe in mehrere, sich möglicherweise überlappende Fragmente aufgeteilt. Dieser Prozess in der Signalanalyse wird Fensterung (englisch windowing) genannt. Das in dieser Arbeit verwendete Fenster ist ein rechteckiges Fenster mit einem Wertebereich von negativ bis positiv unendlich, das heißt, jeder im Segment enthaltene Datenpunkt aus der Zeitreihe bleibt erhalten. Jedes Segment aus der Zeitreihe hat sein eigenes Label, in diesem Fall den Zeitraum zwischen dem Zeitstempel des letzten Datenpunktes im Segment und Lebensende (EOL). Formel 3.1 beschreibt die allgemeine Form jedes Fragments und seines Labels.

$$\begin{aligned} X_i &= \{x_{i \cdot H+0}, x_{i \cdot H+1}, \dots, x_{i \cdot H+W}\} \quad \forall i \in [0, \frac{N-W}{H}] \\ Y_i &= EOL - \text{timestamp}(x_{i \cdot H+W}) \end{aligned} \tag{3.1}$$

Dabei gilt:

- i ist der Index des Segmentes.
- N ist die Anzahl der Datenpunkte in einer Zeitreihe.
- W ist die Länge eines Fensters, beschreibt die Anzahl der Datenpunkte in einem Segment.
- H ist die Verschiebung zwischen zwei Segmenten und $W - H$ ist die Anzahl der überlappenden Datenpunkte.
- $\frac{N-W}{H} + 1$ ist die Anzahl der aus einer Zeitreihe extrahierten Segmente.
- $\text{timestamp}(x_{i \cdot H+W})$ ist der Zeitpunkt, zu dem der letzte Datenpunkt im Segemnt erfasst wurde.

Zur Vereinfachung der Interpretation bezieht sich in dieser Arbeit „Probe“ auf ein Segment mit seinem Label, und „Datenpunkt“ bezeichnet einen im Segment oder in der Zeitreihe enthaltenen Wert.

3.2. Piecewise Aggregate Approximation

Piecewise Aggregate Approximation (PAA) ist eine Technik, die bei der Analyse und Verarbeitung von Zeitreihen angewendet wird. Die Hauptidee dieser Methode besteht darin, die Zeitreihe durch Reduzierung der Anzahl der Punkte zu komprimieren, sodass die grundlegenden Eigenschaften und Merkmale der ursprünglichen Zeitreihe erhalten bleiben. Die Zeitreihe wird zunächst in Fragmente unterteilt, die sich möglicherweise überlappen und aus einer bestimmten Anzahl von Datenpunkten bestehen. Anschließend wird der Durchschnitt der Werte der Datenpunkte in jedem Segment berechnet [22]. Die komprimierte durch PAA Zeitreihe ohne Überlappung kann durch Formel 3.2 dargestellt.

$$x'_i = \frac{1}{W} \sum_{j=0}^{W-1} x_{i \cdot W + j} \quad \forall i \in [0, \frac{N}{W} - 1] \quad (3.2)$$

Dabei gilt:

- x'_i ist der i -te Datenpunkt der komprimierten Zeitreihe der Länge $\frac{N}{W}$
- W ist die Segmentlänge.
- j ist der Index der Datenpunktes in einem Segment.

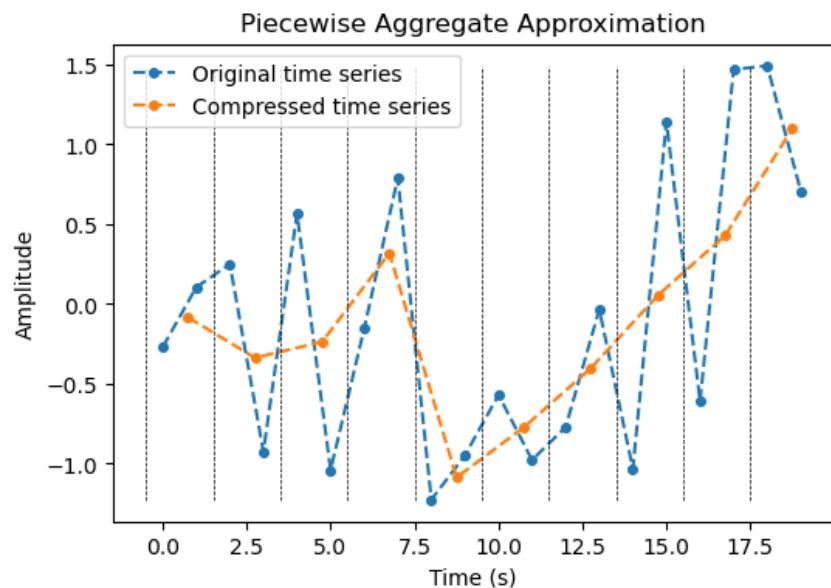


Abbildung 3.1.: Piecewise Aggregate Approximation

Abbildung 3.1 ist ein Beispiel einer Zeitreihe, die aus einer ursprünglichen Länge von 20 und einer Segmentlänge von 2 ohne Überlappung komprimiert wird. PAA wird häufig in Datenvorverarbeitungsschritten angewendet. Sie trägt dazu bei, komplexe Rechenschritte bei der Konvertierung zu Bildern und in Modellen des maschinellen Lernens zu reduzieren. Darüber hinaus spielt PAA auch eine Rolle bei der Rauschunterdrückung, allerdings ist der Verlust von Informationen unvermeidlich.

3.3. Zeitreihen-zu-Bild-Encoding

Die Rohdaten dieses Projekts liegen in Form von Zeitreihen vor, und nachdem sie durch die grundlegenden Schritte in Abschnitt 2.2 vorverarbeitet wurden, werden die Zeitreihen weiter in Bilder umgewandelt, um die Verarbeitungsfähigkeit von CNNs zu nutzen. In diesem Abschnitt werden verschiedene Methoden zur Zeitreihenkonvertierung sowie deren Verwendung vorgestellt.

3.3.1. Kurzzeit-Fourier-Transformation

Bei Signalverarbeitungsaufgaben wird die diskrete Fourier-Transformation (DFT) verwendet, um Signale im Zeitbereich in den Frequenzbereich umzuwandeln [23]. Mit anderen Worten hilft sie zu wissen, wie viele Frequenzen ein Signal enthält. Die DFT geht jedoch davon aus, dass die Komponentenfrequenzen im Signal in der ganzen Zeit gleichzeitig auftreten, das heißt, die DFT hat keine Informationen darüber, in welchem Zeitintervall welche Frequenzkomponenten vorhanden sind. Dies ist in der Praxis sehr unpraktisch, da Signale häufig verrauscht sind und sich die Frequenzen im Laufe der Zeit ändern. Ein Beispiel für dieses Problem besteht darin, dass Signale, die nicht gleich aussehen, dieselbe DFT-Darstellung haben können, wenn sie dieselben Frequenzkomponenten enthalten (siehe Abbildung 3.2). Formel 3.3 beschreibt die Berechnung von DFT. Die maximale Frequenzkomponente, die DFT gemäß dem Nyquist-Theorem darstellen kann, beträgt die Hälfte der Abtastfrequenz.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-i2\pi kn/N} \quad (3.3)$$

Dabei gilt:

- $X(k)$ ist der k-te Wert im Frequenzbereich
- $x(n)$ ist der n-te Datenpunkt im Zeitbereich
- N ist die Anzahl der Datenpunkte.
- i ist die imaginäre Einheit

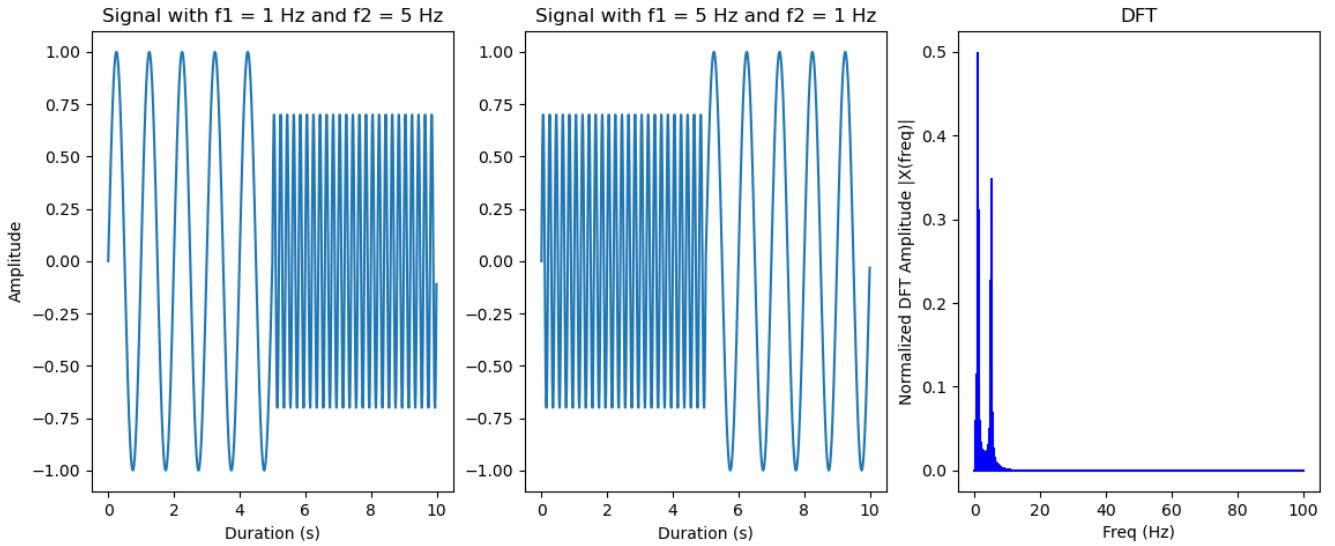


Abbildung 3.2.: Zwei unterschiedliche Signale mit den gleichen Frequenzkomponenten haben gleiche DFT-Darstellung

Um dieses Problem zu lösen, ist Kurzzeit-Fourier-Transformation (STFT oder englisch short-time Fourier transform) in der Lage, Signale sowohl im Zeitbereich als auch im Frequenzbereich darzustellen. Das Signal wird zunächst in gleiche Segmente unterteilt, die überlappt werden können. In jedem dieser Segmente wird die Fenstergleichung beispielsweise Hamming oder Hann unter der Verwendung der Hadamard-Produktoperation angewendet, um spektrale Leckagen zu vermeiden. Anschließend wird die DFT unabhängig für jedes Fenstersegment berechnet, um die darin enthaltenen Frequenzkomponenten zu bestimmen [24]. Der Output der STFT ist eine Matrix namens Spektrogramm, wobei die x-Achse die Zeitinformationen, die y-Achse die Frequenzinformationen und die Farbin-tensität die Magnitude der entsprechenden Frequenz zu einer bestimmten Zeit darstellt. STFT verfügt über Hyperparameter, die sind:

- Fenstergröße (englisch window size) ist die Länge eines Segments, also die Anzahl der Datenpunkte in einem Segment.
- Hopfengröße (englisch hop size) ist die Schrittweite des aktuellen Segments, das auf dem Signal bis zum nächsten Segment verschoben wird. Der Differenz zwischen Fenstergröße und Hopfengröße ist auch die Anzahl der überlappenden Datenpunkte zwischen zwei benachbarten Segmenten.

Diese Hyperparameter wirken sich direkt auf die Ausgabe von STFT aus, also auf das Spektrum des Signals. Zunächst müssen jedoch zwei Konzepte geklärt werden:

- Die Zeitauflösung beschreibt, wie genau STFT den Zeitpunkt bestimmt, zu dem sich die Frequenzen im Signal ändern.

- Die Frequenzauflösung beschreibt, wie genau STFT verschiedene Frequenzen innerhalb eines festen Zeitraums im Signal identifiziert.

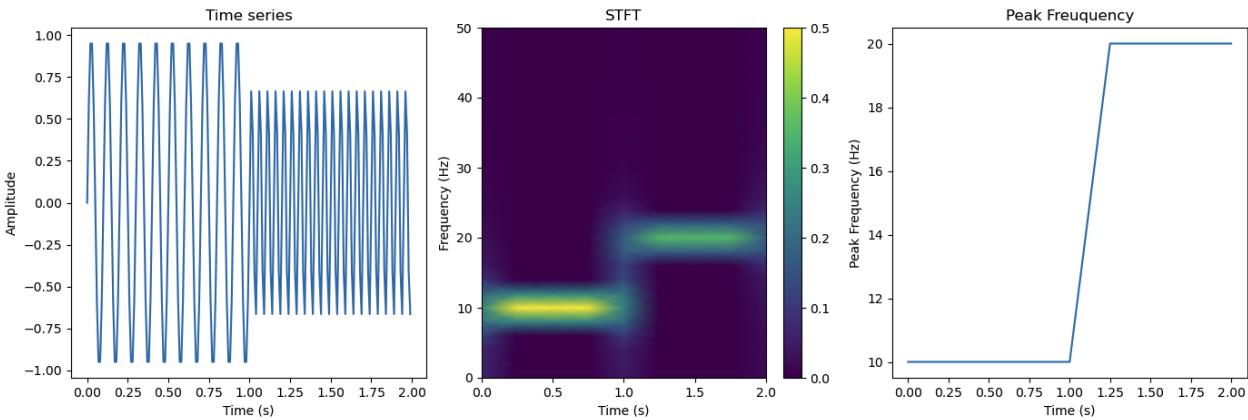


Abbildung 3.3.: Kurzzeit-Fourier-Transformation

Eine Vergrößerung des Fensters führt zu einer besseren Frequenzauflösung, da die Länge des Signals in einem größeren Segment analysiert wird, wodurch nahe beieinander liegende Frequenzen genauer unterschieden werden können. Allerdings verringert sich dadurch die Zeitauflösung. Eine Verringerung der Fenstergröße hat den gegenteiligen Effekt. Es wird immer einen Kompromiss zwischen diesen beiden Auflösungen geben. Durch Erhöhen der Hopfengröße werden die Überlappung zwischen zwei benachbarten Segmente und die Anzahl der Segmente verringert, was die Zeitauflösung, aber auch den Rechenaufwand verringert. Eine Reduzierung der Sprunggröße hat den gegenteiligen Effekt. Die Ausgabe von STFT ist ein Spektrum und auch ein Bild der Größe $\langle window_size/2 + 1, (n_samples - window_size)//hop_size + 1 \rangle$, wobei $window_size//2 + 1$ ist die Anzahl der Frequenz-Bins, die bis zu der maximalen Frequenz im Frequenzbereich darstellen können, die der halben Abtastfrequenz entspricht, die der halben Abtastfrequenz entspricht. $(n_samples - window_size)//hop_size + 1$ ist die Anzahl der Segmente im Zeitbereich. Diskrete STFT wird durch die Formel 3.4 ausgedrückt [24].

$$X(\omega, m) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \quad (3.4)$$

Dabei gilt:

- m ist der Index des Segments im Zeitbereich und liegt zwischen 1 und $((n_samples - window_size)//hop_size + 1)$
- ω ist der Index des Frequenz-Bins im Frequenzbereich und liegt zwischen 1 und $(window_size//2 + 1)$
- $X(m, \omega)$ ist der STFT-Koeffizient im m -ten Segment und im ω -ten Frequenz-Bin.
- $x[n]$ ist das ursprüngliche diskrete Signal, bestehend aus n Abtastwerten.

- $w[n - m]$ ist die Fenstergleichung, bestehend aus n Abtastwerten, die gemäß dem Hadamard-Produkt an Position m im Zeitbereich mit $x[n]$ angewendet wird.

Es ist zu beachten, dass Formel 3.4 den Term $e^{-j\omega n}$ enthält, sodass jeder Wert im STFT-Spektrum eine komplexe Zahl ist. Zur Visualisierung oder Einbeziehung in das Modell zur Berechnung werden ihre Absolutwerte berechnet. Ein Beispiel für eine STFT aus einem Sinussignal, das aus zwei verschiedenen Frequenzen und Amplituden besteht, ist in Abbildung 3.3 dargestellt. Aus der Abbildung lassen sich die Zeitpunkte des Auftretens der Frequenzen ablesen. Für jede Zeitreihe gibt es nur ein einzigartiges STFT-Spektrum. Es wird immer empfohlen, die Hyperparameter mehrmals auszuprobieren, um das beste Spektrum auswählen zu können, aus dem das Modell die Eigenschaften des Signals am besten lernen kann. STFT wird häufig in der Sprach- und Audioanalyse und -verarbeitung eingesetzt.

3.3.2. Log-Mel-Spektrogramm

Zunächst muss die Mel-Skala geklärt werden. Das menschliche Ohr reagiert empfindlicher auf niedrige Frequenzen und weniger empfindlich auf hohe Frequenzen. Beispielsweise kann das menschliche Ohr leicht zwischen zwei Geräuschen mit Frequenzen von 100 Hz und 200 Hz unterscheiden, hat jedoch Schwierigkeiten bei der Erkennung der Differenz zwischen zwei Geräuschen mit 2000 Hz und 2100 Hz. Das Ohr ist nicht in der Lage, lineare Frequenzänderungen wahrzunehmen [25]. Stattdessen kann der Mensch diese Änderung besser auf einer logarithmischen Skala abschätzen, die durch die Mel-Skala dargestellt wird, die die Empfindlichkeit des menschlichen Ohrs gegenüber Frequenzänderungen genau misst. Die Beziehung zwischen Frequenz und Mel wird durch Formeln 3.5 und 3.6 dargestellt, wobei m und f jeweils Mel und die Frequenz sind [26].

$$m(f) = 2595 \times \log_{10}\left(1 + \frac{f}{700}\right) \quad (3.5)$$

$$f(m) = 700 \times \left(10^{\frac{m}{2595}} - 1\right) \quad (3.6)$$

Mel-Spektrogramm (MS oder englisch mel-spectrogram) kann Signale sowohl im Zeit- als auch im Frequenzbereich interpretieren und wird selbst aus der STFT berechnet. Der Hauptunterschied besteht jedoch darin, dass das MS die Darstellungen von Frequenzen basierend auf der Mel-Skala an das Beste des menschlichen Ohrs anpassen kann. Diese Aufgabe übernimmt die Filterbank. Die Filterbank akzeptiert einen Parameter namens n_mels , der die Anzahl der Dreiecksfilter angibt, die benachbarte Frequenzen in entsprechende Mel-Frequenz-Bins gruppieren. Zunächst wird das Intervall zwischen f_{min} und f_{max} auf der Mel-Skala in $n_mels + 2$ gleiche Teile unterteilt. Dann wird jeder Dreiecksfilter durch drei Frequenzen auf der vorab benachbarten Mel-Skala erzeugt. Das bedeutet, dass die Intensität der Frequenz in der Mitte vom Filter mit dem höchsten Wert erfasst wird. Darüber hinaus hat der Filter auch die Funktion, die Intensität niedriger Frequenzen zu erhöhen und die Intensität hoher Frequenzen zu verringern, was der Empfindlichkeit des menschlichen Ohrs entspricht. Auch die Dreiecksfilter

werden flächennormalisiert, das heißt, sie haben den gleichen Flächeninhalt. Formel 3.7 beschreibt die Berechnung der Filterbank für den Fall, dass f_{min} gleich 0 und f_{max} gleich der Nyquist-Frequenz ist. Abbildung 3.4 links zeigt die 1D-Darstellung der Filterbank mit n_mels gleich 8.

$$H(n, \omega) = \begin{cases} 0 & \text{if } f_\omega < f_{n-1} \\ \frac{f_\omega - f_{n-1}}{f_n - f_{n-1}} \cdot norm & \text{if } f_{n-1} \leq f_\omega < f_n \\ \frac{f_{n+1} - f_\omega}{f_{n+1} - f_n} \cdot norm & \text{if } f_n \leq f_\omega < f_{n+1} \\ 0 & \text{if } f_\omega \geq f_{n+1} \end{cases} \quad (3.7)$$

$$norm = \frac{2 \cdot window_size}{sr \cdot (f_{n+1} - f_{n-1})}$$

Dabei gilt:

- n ist der Index des Dreieckfilters also Mel-Frequenz-Bins und liegt zwischen 1 und n_mels .
- f_{n-1} , f_n und f_{n+1} sind jeweils linke, mittlere und rechte Frequenz des n -ten Filters mit $f_{n-1} < f_n < f_{n+1}$. Diese drei Frequenzen auf der Mel-Skala haben den gleichen Abstand.
- ω ist der Index der Frequenz-Bins im Frequenzbereich zwischen f_{min} und f_{max} und ihre Werte werden durch die Triangulationsfilter beeinflusst. ω liegt in diesem Fall zwischen 1 und $window_size//2 + 1$.
- f_ω ist die Frequenz im entsprechenden ω -ten Frequenz-Bin.
- $norm$ ist die hinzugefügte Normalisierungskomponente, um sicherzustellen, dass die Dreiecksfilter gleiche Flächen haben.
- sr ist die Abtastfrequenz.
- $window_size$ ist die in STFT verwendete Segmentlänge.

Tatsächlich ist die Filterbank im Berechnungsprozess eine Matrix der Form $\langle n_mels, window_size//2 + 1 \rangle$, wie in Abbildung 3.4 rechts dargestellt. Anschließend wird sie mit einem STFT-Spektrum der Form $\langle window_size//2 + 1, (n_samples - window_size)//hop_size + 1 \rangle$ durch Matrixmultiplikation angewendet. MS ist schließlich eine Matrix der Form $\langle n_mels, (n_samples - window_size)//hop_size + 1 \rangle$ und die Berechnung kann durch Formel 3.8 dargestellt werden.

$$MS(n, m) = \sum_{\omega=1}^{window_size//2+1} H(n, \omega) \cdot |X(\omega, m)|^2 \quad (3.8)$$

$$LMS(n, m) = 10 \log_{10} \left(\frac{\max(MS(n, m), \epsilon)}{ref} \right) \quad (3.9)$$

Dabei gilt:

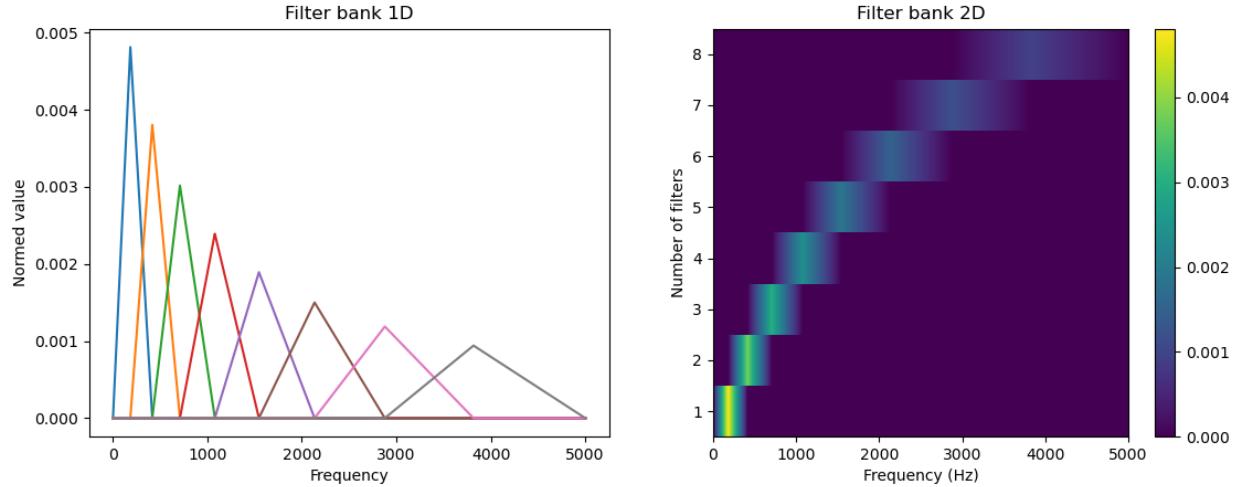


Abbildung 3.4.: Filterbank in 1D- und 2D-Darstellung

- $MS(n, m)$ ist der MS-Koeffizient im n -ten Mel-Frequenz-Bin und im m -ten Segment.
- $H(n, \omega)$ ist der Filterbank-Koeffizient im n -ten Mel-Frequenz-Bin und im ω -ten Frequenz-Bin.
- $|X(\omega, m)|^2$ ist das Quadrat des Absolutwerts vom STFT-Koeffizient im ω -ten Frequenz-Bin und im m -ten Segment.
- ϵ ist eine sehr kleine positive Zahl, die hinzugefügt wird, um den Logarithmus von 0 zu vermeiden.
- ref ist der Referenzwert.

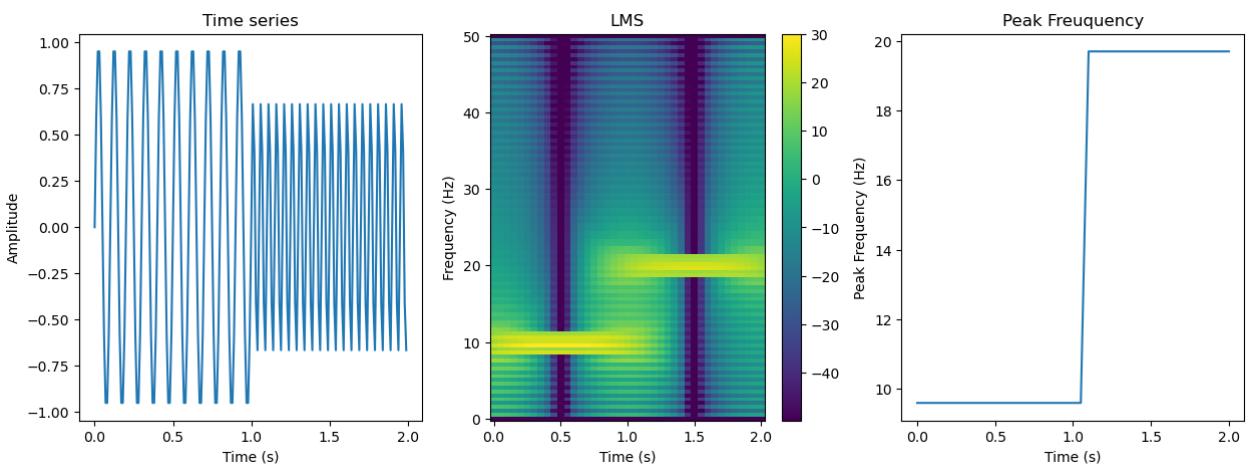


Abbildung 3.5.: Log-Mel-Spekrogramm

Beim Log-Mel-Spekrogramm (LMS) werden die Koeffizienten im Mel-Spekrogramm in Dezibel (dB) wie in der Formel 3.9 umgewandelt. Dies liegt daran, dass die Dezibel-Skala den Wertebereich komprimiert und die Fähigkeit zur Visualisierung niedriger Werte verbessert, sodass diese im Vergleich zu großen Werten nicht unterdrückt werden. Darüber hinaus spiegelt die Dezibel-Skala auch besser wider, wie Menschen Schall wahrnehmen. Ein Beispiel für LMS aus einem Sinussignal, das aus zwei verschiedenen Frequenzen und Amplituden besteht, ist in Abbildung 3.5 dargestellt, aus dem sich die Zeitpunkte des Auftretens der Frequenzen ablesen. Wie STFT verfügt auch LMS über die beiden Parameter *window_size* und *hop_size* und noch den Parameter *n_mels* der Filterbank hinzu. Die Auswahl der richtigen Parameter stellt immer eine Herausforderung dar, daher empfiehlt es sich immer, es mehrmals zu versuchen, um gute Ergebnisse zu erzielen. LMS wird häufig in der Audioverarbeitungsanalyse, Musik, Spracherkennung und -analyse eingesetzt.

3.3.3. Kontinuierliche Wavelet-Transformation

Ähnlich wie STFT und LMS ist auch kontinuierliche Wavelet-Transformation (CWT oder englisch continuous wavelet transform) in der Lage, Signale sowohl im Zeitbereich als auch im Frequenzbereich zu interpretieren. Wenn jedoch in den oben genannten beiden Methoden DFT die Funktion hat, die im Signal in jedem Segment vorhandenen Frequenzen zu bestimmen, dann wird diese Aufgabe für CWT dem Mutter-Wavelet (englisch mother wavelet) zugewiesen oder als Wavelet-Funktion (WF) bezeichnet. WF muss einen Mittelwert von null und eine endliche Energie haben [27]. Diese beiden Eigenschaften werden durch Formel 3.10 dargestellt, wobei $w(t)$ die WF ist. Einige WF zeigt Abbildung 3.6. Um die Frequenzen analysieren zu können, wird eine 1D-Faltung zwischen dem WF und dem Signal angewendet, wobei die Länge des Kernels also der WF gleich der Länge des Signals und die Schrittweite gleich eins ist.

$$\begin{aligned} \int_{-\infty}^{\infty} w(t) dt &= 0 \\ \int_{-\infty}^{\infty} |w(t)| dt &= 0 \end{aligned} \tag{3.10}$$

Wie in Abbildung 3.6 zu sehen ist, hat jede WF eine eigene Frequenz, die mit f_c bezeichnet wird, und die Aufgabe der Faltung besteht in diesem Fall darin, Informationen darüber bereitzustellen, ob eine Frequenzübereinstimmung zwischen dem WF und dem Signal besteht. Formel 3.11 beschreibt, wie 1D-Faltung berechnet wird [28].

$$(x * h)[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] \tag{3.11}$$

Dabei gilt:

- $x[k]$ ist der k -te Wert des Signals.

- $h[n-k]$ ist der k -te Wert des umgedrehten Filters, der um n nach rechts verschoben ist.
- $(x * h)[n]$ ist der n -te Wert der Faltung zwischen dem Signal und dem Filter.

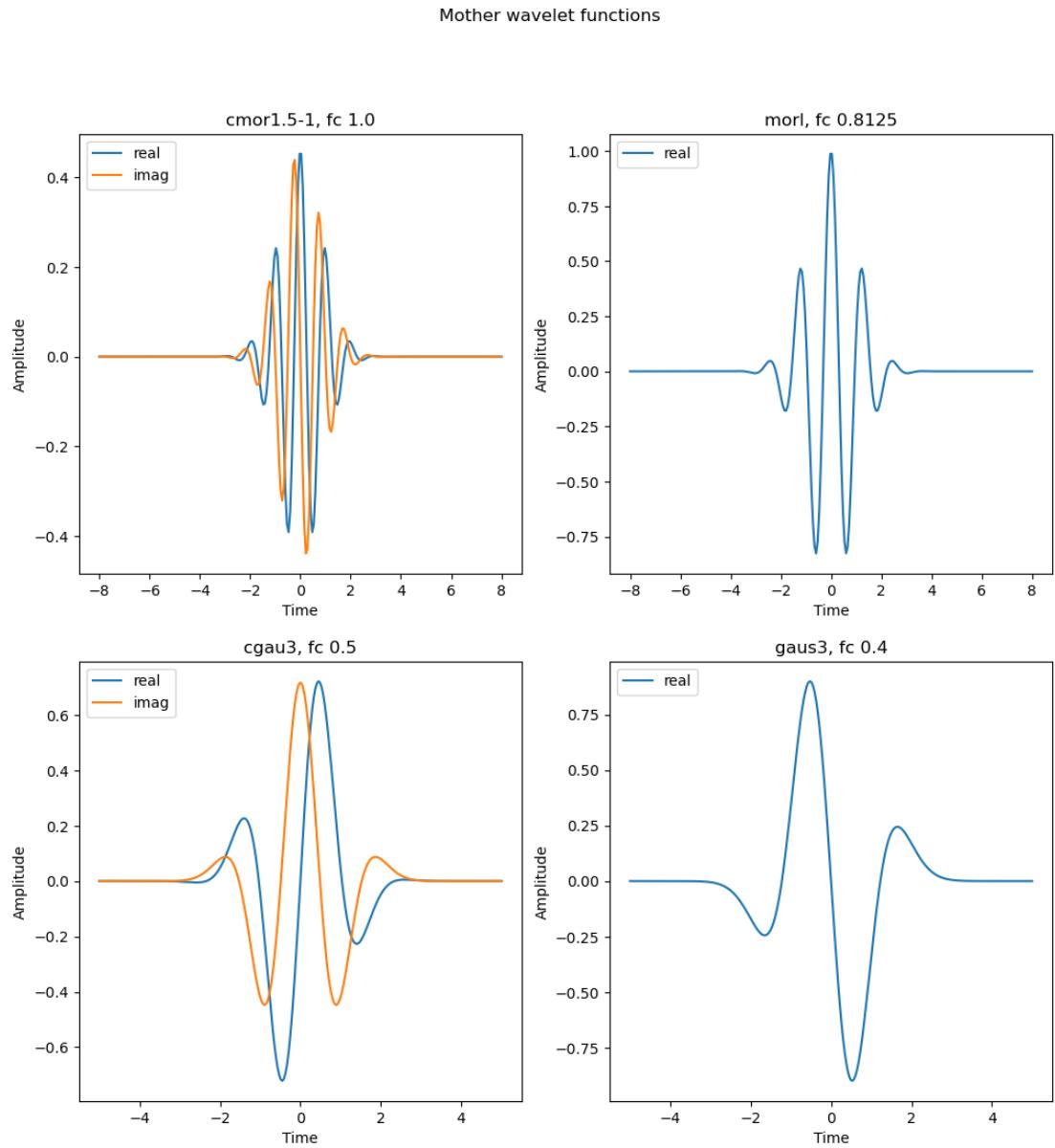


Abbildung 3.6.: Wavelet-Funktionen

Die Faltung ist null, wenn das Signal nicht die gleiche Frequenz wie die Frequenz von WF hat. Wie in Abbildung 3.7 oben gezeigt, erkennt zum Zeitpunkt die WF namens Morlet eine Übereinstimmung und die Faltung nun eine Sinuswellenform hat. Allerdings gibt es hier ein kleines Problem, da diese Sinuswellen Nullwerte enthalten, wenn der Phasenwinkel zwischen den beiden an dieser Stelle 90 Grad beträgt und sich ihre überlappenden

Bereiche gegenseitig aufheben. Dies ist sowohl für Menschen als auch für Modelle des maschinellen Lernens ein Hindernis bei der Unterscheidung zwischen diesen Nullen, die die Frequenzübereinstimmung oder -differenz des Signals und der WF darstellen. Eine Lösung für dieses Problem ist die Verwendung komplexer Wavelet-Funktionen. Auf die Formeln 3.12 und 3.13 wird in [29] verwiesen.

$$\psi_{\text{morl}}(t) = e^{-t^2/2} \cdot \cos(5t) \quad (3.12)$$

$$\psi_{\text{cmorl}}(t) = \frac{1}{\sqrt{\pi B}} \cdot e^{-t^2/B} \cdot e^{j2\pi C t} \quad (3.13)$$

Dabei gilt:

- $\psi_{\text{morl}}(t)$ ist Morlet-Funktion
- $\psi_{\text{cmorl}}(t)$ ist komplexe Morlet-Funktion
- B ist die Bandbreite, ein Hyperparameter, der sich einscheidet, wie breit der Teil der WF ist, dessen Energie größer als Null ist.
- C oder auch f_c genannt ist auch ein Hyperparameter, der die zentrale Frequenz oder Standardfrequenz der WF darstellt.

Die Formel 3.13 enthält $e^{j2\pi C t}$, was die eulersche Darstellung der komplexen Zahl ist, die als Formel 3.14 umgeschrieben werden kann.

$$e^{i\theta} = \cos(\theta) + i \sin(\theta) \quad (3.14)$$

Das Endergebnis wird nun die Faltung sowohl des Real- als auch des Imaginärteils des WF mit dem Signal sein. Der Beitrag der Frequenzkomponenten zu jedem Zeitpunkt wird jetzt als absoluter Wert sowohl der Faltung des Imaginär- als auch des Realteils berechnet. Wie bereits erwähnt, sind Nullwerte der Faltung auf eine 90-Grad-Phasendifferenz zwischen dem Realteil und Signal zurückzuführen, die bei Vorhandensein des Imaginärteils vollständig verschwindet, da der Imaginärteil nun in Phase mit dem Signal ist, und ihre absoluten Werte ergeben positive Werte. Es kann eindeutig angegeben werden, weil der Imaginärteil die Sinusfunktion und der Realteil die Kosinusfunktion enthält (siehe Formel 3.14), dass sie offensichtlich immer um 90 Grad phasenverschoben zueinander. Wenn sich die komplexe Wavelet-Funktion wie in Abbildung 3.7 unten anwenden lassen, werden negative Werte und Nullwerte bei der Faltung vollständig eliminiert, falls eine Übereinstimmung zwischen der Frequenz der WF und dem Signal besteht. Dies wird Modellen helfen, die Eigenschaften des Signals besser zu verstehen. Der Nachteil wäre jedoch, dass sich die Berechnungskosten verdoppeln würde.

Tatsächlich ist in einem Signal mehr als eine Frequenz vorhanden, und viele Faltungen mit unterschiedlichen Frequenzen der WF durchgeführt. Daher wird es eine Größe geben, um die Frequenz der WF zu ändern, nämlich die Skala (englisch scale). Skala fungiert in CWT als vom Benutzer wählbarer Hyperparameter und ändert f_c von WF, auch bekannt als die Standard- oder Basisfrequenz. Die Beziehung zwischen Skala und f_c wird durch Formel 3.15 dargestellt. Basierend auf der Formel 3.15 ist ersichtlich, dass bei

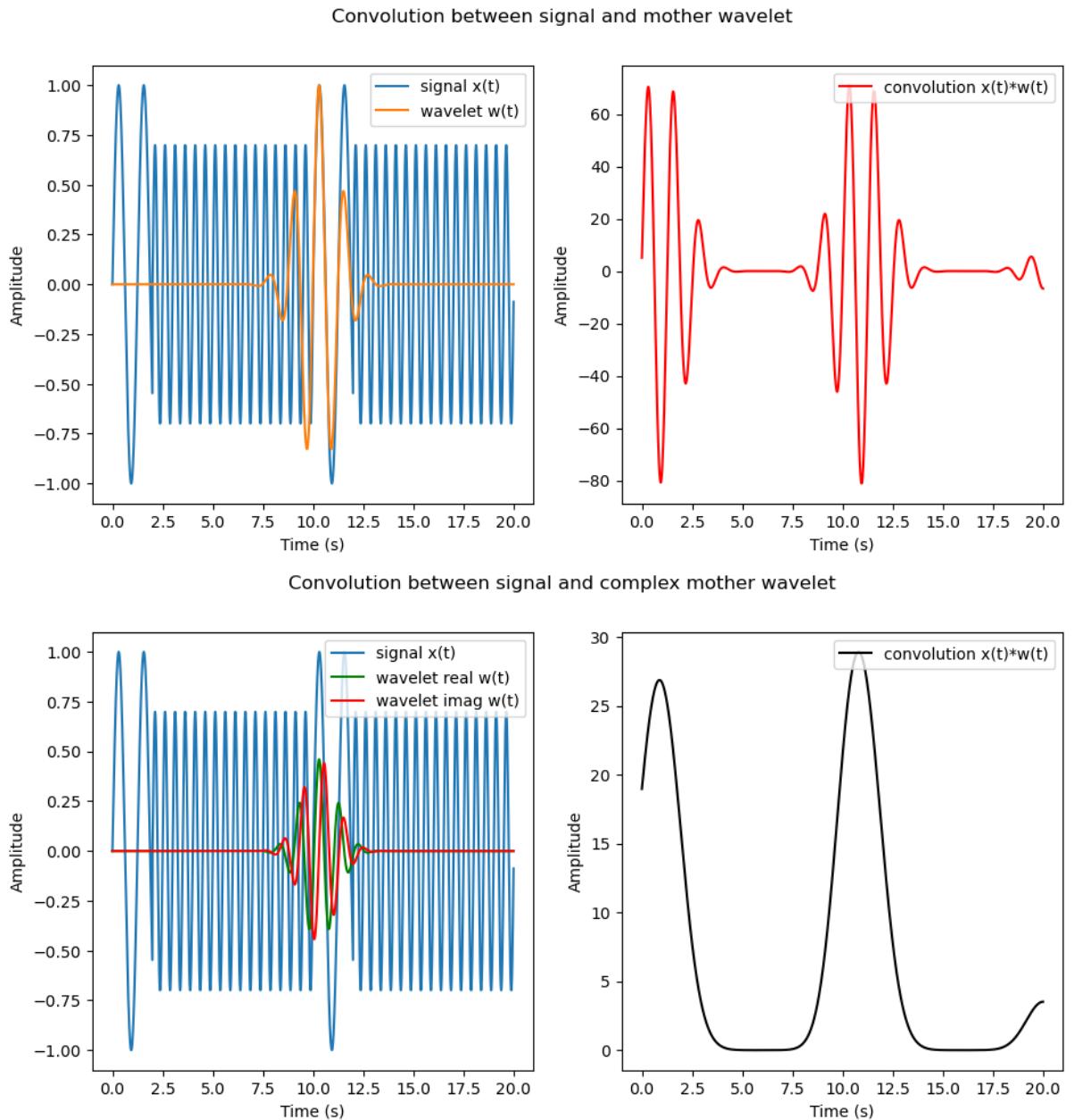


Abbildung 3.7.: 1D-Faltung zwsichen dem Signal und der Wavelet-Funktion

einer Erhöhung der Skala der WF gedeht wird, während eine Verringerung der Skala zu einer Komprimierung der WF führt [27].

$$f = \frac{f_c}{scale \cdot si} \quad (3.15)$$

Dabei gilt:

- f_c ist die Standardfrequenz von der WF.

- f ist die Frequenz, nach der die WF im Signal suchen möchte.
- si ist die Abtastperiode und entspricht dem Kehrwert der Abtastfrequenz.
- $scale$ ist Skala, ein positiver Hyperparameter.

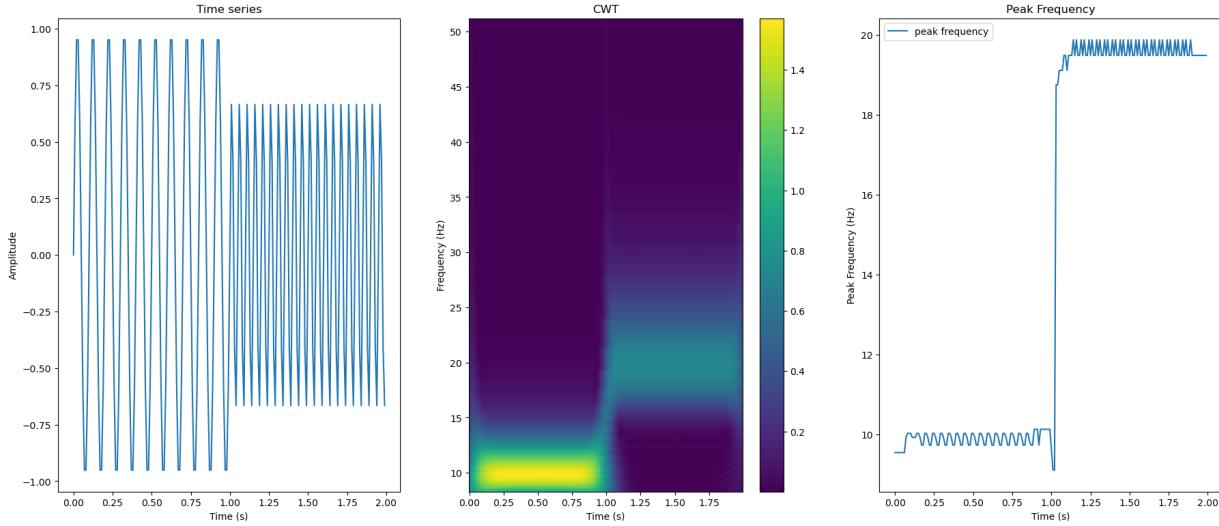


Abbildung 3.8.: Kontinuierliche Wavelet-Transformation

Die Ausgabe des CWT ist eine Matrix der Form $\langle n_scales, n_samples \rangle$, wobei n_scales die Anzahl der Skalen ist und $n_samples$ der Länge des Signals entspricht. Jede Zeile der Matrix entspricht der Faltung zwischen dem Signal und WF auf einer bestimmten Skala. Jeder CWT-Koeffizient wird durch Formel 3.16 erklärt [27].

$$X(a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} x(t) \cdot \bar{\psi}\left(\frac{t-b}{a}\right) dt \quad (3.16)$$

Dabei gilt:

- $X(a, b)$ ist der b -te Koeffizient in der Faltung zwischen der Wavelet-Funktion und dem Signal auf der Skala a .
- $x(t)$ ist das zu analysierende Signal.
- $\bar{\psi}\left(\frac{t-b}{a}\right)$ ist eine komplexe konjugierte Wavelet-Funktion, die um b nach rechts verschoben ist und deren Frequenz um a skaliert ist.

Obwohl gemäß Formel 3.16 die vertikale Achse der Skala entspricht, zeigt Abbildung 3.8 zur Vereinfachung der Visualisierung das CWT-Spektrum, das das Signal mit zwei verschiedenen Frequenzen und Amplituden im Zeit- und Frequenzbereich mithilfe von der komplexen Wavelet-Funktion Morlet wie in Formel 3.13 darstellt. Es ist zu beachten, dass Formel 3.15 die Beziehung zwischen Skalen und Frequenz darstellt. Wenn die Frequenzen, die bei STFT und LMS analysiert werden können, im Bereich von 0 bis der Hälfte der Abtastfrequenz beschränkt sind, können die verschiedenen Skalen bei CWT

ausgewählt werden, um zu entscheiden, welche Frequenzen wahrscheinlich im Signal vorhanden sind. Im Vergleich zu den beiden oben genannten Methoden verfügt CWT auch über mehr Hyperparameter.

3.3.4. Rekurrenzplot

Anstatt wie bei den drei oben genannten Methoden die Frequenzen im Signal zu analysieren, konzentriert sich Recurrenzplot (RP oder englisch recurrence plot) darauf, den Abstand zwischen den Punkten anzuzeigen. Die Formel von RP wird durch Formel 3.17 dargestellt [30].

$$R(i, j) = \Theta(\epsilon - \|x_i - x_j\|) \quad (3.17)$$

$$\begin{aligned} x_i &= (x_i, x_{i+\tau}, \dots, x_{i+(m-1)\tau}) \quad \forall i \in \{1, \dots, n - (m-1)\tau\} \\ x_j &= (x_j, x_{j+\tau}, \dots, x_{j+(m-1)\tau}) \quad \forall j \in \{1, \dots, n - (m-1)\tau\} \end{aligned} \quad (3.18)$$

Dabei gilt:

- $R(i, j)$ ist der Koeffizient der RP-Matrix in der i -ten Zeile und j -ten Spalte.
- ϵ ist der Grenzwert.
- Θ ist Heaviside-Funktion.
- x_i und x_j sind zwei Vektoren, die durch Datenpunkte x_i und x_j in einer Zeitreihe mit der Länge n erzeugt werden, die von m , n und τ abhängen.
- m die Dimension, ein Hyperparameter, der die Anzahl der Dimensionen also die Länge der Vektoren x_i und x_j bestimmt.
- τ ist die Zeitverzögerung, ein Hyperparameter und auch der Zeitschritt zwischen zwei Datenpunkten in einer Zeitreihe, die zur Bildung eines Vektors herangezogen wird.
- $\|x_i - x_j\|$ ist der Abstand zwischen zwei Vektoren.

Der Grenzwert fungiert in diesem Fall als Hyperparameter, der sich entscheidet, ob jeder Wert in der RP-Matrix einen Wert von eins oder null hat. Es kann sich um einen beliebigen Gleitkommawert oder einen Prozentsatz des maximalen Abstands zwischen zwei durch die Datenpunkte erstellten Vektoren handeln. Zum Vergleich mit dem Grenzwert wird der euklidische Abstand angewendet. In einer anderen Variante von RP wird der Grenzwert ignoriert, stattdessen ist $R(i, j)$ direkt gleich dem Abstand zwischen x_i und x_j . Der Output von RP ist eine quadratische Matrix der Form $(n - (m-1)\tau, n - (m-1)\tau)$. Eine Erhöhung von m und n kann zu einem Anstieg der Rechenkosten führen. Abbildung 3.9 ist ein Beispiel für RP einer sinusförmigen Ursprungszeitreihe mit n gleich 10, ϵ gleich 0.5, m und τ gleich 1, was bedeutet, wenn der Abstand zwischen 2 Datenpunkten kleiner als 0.5 ist, sollte ihr Pixel gleich eins sein.

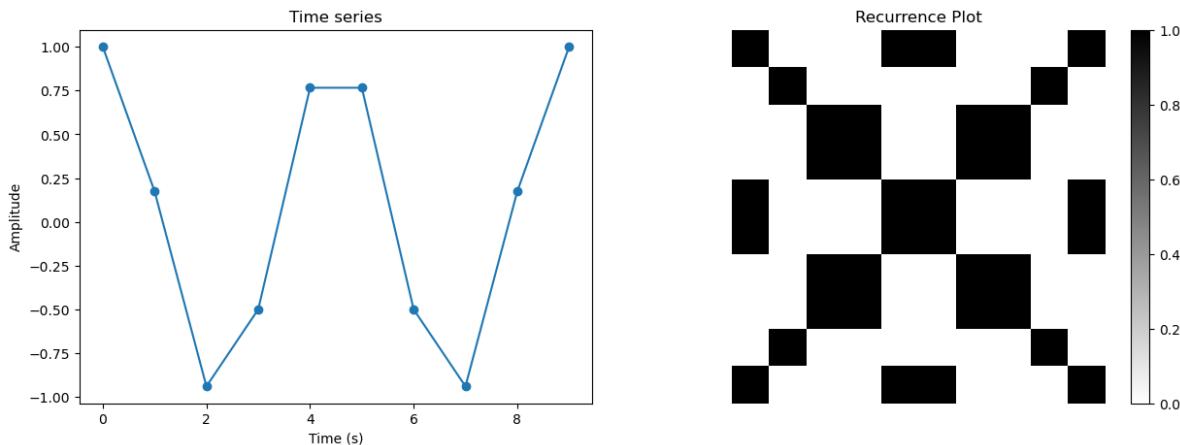


Abbildung 3.9.: Rekurrenzplot

3.3.5. Gramian-Winkelfeld

Das Gramian-Winkelfeld (GAF oder englisch Gramian angular field) ist auch eine Methode zur Umwandlung von Zeitreihen zu Bildern. Zuerst werden die Zeitreihen zwischen -1 und 1 normalisiert, dann in der Polarkoordinate umgewandelt und jeder Koeffizient in der Matrix ist die Summe oder Differenz ihrer Trigonometrie [31]. Formel 3.19 beschreibt detailliert, wie die GAF-Matrix berechnet wird.

$$\begin{aligned} \begin{cases} \phi_i = \arccos(x_i) & \forall x_i \in [-1, 1] \\ r_i = \frac{i}{N} & \forall i \in [0, N] \end{cases} \\ G(i, j) = f(\phi_i, \phi_j) \quad \forall \phi_i, \phi_j \in [0, \pi] \\ f(\phi_i, \phi_j) = \begin{cases} \cos(\phi_i + \phi_j) & \text{if method = summation} \\ \sin(\phi_i - \phi_j) & \text{if method = difference} \end{cases} \end{aligned} \quad (3.19)$$

Dabei gilt:

- x_i ist der i -te Datenpunkt in der Zeitreihe mit der Länge N .
- ϕ_i und r_i sind die Koordinaten von x_i in der Polarkoordinate.
- $G(i, j)$ ist der Koeffizient der GAF-Matrix in Zeile i und Spalte j .
- $method$ ist ein Hyperparameter.

Aus der Formel 3.19 ist ersichtlich, dass bei der Berechnung von \arccos nur positive Winkel zwischen 0 und π berücksichtigt werden. $method$ in GAF fungiert als Hyperparameter, der bestimmt, ob der Sinus der Differenz oder der Kosinus der Summe zweier Winkel verwendet wird. Formel 3.20 beschreibt die GAF-Matrix für den Fall, dass $method$ „summation“ entspricht, sie wird als Gramian-Winkelsummenfeld (GASF) bezeichnet, im anderen Fall wird diese Matrix als Gramian-Winkeldifferenzfeld (GADF) genannt. Abbildung 3.10 zeigt eine GAF-Matrix mit einer Auswahl verschiedener $method$

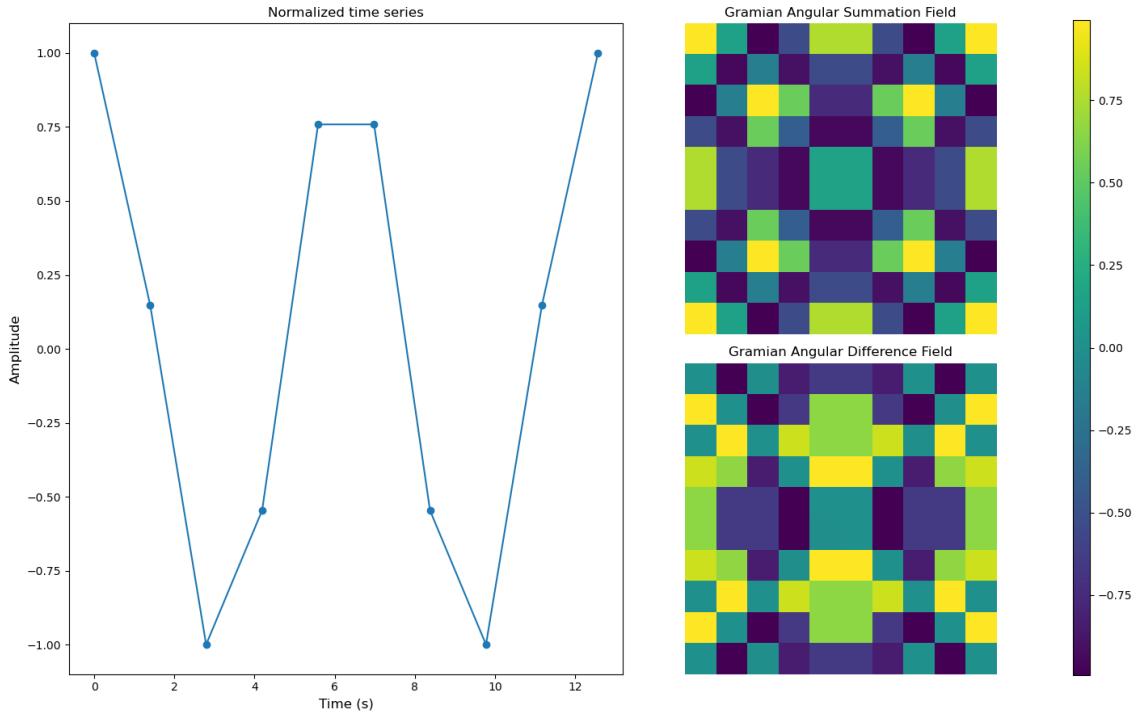


Abbildung 3.10.: Gramian-Winkelfeld

mit einer sinusförmigen Zeitreihe als Input, wobei n gleich 10 ist. Die GAF-Matrix hat die Form $\langle N, N \rangle$.

$$GASF = \begin{pmatrix} \cos(\phi_1 + \phi_1) & \cos(\phi_1 + \phi_2) & \cdots & \cos(\phi_1 + \phi_n) \\ \cos(\phi_2 + \phi_1) & \cos(\phi_2 + \phi_2) & \cdots & \cos(\phi_2 + \phi_n) \\ \vdots & \vdots & \ddots & \vdots \\ \cos(\phi_n + \phi_1) & \cos(\phi_n + \phi_2) & \cdots & \cos(\phi_n + \phi_n) \end{pmatrix} \quad (3.20)$$

3.3.6. Markov-Übergangsfeld

Eine Markov-Kette (englisch Markov chain) ist ein stochastisches Modell oder Prozess, der den Übergang zum nächsten Zustand beschreibt. Dieser Vorgang hängt nur vom aktuellen Zustand ab und nicht von der Abfolge der Ereignisse, die zuvor stattgefunden haben [32]. Ein Beispiel für eine Markov-Kette ist in Abbildung 3.11 links dargestellt. Dabei sind A, B und C die Zustände, und die Pfeile und Werte darauf geben die Wahrscheinlichkeit der Bewegung zwischen den Zuständen an. Die Summe der Wahrscheinlichkeiten aus einem Zustand ist gleich 1. Die Übergangsmatrix dient als allgemeine Beschreibung der Markov-Kette [33], wobei jeder Koeffizient in dieser Matrix die Wahrscheinlichkeit des Übergangs von einem Zustand mit Spaltenindex zu einem Zustand mit Zeilenindex darstellt. Abbildung 3.11 rechts zeigt die Übergangsmatrix der Markov-

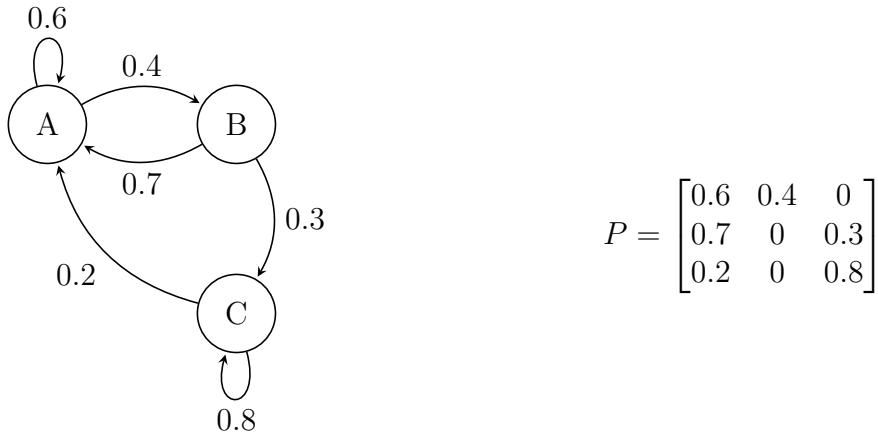


Abbildung 3.11.: Markov-Kette und Übergangsmatrix

Kette in Abbildung 3.11 links und Formel 3.21 ist ihre allgemeine Formel.

$$P(i, j) = p_{S_i \rightarrow S_j} \quad (3.21)$$

Dabei gilt:

- $P(i, j)$ ist der Koeffizient der Übergangsmatrix in Zeile i und Spalte j .
- $p_{S_i \rightarrow S_j}$ ist die Wahrscheinlichkeit der Übergang vom Zustand S_i zum Zustand S_j

Beim Beispiel der Übergangsmatrix in Abbildung 3.11 ist 0.7 die Wahrscheinlichkeit des Übergangs von Zustand B zu A, 0.8 die Wahrscheinlichkeit des Übergangs von Zustand C zu sich selbst und so weiter. Es ist zu beachten, dass die Summe der in einer Zeile dieser Matrix enthaltenen Koeffizienten 1 ergibt.

Markov-Übergangsfeld (MTF oder englisch Markov transition field) ist eine Methode zur Konvertierung einer Zeitreihe zum Bild. Das Ziel ist es, die Übergangswahrscheinlichkeiten von Datenpunkten in diskreten Zeitreihen zwischen den Bins zu erfassen [31]. Die Bins funktionieren ebenfalls auf die gleiche Weise wie der oben beschriebene Zustand, bedeuten jedoch im Kontext von MTF einen Teil eines Wertebereichs, der bestimmte Datenpunkte umfasst. Diese Methode erfordert zwei Hyperparameter, n_bins und $strategy$, wobei n_bins die Anzahl der Zustände also Bins ist und $strategy$ beschreibt, wie die Bins im Werterraum aufgeteilt sind. Zu den in $strategy$ enthaltenen Optionen gehören [34]:

- **uniform** bedeutet, dass der Wertebereich in Bins gleicher Breite unterteilt wird. Wenn der Bereich beispielsweise zwischen 0 und 5 liegt und n_bins 2 ist, enthält das erste Bin Datenpunkte zwischen 0 und 2,5 und das zweite Bin Datenpunkte zwischen 2,5 und 5.
- **quantile** bedeutet, dass der Wertebereich in Teile unterteilt wird, sodass jeder Teil über die gleiche Anzahl von Datenpunkten verfügt. Beispielsweise umfasst für eine Zeitreihe mit 10 Datenpunkten und n_bins gleich 2 das erste Bin die

5 Datenpunkte mit dem größeren Wert und das zweite Bin die verbleibenden 5 Datenpunkte.

- **normal** bedeutet, dass der Wertebereich gemäß der Gauß-Verteilung aufgeteilt wird.

Es ist zu beachten, dass sich die Bins nicht überlappen und nicht verschachteln. Das heißt, jeder Datenpunkt gehört nur zu einem Bin. Um zunächst die Übergangsmatrix im Kontext von Zeitreihen und Bins zu berechnen, werden Formeln 3.22 und 3.23 angewendet.

$$P(ib, jb) = \frac{N_{ib \rightarrow jb}}{N_{ib}} \quad \forall ib, jb \in [1, n_bins] \quad (3.22)$$

$$N_{ib \rightarrow jb} = \sum_{i=1}^{N-1} \begin{cases} 1 & \text{if } x_i \in B_{ib} \text{ and } x_{i+1} \in B_{jb} \\ 0 & \text{otherwise} \end{cases} \quad (3.23)$$

Dabei gilt:

- ib und jb sind die Indexe von Bins.
- B_{ib} und B_{jb} sind jeweils das ib -te und jb -te Bin.
- $P(ib, jb)$ ist der Koeffizient der Übergangsmatrix in der ib -ten Zeile und jb -ten Spalte.
- N_{ib} ist die Anzahl der Paare von x_i und x_{i+1} , wenn zwei aufeinanderfolgende Datenpunkte x_i und x_{i+1} jeweils im ib -ten und einem beliebigen Bin sind.
- $N_{ib \rightarrow jb}$ ist die Anzahl der Paare von x_i und x_{i+1} , wenn zwei aufeinanderfolgende Datenpunkte x_i und x_{i+1} jeweils im ib -ten und jb -ten Bin sind.
- x_i ist der i -te Datenpunkt, der in einer Zeitreihe der Länge N enthalten ist.

Um die MTF-Matrix zu berechnen, ist es notwendig, die oben definierte Übergangsmatrix gemäß Formel 3.24 anzuwenden. MTF ist eine quadratische Matrix der Form $\langle N, N \rangle$ und diagonal symmetrisch.

$$M(i, j) = P(ib, jb) \quad \text{if } x_i \in B_{ib} \text{ and } x_j \in B_{jb} \quad \forall i, j \in [1, N] \quad (3.24)$$

Dabei gilt:

- $M(i, j)$ ist der Koeffizient der Übergangsmatrix in der i -ten Zeile j -ten und Spalte.
- x_i und x_j sind jeweils die i -ten und j -ten Datenpunkte der Zeitreihe der Länge N .

Abbildung 3.12 ist ein Beispiel für ein MTF, bei dem der Input eine sinusförmige Zeitreihe der Länge 10 mit n_bins von 5 und *strategy* als „quantile“ ist. Datenpunkte, die zum selben Bin gehören, haben dieselbe Farbe.

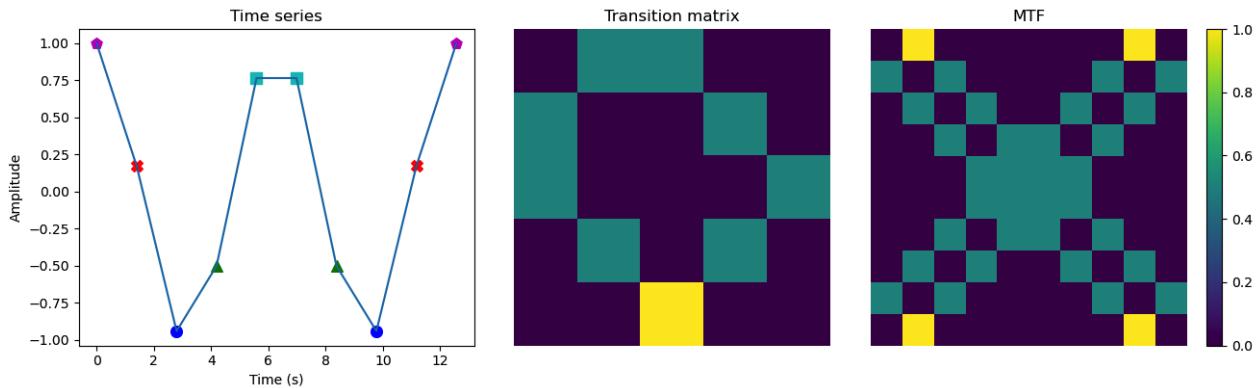


Abbildung 3.12.: Markov-Übergangsfeld

3.4. Temporal Cycle-Consistency Learning

Temporal Cycle-Consistency Learning (TCL) wurde erstmals im April 2020 in der gleichnamigen wissenschaftlichen Arbeit vorgestellt. Im Kontext wollen die Autoren Frames mit gleicher Funktion aus zwei aufeinanderfolgenden Videosequenzen miteinander verknüpfen [35]. Wenn beispielsweise zwei Videos einen Baseballspieler zeigen, der versucht, einen Ball zu werfen. Es ist davon ausgegangen, dass die beiden Videos Aktionen von zwei verschiedenen Personen mit unterschiedlicher Kleidung in zwei anderen Umgebungen aus zwei verschiedenen Perspektiven zeigen. Jeder Rahmen im Video beschreibt die

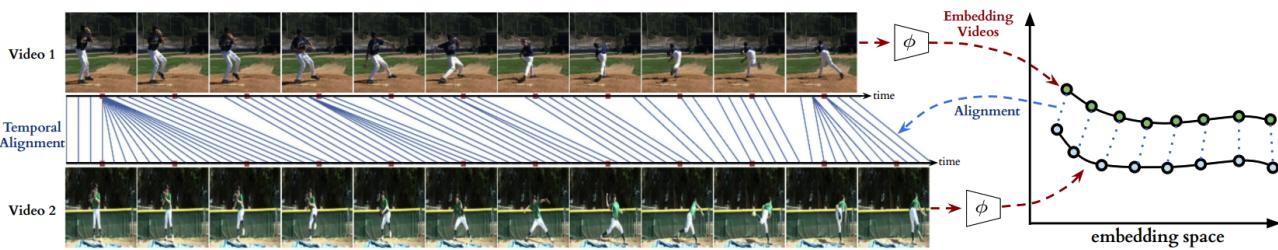


Abbildung 3.13.: Temporal Cycle-Consistency Learning [35]

Phasen der Aktion. Beispielsweise steht der Baseballspieler zunächst im Ruhezustand still und hält den Ball vor seiner Brust. Bei der Vorbereitung zum Ballwurf hebt er ein Bein an und streckt einen Arm nach hinten, um Schwung zu gewinnen. Beim Ballwurf ist der Schwerpunkt des Athleten nach vorne gerichtet. Dies kann in der Abbildung 3.13 dargestellt werden.

Die Frage ist also, wie ein künstliches neuronales Netzwerk trainiert wird, um Frames, die denselben Zustand einer Aktion darstellen, zusammenzufügen, um inhaltliche Korrespondenzen im Zeitverlauf in verschiedenen Videos zu erkennen? Methoden des überwachten Lernens sind stark von Labels abhängig. Das heißt, jedes Frame im Video muss mit einem Label versehen sein. An diesem Punkt treten zwei Probleme auf. Erstens muss die Labels manuell von Menschen vorgenommen werden, und zweitens ist es

unmöglich zu wissen, wie viele Labels benötigt werden. Ein anderer überwachter Ansatz besteht in der Frame-zu-Frame-Ausrichtung von Videos, was bedeutet, dass zwei entsprechende Frames zu einem selben Zeitpunkt jedes Videos denselben Aufkleber haben. Dies ist für reale Daten äußerst unnatürlich und das Modell hat Problem mit Aktionsvariationen [35].

TCL ist ein selbst-überwachter Lernansatz (englisch self-supervised learning), der das Potenzial hat, die oben genannten Probleme zu lösen, da er das Problem fehlender Datenlabels lösen kann. Das Modell erzeugt einen Feature-Raum, in dem zwei Feature-Vektoren zweier Frames mit der gleichen Funktion möglichst nahe beieinander liegen, dieses Konzept wird als zykluskonsistente Punkte genannt (CCP oder englisch cycle-consistent points). Das Ziel der Modelloptimierung ist es, möglichst viele CCPs zu haben [35].

3.4.1. Zykluskonsistenz und Optimierung

In diesem Unterabschnitt wird das Konzept, die Bestimmung der Zykluskonsistenz und die Optimierung des Modells näher erläutert. Gegeben seien ein Sequenz S mit N Frames und T mit M Frames. Ihre Feature-Vektoren im Feature-Raum sind jeweils U bzw. V , wobei ϕ das neuronale Netzwerk mit den lernfähigen Gewichten w ist. Die Beziehung dazwischen kann durch Formel 3.26 ausgedrückt werden [35].

$$\begin{aligned} S &= \{s_1, s_2, \dots, s_N\} \\ T &= \{t_1, t_2, \dots, t_M\} \\ U &= \{u_1, u_2, \dots, u_N\} = \phi(s_i, w) \\ V &= \{v_1, v_2, \dots, v_M\} = \phi(t_i, w) \end{aligned} \tag{3.25}$$

Vom Punkt $u_i \in U$ aus wird $v_j \in V$ als sein nächster Nachbar (englisch nearest neighbor) bezeichnet, wenn der Abstand zwischen u_i und v_j der kürzeste im Feature-Raum ist. Von $v_j \in V$ ist $u_k \in U$ sein nächster Nachbar, wenn der Abstand zwischen v_j und u_k am kürzesten ist. u_i ist Zykluskonsistenz, wenn und nur wenn i gleich k ist. Dies kann durch Formel 3.26 gezeigt werden [35]. Abbildung 3.14 zeigt den Fall, dass u_i zykluskonsistent und nicht zykluskonsistent ist.

$$\begin{aligned} v_j &= \operatorname{argmin}_{v \in V} \|u_i - v\| \\ u_k &= \operatorname{argmin}_{u \in U} \|v_j - u\| \\ u_i \text{ ist cycle-consistent if } i &= k. \end{aligned} \tag{3.26}$$

Um das Netzwerk zu trainieren, wird zuerst seine Verlustfunktion bestimmt. Betrachte zuerst einen Punkt $u_i \in U$ und sein weicher nächster Nachbar zu V in Feature-Raum wird durch Formel 3.27 [35] berechnet, wobei α_j die Wahrscheinlichkeit des kürzesten Abstands zwischen einem Punkt $u_i \in U$ und $v_j \in V$ von allen Abständen zwischen u_i

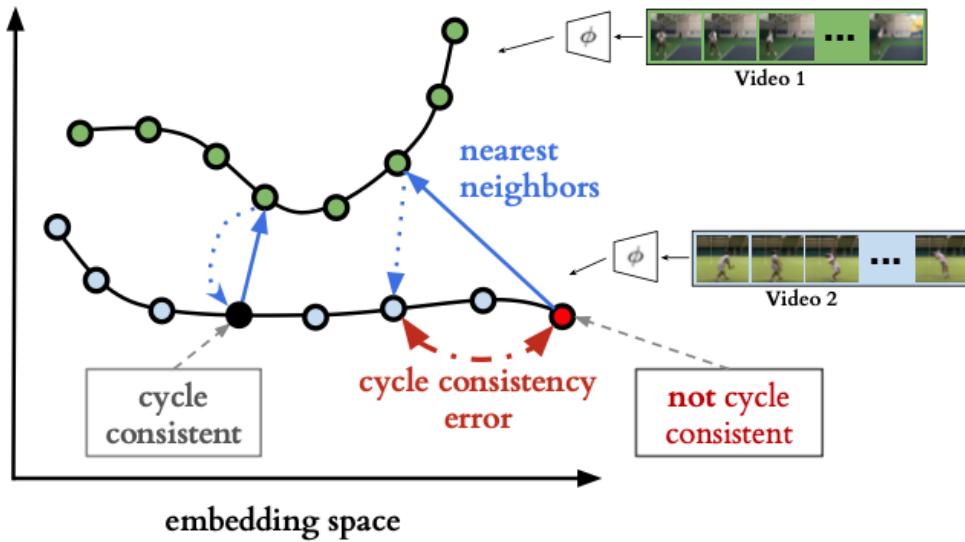


Abbildung 3.14.: Zykluskonsistenz [35]

und allen $v \in V$ ist.

$$\tilde{v} = \sum_j^M \alpha_j v_j \quad (3.27)$$

$$\alpha_j = \frac{e^{-\|u_i - v_j\|^2}}{\sum_k^M e^{-\|u_i - v_k\|^2}}$$

Unter der Annahme, dass die Dimension vom Output des Netzwerks D ist, haben U und V die Formen $\langle N, D \rangle$ bzw. $\langle M, D \rangle$, was der Form $\langle 1, D \rangle$ von u_i und v_j entspricht. Im Vergleich zur Formel 3.27 ist α_j eine Skala, dieser Wert wird mit v_j multipliziert, um \tilde{v}_j zu berechnen. Schließlich ist der weiche nächste Nachbar \tilde{v} der Form $\langle 1, D \rangle$ von u_i zu allen Punkten in V , der die Summe aller Produkte zwischen dem entsprechen α_j und Punkt $v_j \in V$ ist. Als nächstes wird die Wahrscheinlichkeit des kürzesten Abstands β_k zwischen dem Punkt \tilde{v} und $u_k \in V$ von allen Abständen zwischen \tilde{v} und allen $u \in U$ berechnet. Die Berechnung von β_k ähnelt der Berechnung von α_j . Der Grund für diese Berechnung ist, dass β_k voraussichtlich seinen Maximalwert von 1 erreichen wird, wenn k gleich i ist, da u_i dann als zykluskonsistent betrachtet wird. Das Problem wird nun zur Klassifikation mit N Klassen von U , das durch die Kreuzentropieverlustfunktion wie Formel 3.28 optimiert wird, wobei y ein wahres Label ist, das Nullwerte enthält, mit Ausnahme am Index i gleich 1. Der Zweck der Optimierung dieser Verlustfunktion

besteht darin, so viele Paare von CCPs wie möglich zu haben. [35].

$$\beta_k = \frac{e^{-\|\tilde{v} - u_k\|^2}}{\sum_j^N e^{-\|\tilde{v} - u_j\|^2}} \quad (3.28)$$

$$L_{cyc}^{(U,V)} = - \sum_{i=1}^N y_i \log(\beta_i)$$

3.4.2. Zykluskonsistenz von Zeitreihen

In diesem Abschnitt wird TCL zur Bestimmung von RUL angewendet. Auf die Idee dieser Methode wird von [36] verwiesen. Es ist deutlich zu erkennen, dass die Zeitreihen, die den Lebenszyklus einer Maschine beschreiben, sequentiell sind. Das heißt, es ist durchaus möglich, den Unterschied deutlich zu erkennen, wenn eine Maschine gesund und nicht benutzbar ist. Allerdings beschreibt die RUL zu einem Zeitpunkt manchmal nicht eindeutig den Gesundheitszustand (siehe Abbildung 3.15 links). Der Vorteil von TCL bei Problemen im Zusammenhang mit RUL besteht darin, dass es die Korrelation zwischen zwei Sequenzen erkennt. Genauer gesagt wird ein Modell trainiert, um in der Zeitreihe den ersten vorhergesagten Zeitpunkt (FPT oder englisch first predicting time) zu bestimmen, an dem die Maschine beginnt, sich zu verschlechtern (siehe Abbildung 3.15 rechts). Angenommen ist der Input des Modells eine Reihe von Zeitfenstern, die aus der Zeitreihe zu Bildern konvertiert wurden. Dann wird der RUL-Prozentsatz (englisch percentage RUL oder PRUL) der j -ten Trainingsmaschine zum Zeitpunkt i durch Formel 3.29 berechnet [36]. y_i^j gilt gleich 1 als gesund und kleiner als 1 bis 0 als Degradierung.

$$y_i^j = \begin{cases} \frac{N_{sample}^j - i}{N_{sample}^j - T^j} & \text{if } i \in [T^j, N_{sample}^j] \\ 1 & \text{otherwise} \end{cases} \quad (3.29)$$

Dabei gilt:

- N_{sample}^j ist die Länge der Zeitreihe der j -ten Trainingsmaschine.
- T^j ist der Zeitpunkt von FPT der j -ten Trainingsmaschine.

Das neuronale Netzwerk im Trainingsprozess wird entsprechend der zykluskonsistenten Verlustfunktion wie Formeln 3.27 und 3.28 optimiert, wobei U und V jeweils zwei Feature-Vektoren zweier Run-to-Failure-Zeitfenstersequenzen S und T von zwei Trainingsmaschinen im Feature-Raum sind, die nacheinander in das Modell eingespeist werden. Abschließend wird die Verlustfunktion für den gesamten Trainingsdatensatz als Formel 3.30 berechnet [36].

$$L_{cyc}^{train} = \sum_{U=1}^{N_{train}-1} \sum_{V=U+1}^{N_{train}} (L_{cyc}^{(U,V)} + L_{cyc}^{(V,U)}) \quad (3.30)$$

Dabei gilt:

- L_{cyc}^{train} ist die Verlustfunktion aller Maschinen im Trainingsdatensatz. Sie umfasst alle Maschinenpaare, die miteinander gepaart werden können.
- $L_{cyc}^{(U,V)}$ ist die zykluskonsistente Verlustfunktion der Sequenz U in Bezug auf V .
- $L_{cyc}^{(V,U)}$ ist die zykluskonsistente Verlustfunktion der Sequenz V in Bezug auf U .
- N_{train} ist die Anzahl der Maschinen im Trainingsdatensatz.

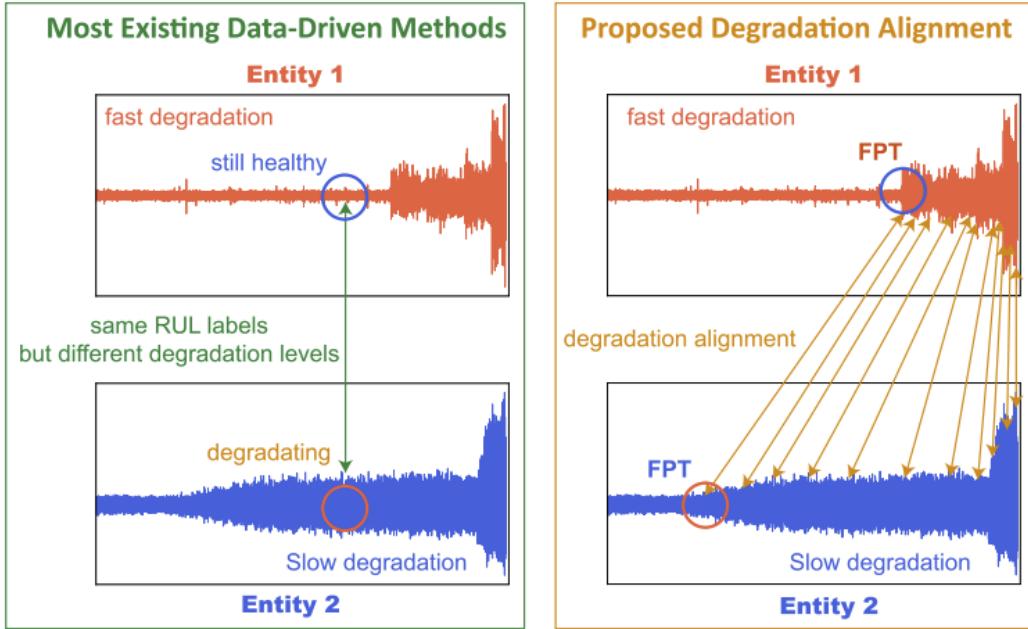


Abbildung 3.15.: RUL können nicht die Degradierung richtig darstellen [35].

Nachdem das Modell trainiert wurde, ist es möglich, vollständig zu wissen, ob jede Probe in der Sequenz S im Vergleich zu einer anderen Sequenz T einen konsistenten Zyklus aufweist oder nicht. Wenn in der Sequenz mehr als n_FPT in n_period benachbarten kontinuierlichen Punkten vorhanden sind, ist FPT der erste Zeitpunkt davon [36]. Ein Beispiel zur FPT-Bestimmung kann durch Abbildung 3.16 dargestellt werden, wobei schwarze Punkte zykluskonsistent und weiße Punkte nicht zykluskonsistent sind. n_FPT und n_period dienen als Hyperparameter.

Abschließend ist es notwendig, die FPT einer Trainingssequenz zu allen verbleibenden Sequenzen des Trainingsdatensatzes mithilfe der Formel 3.31 zu bestimmen und danach kann PRUL einfach durch Formel 3.29 ermittelt werden.

$$T_{train}^j = \frac{1}{N_{train} - 1} \sum_{i=1, i \neq j}^{N_{train}} T_{train}^{(j,i)} \quad (3.31)$$

Dabei gilt:

- T_{train}^j ist durchschnittliche FPT der j -ten Trainingsmaschine.

- $T_{train}^{(j,i)}$ ist FPT einer j -ten zu einer anderen i -ten Trainingsmaschine.

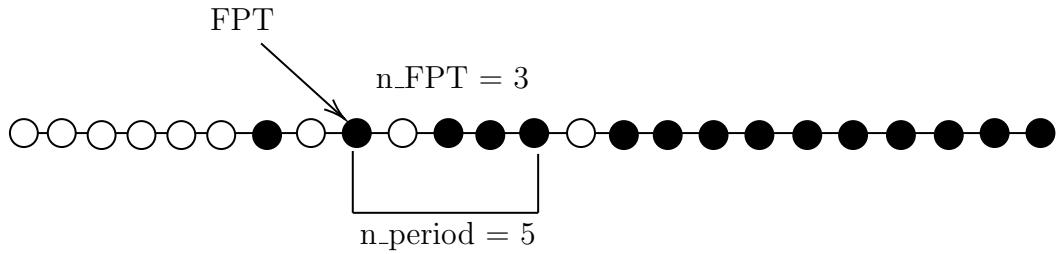


Abbildung 3.16.: Bestimmung von FPT

Beim Testen werden die Sequenzen im Testdatensatz mit Sequenzen im Trainingsdatensatz gepaart, um FPT wie Formel 3.32 zu berechnen.

$$T_{test}^j = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} T^{(j,i)} \quad (3.32)$$

Dabei gilt:

- T_{test}^j ist durchschnittliche FPT der j -ten Testmaschine.
- $T^{(j,i)}$ ist FPT einer j -ten Testmaschine zu einer i -ten Trainingsmaschine.

Der PRUL der Testmaschine kann jedoch nicht gemäß Formel 3.29 ermittelt werden, da N_{sample} der Testsequenzen nicht verfügbar ist. Um dieses Problem zu lösen, soll für jeden Featurevektor jeder Probe in der Testsequenz der nächste Nachbar in der Trainingssequenz im Trainingsdatensatz im Feature-Raum gefunden werden. PRUL jeder Testprobe ist derselbe wie PRUL der Trainingsprobe, wenn sie im Feature-Raum einander am nächsten liegen. Dies kann durch Formel 3.33 ausgedrückt werden [36].

$$\begin{aligned} u_m &= \operatorname{argmin} \|v_n - u_i\|^2 \quad \forall i \in [1, N_{samples}] \\ y_n &= y_m \end{aligned} \quad (3.33)$$

Dabei gilt:

- v_n ist der Feature-Vektor der n -ten Probe einer Testsequenz.
- u_m ist der Feature-Vektor der m -ten Probe einer Trainingssequenz und auch der nächste Nachbar von v_n im Feature-Raum.
- y_n ist PRUL der n -ten Probe einer Testsequenz.
- y_m ist PRUL der m -ten Probe einer Trainingssequenz.
- $N_{samples}$ ist die Anzahl der Proben einer Trainingssequenz.

Letztendlich ist der PRUL einer Testsequenz wie Formel 3.34 der durchschnittliche PRUL-Wert für alle Trainingssequenzen. Der RUL jeder Probe einer Testsequenz kann

gemäß Formel 3.35 berechnet werden [36].

$$y_{n,test}^j = \begin{cases} \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} y_{m,train}^i & \text{if } n > T_{test}^j \\ 1 & \text{otherwise} \end{cases} \quad (3.34)$$

$$r_{n,test}^j = \frac{n - T_{test}^j}{1 - y_{n,test}^j} \cdot y_{n,test}^j \quad \text{if } n > T_{test}^j \quad (3.35)$$

Dabei gilt:

- m ist der Index des nächsten Nachbarn von n .
- $y_{n,test}^j$ ist n -te PRUL der j -ten Testmaschine.
- $y_{m,train}^i$ ist m -te PRUL der i -ten Trainingsmaschine.
- N_{train} ist die Anzahl der Maschinen im Trainingsdatensatz.
- T_{test}^j ist durchschnittliche FPT der j -ten Testmaschine.
- $r_{n,test}^j$ ist n -te RUL der j -ten Testmaschine.

3.5. Hyperparameter-Tuning

Um Hyperparameter für das Modell auswählen zu können, wird Kreuzvalidierung (CV) in dieser Arbeit genutzt. Zur Parameteroptimierung wird Tree-structured-Parzen-Estimator [37] des Optuna-Frameworks verwendet. Nachdem die Hyperparameter mit dem besten Ergebnis im Validierungsdatensatz gefunden wurden, gibt es zwei Möglichkeiten, das Ergebnis im Testsatz zu bestimmen. Angenommen ist \mathbf{y} der Output und $\phi(w, \mathbf{x})$ ist das Netz einschließlich der Parameter w , der \mathbf{x} als Input, k als die Anzahl der Splitte.

1. Das Modell mit den ausgewählten Hyperparametern wird auf dem gesamten Trainings- und Validierungsdatensatz neu trainiert und anhand des Testdatensatzes ausgewertet. Dies kann durch die Formel 3.36 ausgedrückt werden.

$$\mathbf{y} = \phi(w, \mathbf{x}) \quad (3.36)$$

2. k Modelle, die mit dem Trainingsdatensatz trainiert wurden und die besten Ergebnisse mit dem Validierungsdatensatz bei CV erzielen, werden mit dem Testdatensatz ausgewertet. Das Endergebnis ist der Mittelwert von k Modellen, der in Gleichung 3.37 ausgedrückt werden kann.

$$\mathbf{y} = \frac{1}{k} \sum_{i=1}^k \phi_{cv,i}(w_i, \mathbf{x}_i) \quad (3.37)$$

4. Experimente und Ergebnisse

In diesem Kapitel wird ausführlich auf die Daten, den Umgang mit ihnen, das Ziel dieser Arbeit, den Umgang mit dem Problem sowie die Ergebnisse eingegangen.

4.1. Datensätze

2012 veranstaltete das FEMTO-ST-Institut einen Wettbewerb namens IEEE PHM 2012 Data Challenge zur Vorhersage der verbleibenden Nutzungsdauer (RUL) der Lager. Dies ist ein wichtiger Teil rotierender Geräte und wirkt sich direkt auf die Betriebskapazität dieses Geräts aus. Die Vorhersage der RUL von Lagern kann Sicherheit und Wartungskosten in der Industrie gewährleisten. Die drei Lagereigenschaften Temperatur sowie Längs-

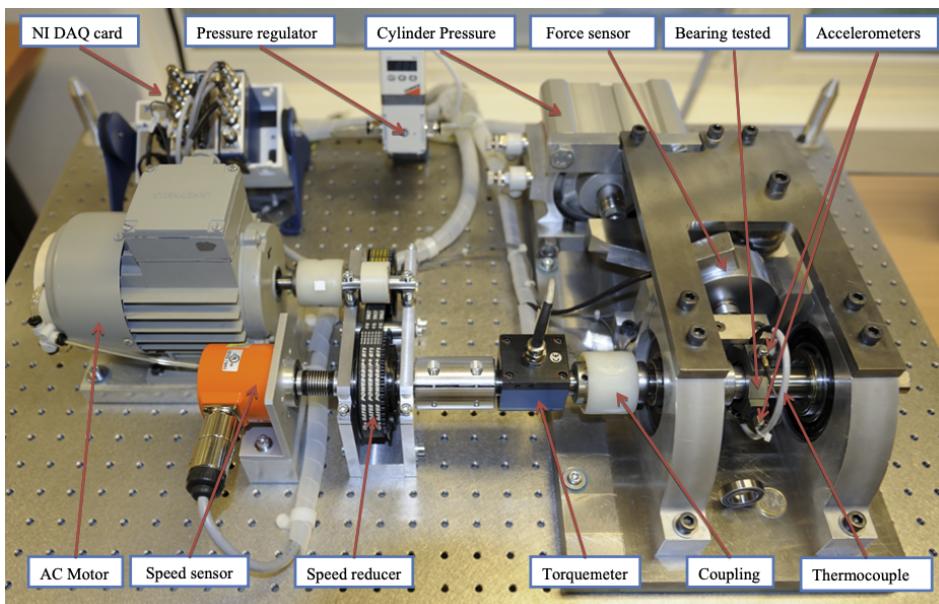


Abbildung 4.1.: PROGNOSTIA-Plattform [38]

und Querschwingung werden online über Sensoren unter konstanten Betriebsbedingungen gemessen. Der Betrieb der Lager erfolgt auf der experimentellen Plattform namens PROGNOSTIA (siehe Abbildung 4.1). Bei den Trainingsdaten handelt es sich um die Zeitreihen von 6 verschiedenen Lagern unter 3 verschiedenen Betriebsbedingungen vom gesunden Zustand bis zum Ende der Lebensdauer (EOL). Das Ziel des Teilnehmers in

diesem Wettbewerb und auch dieser Arbeit besteht darin, die RUL der verbleibenden 11 Lager im Testdatensatz vorherzusagen. Es wird davon ausgegangen, dass alle Lager unter den gegebenen Bedingungen und ohne sonstige Umweltbeeinträchtigungen betrieben werden. Alle Lager dieses Wettbewerbs werden bis zum Ende ihrer Lebensdauer

Tabelle 4.1.: Betriebsbedingungen der Lager [38]

Conditions	Rotating speed [rpm]	Load force [N]
1	1850	4000
2	1600	4200
3	1500	5000

unter unterschiedlichen Drehzahlen und Belastungen betrieben [38]. Die Betriebsbedingungen sind in Tabellen 4.1 und Details zum Datensatz können durch die Tabelle 4.2 ausführlich beschrieben werden. Die EOL-Spalte gibt die Lebensdauer der Lager an und die RUL-Spalte die verbleibende Zeit vom Abschneiden der Zeitreihen der Testlager bis zum Ende der Lebensdauer.

Tabelle 4.2.: Detaillierte Informationen zu den Lagern.

Datasets	Name	Condition	EOL [s]	RUL [s]	N_samples
Train data	Bearing1_1	1	28020	-	7175680
	Bearing1_3	1	8700	-	2229760
	Bearing2_1	2	9100	-	2332160
	Bearing2_2	2	7960	-	2040320
	Bearing3_1	3	5140	-	1318400
	Bearing3_2	3	16360	-	4190720
Test data	Bearing1_3	1	23740	5730	6080000
	Bearing1_4	1	14270	2890	3655680
	Bearing1_5	1	24620	1610	6305280
	Bearing1_6	1	24470	1460	6266880
	Bearing1_7	1	22580	7570	5783040
	Bearing2_3	2	19540	7530	5004800
	Bearing2_4	2	7500	1390	1922560
	Bearing2_5	2	23100	3090	5916160
	Bearing2_6	2	7000	1290	1794560
	Bearing2_7	2	2290	580	588800
	Bearing3_3	3	4330	820	1111040

Für horizontale und vertikale Vibrationssignale beträgt die Abtastfrequenz 25.6 kHz, es werden jedoch nur 2560 Datenpunkte während der ersten 0.1 Sekunde aller 10 Sekunden erfasst. Die Visualisierung der vertikalen und horizontalen Vibrationssignale in den Trainings- und Testdatensätzen wird durch die blaue bzw. rote Zeitreihe in Anhang A dargestellt. Der grüne Teil im Testdatensatz stellt das Abschneiden der Zeitreihe bis

zum Ende der Lebensdauer dar. Rohdaten liefern den Zeitpunkt, zu dem jeder Wert von den Sensoren gemessen wurde. Da die Temperatur nur in bestimmten Lagern gemessen wird, werden in dieser Arbeit nur Vibrationssignale als Input für das Modell verwendet.

4.1.1. Score

Zusätzlich zur Verlustfunktion, mit der die Leistung des Modells bewertet werden kann, gibt es auch einen Score [38]. Die Formel zur Berechnung ist in Formeln 4.3, 4.4 und Abbildung 4.2 ausgedrückt.

$$\%Er_i = 100 \times \frac{RUL_{true,i} - RUL_{predict,i}}{RUL_{true,i}} \quad (4.1)$$

$$A_i = \begin{cases} \exp^{-\ln(0.5) \cdot (Er_i/5)} & \text{if } E_i \leq 0 \\ \exp^{\ln(0.5) \cdot (Er_i/20)} & \text{if } E_i > 0 \end{cases} \quad (4.2)$$

$$Final_score = \frac{1}{11} \sum_{i=1}^{11} A_i \quad (4.3)$$

$$Score = \frac{1}{N} \sum_{i=1}^N A_i \quad (4.4)$$

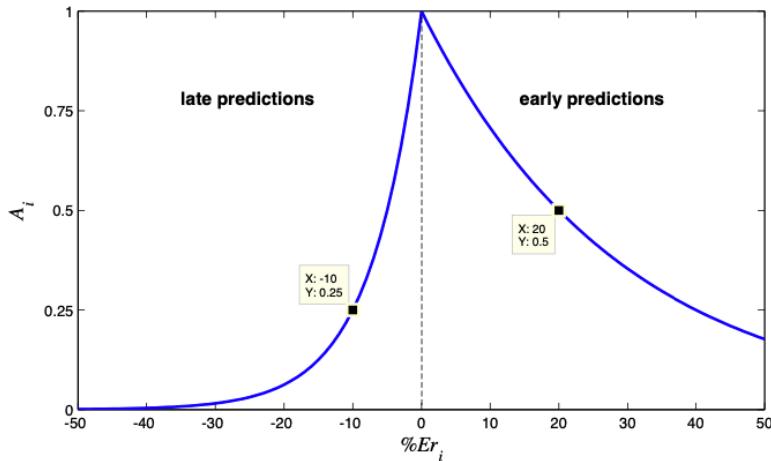


Abbildung 4.2.: Score-funktion [38]

Es ist zu beachten, dass bei diesem Wettbewerb die Zeitreihen der 11 Testlager geschnitten werden und die Aufgabe darin besteht, die RULs zum Zeitpunkt der Schnittpunkte der Zeitreihen zu berechnen (siehe RUL-Spalte in Tabelle 4.2). Dieses Ergebnis wird durch Final_score wie Formel 4.3 dargestellt. Der Fairness halber wird in dieser Arbeit

jedoch vorgeschlagen, den Score wie Formel 4.4 hinzuzufügen zu bestimmen, also das durchschnittliche Ergebnis aller Proben im Datensatz. Anhand von Abbildung 4.2 ist ersichtlich, dass das Modell eine bessere Leistung erbringt, wenn die frühere RUL als spätere RUL vorhergesagt werden kann.

4.1.2. Verwandte Arbeiten

Bei der IEEE PHM 2012 Data Challenge ging der Gewinner an Sutrisno et al. Die von ihnen verwendete Methode war „Vibration Frequency Signature Anomaly Detection and Survival Time Ratio“ und erreichte einen Final_score von 0.3068 [39]. 2014 erreichten Hong et al. mit der „Self-organization mapping“ ein Ergebnis von 0.418 [40]. 2015 wurde das Ergebnis von Lei et al. mithilfe der Methode „Particle Filtering with Maximum Likelihood Estimator“ auf 0.4285 erhöht [41]. Das höchste Ergebnis im Jahr 2018 liegt bei 0.69 und gehört Patil et al. Sie verwenden „Random forest and Gradient Boosting Regressor“ [42].

4.2. Vorbearbeitung der Zeitreihen

Im Vorverarbeitungsschritt werden die Zeitreihen zu Bildern umgewandelt. In dieser Arbeit werden insgesamt 6 Methoden angewendet, darunter Kurzzeit-Fourier-Transformation (STFT), Log-Mel-Spektrogramm (LMS), kontinuierliche Wavelet-Transformation (CWT), Rekurrenzplot (RP), Gramian-Winkelfeld (GAF) und Markov-Übergangsfeld (MTF). Zunächst werden die Run-to-Failure-Zeitreihen gemäß dem Min-Max-Skalierung oder Standardisierung normalisiert. Es ist wichtig zu beachten, dass die Normalisierung zuerst für alle Datenpunkte des Trainingsdatensatzes berechnet und dann auf die Test- und Validierungsdatensätze angewendet wird. Fensterung lässt sich anwenden, um die Proben mit einer *window_size* von 25600 Datenpunkten und einer *hop_size* von 2560 zu generieren. Zusammenfassend wird es 7480 Proben im Trainingsdatensatz und 17256 Proben in den Testdaten geben. Durch viele Tests verbrauchen die drei Methoden GAF, MTF und RP eine Menge Ressourcen für die Berechnung des Bildes. Daher werden die Fenster also Proben vor der Konvertierung zu Bildern durch PAA auf 224 Datenpunkte komprimiert. Da es sich bei STFT, CWT und LMS um Methoden handelt, die sich auf die Analyse der im Signal vorhandenen Frequenzen konzentrieren, ist es von Vorteil, über mehr Datenpunkte zu verfügen. Basierend auf unterschiedlichen Parametern weist die Ausgabe jeder Methode jedoch unterschiedliche Größen auf, sodass bikubische Interpolation angewendet wird, um die Bildgröße zu ändern. Vor der Eingabe in das Modell hat das Bild die Größe (224,224). Auch diese Spektrogramme werden auch normalisiert, das heißt, die Werte der Pixel jedes Kanals liegen zwischen 0 und 1. Es ist zu beachten, dass Informationsverluste durch Bildkomprimierung oder Zeitreihen unvermeidlich sind. Tabellen 4.3 und 4.4 beschreibt die Zeitreihenvorverarbeitung im Detail.

Tabelle 4.3.: Anzahl der Proben nach der Fensterung

Datasets	Name	N_windows
Train data	Bearing1_1	2794
	Bearing1_2	862
	Bearing2_1	902
	Bearing2_2	788
	Bearing3_1	506
	Bearing3_2	1628
Test data	Bearing1_3	2366
	Bearing1_4	1419
	Bearing1_5	2454
	Bearing1_6	2439
	Bearing1_7	2250
	Bearing2_3	1946
	Bearing2_4	742
	Bearing2_5	2302
	Bearing2_6	692
	Bearing2_7	221
	Bearing3_3	425

Tabelle 4.4.: Vorbearbeitung der Zeitreihen

Method	Normalization	Downsampling	Resize	Framework
STFT	Standardize	-	Bicubic	librosa
LMS	Standardize	-	Bicubic	librosa
CWT	Standardize	-	Bicubic	pywt
RP	Min-max-normalization	PAA	-	pyts
GAF	Min-max-normalization	PAA	-	pyts
MTF	Min-max-normalization	PAA	-	pyts

4.3. Ansätze und Ergebnisse

In diesem Abschnitt werden die Ansätze für das Problem erläutert. Die Daten werden wie in Abschnitt 4.2 vorverarbeitet. Jedes Zeitreihen-zu-Bild-Encoding verfügt über ihre eigenen Hyperparameter. Allerdings wird es einige feste Hyperparameter geben, die $batch_size = 32$, $epoch = 100$ sind. Zur Optimierung der Hyperparameter wird k-Fold-Kreuzvalidierung mit $k = 5$ angewendet. Die Labels also RUL der Proben wird ebenfalls zwischen 0 und 1 normalisiert. Das heißt, das Modell, das über einen letzten fully-connected Layer mit der Ausgabedimension von 1 verfügt, wird einen Wert vorhersagen, der den Prozentsatz der verbleibenden Gesundheit darstellt. Da die Rohdaten vollständig angeben, wann diese Proben entnommen wurden und bevor sie diesem Experiment und der Messung von Vibrationssignalen unterzogen wurden, befinden sich die

Lager in einem gesunden Zustand. Daher ist es möglich, RUL in Sekunden aus Modellvorhersagen zu bestimmen. Ähnlich wie bei Regressionsaufgaben wird die Verlustfunktion MAE zum Aktualisieren der Gewichte verwendet. In den ersten drei Ansätzen wird nur eine Grafikkarte zum Trainieren des Modells benötigt.

4.3.1. Im_CNN

Der Modellaufbau ist nie eine einfache Angelegenheit und erfordert immer viel Recherchezeit sowie viele Ressourcen und Kosten zum Testen. Modelle können auch nur dann eine gute Leistung erbringen, wenn sie auf umfangreichen Datensätzen trainiert werden. Es ist ersichtlich, dass die explosionsartige Verbreitung von CNN und den jetzt verfügbaren Modellen in den letzten Jahren zu guten Ergebnissen beim ImageNet-Datensatz geführt hat und auch die Kosten ein Thema sind, auf das sich die Forschung konzentriert. Dabei handelt es sich allesamt um tief strukturierte Modelle, wie zum Beispiel EfficientNet, das erstmals im Mai 2019 eingeführt wurde. Genauer gesagt konnte das einfachste strukturierte Modell, EfficientNet-B0, mit nur 5,3 Millionen Parametern Top-1- und Top-5-Accuracy im ImageNet-Datensatz von 77,1 % bzw. 93,3 % erreichen. Eine erstaunliche Zahl im Vergleich zu ResNet oder DenseNet in der Vergangenheit [5]. Aus allen oben genannten Gründen und in dieser Arbeit verfügbaren Ressourcen wird die Struktur von EfficientNet-B0 für das Training ausgewählt. Abbildung 4.3 beschreibt den Aufbau im Detail [43]. Wie viele andere CNN verwendet EfficientNet-B0 hauptsächlich

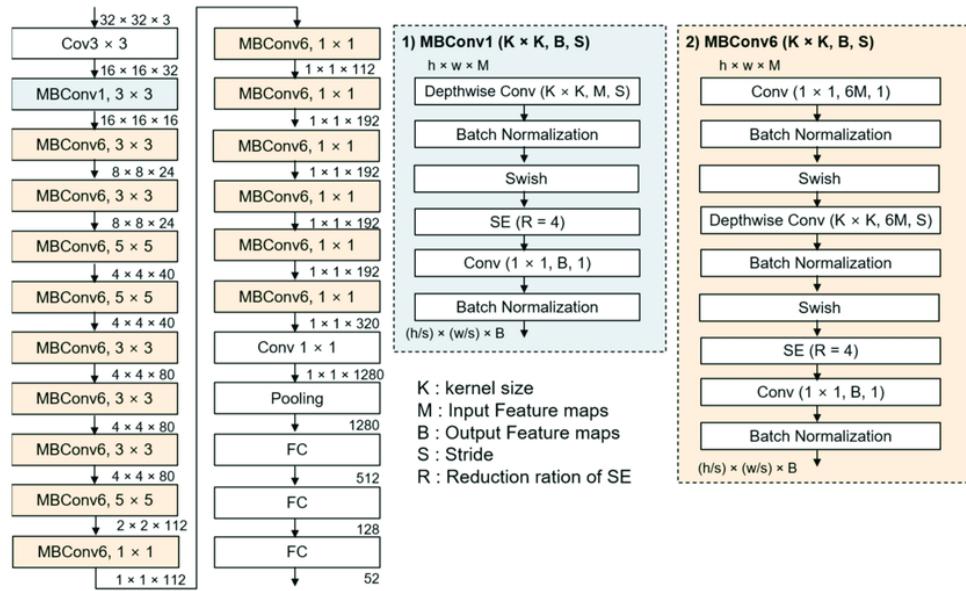


Abbildung 4.3.: Strukture von EfficientNet-B0 [43]

Faltungslayer, Batch-Normalisierung und SiLU also Swish als Aktivierungsfunktion. Jeder MBConv-Blockblock verfügt über eine Mobile-Inverted-Bottleneck-Struktur, die auch in MobileNetV2 enthalten ist [44], außerdem wird auch ein Squeeze-and-Excitatio-Block

(SE) angewendet [45], [5].

Bei dem ersten Ansatz Im_CNN erhält das Modell mit der Struktur von EfficientNet-B0 mit zufällig initialisierten Parametern die 2D-Spektralbilder der Größe $\langle 2, 224, 224 \rangle$ als Input, die vertikale und horizontale Vibration beschreiben. Das bedeutet, dass das Modell von Grund auf trainiert wird. Der Grund für die Nichtverwendung von Fine-Tuning liegt darin, dass der Input vom vortrainierten EfficientNet-B0 auf ImageNet-Datensatz ein RBG-Bild der Größe $\langle 3, 224, 224 \rangle$ ist und die Daten in dieser Arbeit sind Spektrogramme, die sich mit Merkmalen und Eigenschaften von den Bildern in ImageNet unterscheiden. Dieser Ansatz basiert auf überwachtem Lernen, das heißt, für jeden Input sollte das Modell die RUL zum Zeitpunkt der Erfassung des Spektrums ausgeben, indem der letzte fully-connected Layer auf die Ausgabedimension von 1 wie Abbildung B.1 verfeinert wird. Das Modell umfasst 4 Millionen lernbare Parameter. Die Ergebnisse und Hyperparameter dieses Ansatzes sind jeweils in Tabellen C.1 und C.2 dargestellt, wobei Result-Spalte und Result-CV-Spalte jeweils die erste und zweite Möglichkeit zu Bestimmung der Ergebnisse im Abschnitt 3.5 sind. Bei diesem Ansatz zeigt STFT die beste Leistung, seine Ergebnisse in der Tabelle sind fett geschrieben und seine Visualisierung ist in Abbildung C.1 dargestellt.

4.3.2. Ts_im_CNN

Es ist ersichtlich, dass das künstliche neuronale Netzwerk in dem obigen Ansatz nur Bilder als Inputs empfangen kann. Die in dieser Arbeit betrachteten sechs Zeireihen-zu-Bilder-Encoding können nicht wieder zu Zeitreihen aus Bildern zurückkonvertiert werden. In der Zeitreihe kann es Eigenschaften geben, die Bilder nicht klar darstellen können, zum Beispiel die Größen der Datenpunkte. Daher wird bei diesem Ansatz erwartet, dass das Hinzufügen von Zeitreihe als Input zusammen mit Spektralbildern die Leistung des Netzes verbessert.

Das Spektrogramm wird weiterhin wie in Abschnitt 4.2 beschrieben erstellt. Die Zeitreihe wird wie Tabelle 4.4 gemäß der Zeitreihen-zu-Bild-Konvertierung normalisiert, jedoch nicht komprimiert. Das heißt, die Zeitreihe hat bei der Einspeisung ins Netzwerk immer noch die Länge von 25600.

Für die Bildverarbeitung wird weiterhin die Struktur von EfficientNetB0 verwendet. Die Modellstruktur für die Zeitreihenverarbeitung wird aus [46] referenziert und leicht an die Inputs als Zeitreihen angepasst.

Diese Struktur besteht hauptsächlich aus 1D-Faltung-, 1D-Batchnorm1D-, 1D-Maxpool-Layer und der ReLU-Aktivierungsfunktion. Die Feature-Maps der schon bearbeiteten Zeitreihen werden dann in Bildern umgeformt und in die 2D-Faltungs-, 2D-Batchnorm-, Dropout-Layer und Relu-Aktivierungsfunktion eingespeist. Schließlich werden die Feature-Vektoren der beiden Submodelle nach dem Flatten zu einer fully-connected Layer mit einem Dropout verkettet, der zur Vorhersage der Outputs verwendet wird, der in diesem Fall die RUL der Zeitreihe und das entsprechende Spektrogramm ist. Abbildung B.3 beschreibt dieses Modell im Detail, das 5.3 Millionen lernbare Parameter umfasst. Dieser Ansatz basiert auch auf überwachtem Lernen. Tabelle C.3 und C.4 zei-

gen jeweils die Ergebnisse und die besten Hyperparameter davon. Bei diesem Ansatz zeigt CWT die beste Leistung, deren Ergebnisse in der Tabelle fett geschrieben sind und deren Visualisierung in Abbildung C.2 dargestellt ist.

4.3.3. Pann

Bei diesem Ansatz wird ein spezielles künstliches neuronales Netzwerk namens Pre-trained Audio Neural Networks (PANN) für die Audioverarbeitung verwendet. Ähnlich wie EfficientNet-B0 hat Pann auch Erfolge erzielt, wenn es von Anfang an mit dem AudioSet-Datensatz trainiert und dann Fine-Tuning am ESC-50-Datensatz vorgenommen hat, wobei eine Genauigkeit von 94.7 % erreicht wurde [46]. Die ursprüngliche Idee dieses Ansatzes bestand darin, Fine-Tuning anzuwenden, aber die Ergebnisse waren selbst während der Kreuzvalidierung wirkungslos. Der Grund dafür ist, dass PANN zuvor anhand eines Datensatzes trainiert wurde, der aus Audiodaten in YouTube-Videos gesammelt wurde und sich von dem in dieser Arbeit verfügbaren Datensatz unterscheidet.

Dieser Ansatz unterscheidet sich von den beiden oben genannten Ansätzen dadurch, dass das künstliche neuronale Netzwerk nur Zeitreihen als Inputs nimmt. Während des Modelltrainings werden nicht nur die Zeitreihenmerkmale, sondern auch das Log-Mel-Spekrogramm, das ihren vorab ausgewählten Hyperparametern entspricht, in die faltenden Layer eingespeist. Die Hauptstruktur von PANN ist die gleiche wie bei anderen Deep Vision Neural Networks, sie umfasst Faltungs-, Pooling-, BatchNorm-Layer, ReLU als Aktivierungsfunktion und Dropout. Strukturen wie Residual-Block oder Variationen mit ähnlichen Funktionen werden nicht angewendet. Die Struktur von PANN wird im Detail in Abbildung B.4 dargestellt. In dieser Arbeit sind die verfügbaren Eingänge jedoch vertikale und horizontale Vibrationssignale. Daher wird dieses Modell an die RUL-Bestimmung angepasst, das umfasst 156 Millionen zu erlernende Parameter umfasst und von Grund auf trainiert wird. Dies ist in Abbildung B.5 dargestellt. Tabelle C.5 und C.6 zeigen die Ergebnisse und die besten Hyperparameter dieses Ansatzes. Die Visualisierung der vorhersagten RUL kann durch Abbildung C.3 ausgedrückt werden.

4.3.4. **Tcl_im_CNN**

Bei diesem Ansatz wird Temporal Cycle-Consistence Learning (TCL) zur Bestimmung von RUL angewendet, da sowohl für die Trainings- als auch für die Testdatensätze Run-to-Failure-Zeitreihen verfügbar sind. Dies ist auch eine notwendige Voraussetzung, um künstliche neuronale Netze auf diese Weise zu trainieren. Es ist auch ersichtlich, obwohl die Komplexität sowie die Anzahl der lernfähigen Gewichte der Modelle erhöht wurden, haben die oben genannten drei überwachten Lernansätze ein großes Problem mit dem Datenmangel, wenn die Testdaten 2.5-mal mehr als die Trainingsdaten sind, Dies ist auch die Schwäche von Deep-Learning-Algorithmen im Allgemeinen. TCL hat einen herausragenden Vorteil beim Auffinden von Ähnlichkeiten der Inputs, insbesondere der

Degradierung der Lager zu einem bestimmten Zeitpunkt, was durch den ersten vorhergesagten Zeitpunkt (FPT) ausgedrückt werden kann, der in Abschnitt 3.4.2 vorgestellt wurde. Darüber hinaus kann TCL die zeitliche Position der Probe in der Run-to-Failure-Zeitreihe erfassen, die die oben genannten drei Ansätze nicht haben.

Theoretisch muss das Modell zur Anwendung von TCL den Input als die Run-to-Failure-Sequenz erhalten, um den Zeitpunkt der Degradierung ermitteln zu können. Dies ist jedoch in der Praxis nicht umsetzbar, da es sehr viele Ressourcen verbraucht, vornehmlich den VRAM der GPU. Um dieses Problem zu lösen, wird eine Lösung vorgeschlagen. Zuerst werden die Daten während der Vorverarbeitung gefenstert, wobei *hop_size* auf 25600 erhöht wird und *window_size* von 25600 bleibt, was bedeutet, dass es keine Überlappung zwischen zwei benachbarten Fenstern gibt. Dies führt dazu, dass die Run-to-Failure-Sequenz im Vergleich zu Abschnitt 4.2 kürzer werden, was in Tabelle 4.5 dargestellt werden kann. Diese Sequenzen werden dann wie Tabelle 4.4 in Spektralbilder umgewandelt.

Tabelle 4.5.: Anzahl der Proben nach der Fensterung vom Ansatz Tcl_im_CNN

Datasets	Name	N_windows
Train data	Bearing1_1	280
	Bearing1_2	87
	Bearing2_1	91
	Bearing2_2	79
	Bearing3_1	51
	Bearing3_2	163
Test data	Bearing1_3	237
	Bearing1_4	142
	Bearing1_5	246
	Bearing1_6	244
	Bearing1_7	225
	Bearing2_3	195
	Bearing2_4	75
	Bearing2_5	231
	Bearing2_6	70
	Bearing2_7	23
	Bearing3_3	43

Während des Trainings gibt es für jede *epoch* insgesamt $C_2^{N_{train}}$ Iteration, was die Anzahl der Paare zweier Sequenzen im Trainingsdatensatz darstellt, wobei N_{train} die Anzahl der Trainingslager ist. Bei jeder Iteration werden 2 Run-to-Failure-Sequenzen zufällig von 2 Lagern ausgewählt, und es werden *batch_size* zufällige Proben der beiden Sequenzen genommen, die in das Modell einbezogen werden sollen. Dieser Vorgang wird $\lceil \frac{N_{samples}}{batch_size} \rceil$ wiederholt, sodass alle Proben mindestens einmal vom Modell gesehen werden. Der Gradient wird in jedem Batch zweimal durch die beiden Verlustfunktionen $L_{cyc}^{(U,V)}$ und $L_{cyc}^{(V,U)}$ wie in Formel 3.28 aktualisiert. Wenn $N_{samples}$ kleiner als *batch_size* ist, wird

die gesamte Sequenz ausgewählt.

Zur Optimierung von Hyperparametern wird auch die Kreuzvalidierung verwendet, allerdings wird k gleich 6 gewählt, wobei 5 Lager zum Trainingsdatensatz gehören und ein verbleibender Lager aus dem Validierungsdatensatz stammt. Die Struktur des Modells wird die gleiche wie die von EfficientNet-B0 sein. Der letzte fully-connected Layer des Modells verfügt über *embedding_dim* Neuronen, was die Dimensionen des Feature-Vektors in Feature-Raum darstellt. Es ist zu beachten, dass es bei diesem Ansatz drei zusätzliche Hyperparameter gibt, die *embedding_dim*, *n_FPT* und *n_period* sind. Um die in dieser Arbeit verfügbaren Ressourcen unterzubringen, werden *batch_size* auf 100 und *epoch* auf 50 festgelegt. Bei diesem Ansatz sind zwei Grafikkarten erforderlich, um das Modell zu trainieren, was doppelt so viel ist wie bei den anderen drei Ansätzen.

5. Zusammenfassung

In dieser Arbeit werden die Zeitreihen zu Bildern umgewandelt, mit der Erwartung, mehr Informationen über den Gesundheitszustand der Lager zu liefern. Insgesamt werden 6 Zeitreihen-zu-Bild-Konvertierung untersucht, darunter Kurzzeit-Fourier-Transformation, Log-Mel-Spektrogramm, kontinuierliche Wavelet-Transformation, Rekurrenzplot, Gramian-Winkelfeld und Markov-Übergangsfeld. Die ersten drei Methoden stellen Zeitreihen sowohl im Zeit- als auch im Frequenzbereich dar, während sich die verbleibenden drei Konvertierungen auf die Beziehung zwischen Datenpunkten in der Zeitreihe konzentrieren. Durch die Verwendung erwarteter Bilder können RUL besser vorhergesagt werden, indem die Leistungsfähigkeit von Deep Vision Neural Networks genutzt wird.

Für den in dieser Arbeit verwendeten Datensatz werden vier Ansätze vorgeschlagen. Die ersten drei Methoden basieren hauptsächlich auf überwachtem Lernen, bei dem RUL direkt aus dem Modell vorhergesagt wird. Obwohl der zweite Ansatz die Ergebnisse im Validierungsdatensatz verbesserte, erzielten alle drei im Allgemeinen nur bescheidene Ergebnisse im Testdatensatz. Der Grund dafür ist, dass der Trainingsdatensatz zu klein ist und ein Teil der Testdaten nicht in derselben Domäne wie die Trainingsdaten liegt. Dies ist auch die Schwäche von Deep-Learning-Algorithmen im Allgemeinen. Daher wurde ein vierter Ansatz vorgeschlagen, der unbeaufsichtigtes Lernen anwendet, um die Korrelation zwischen Zeitreihen zu ermitteln, insbesondere hier die Verschlechterungszeit von Lagern. Allerdings wurde diese Idee erst eine Woche vor Ende des Arbeitszeitraums umgesetzt und die Intervalle der Hyperparameter wurden bei der Optimierung nicht richtig ausgewählt, sodass die Ergebnisse nicht vorliegen. Wenn genügend Zeit für weitere Forschung bleibt, wird erwartet, dass Temporal Cycle-Consistency Learning im Vergleich zu anderen Ansätzen des überwachten Lernens immer noch vielversprechende Ergebnisse liefert. Basierend auf den verfügbaren Ergebnissen bieten STFT, LMS und CWT die besten Leistungen.

Literaturverzeichnis

- [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [2] Wikimedia Commons. File:rubik's cube resolved, solo.svg — wikimedia commons, the free media repository, 2023. [Online; accessed 28-August-2023].
- [3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [5] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [6] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [7] Vladimir Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62, 2017.
- [8] SGOPAL Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.
- [9] Sun-Chong Wang and Sun-Chong Wang. Artificial neural network. *Interdisciplinary computing in java programming*, pages 81–100, 2003.
- [10] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.
- [11] Daniel Bashir, George D Montañez, Sonia Sehra, Pedro Sandoval Segura, and Julius Lauw. An information-theoretic perspective on overfitting and underfitting. In *AI 2020: Advances in Artificial Intelligence: 33rd Australasian Joint Conference, AI 2020, Canberra, ACT, Australia, November 29–30, 2020, Proceedings 33*, pages

- 347–358. Springer, 2020.
- [12] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)*, pages 1836–1841. IEEE, 2018.
 - [13] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.
 - [14] Pytorch conv2d. <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>. Accessed: September 24, 2023.
 - [15] ML Cheat Sheet. ML Cheat Sheet - Loss Functions. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html, 2017. Accessed: August 29, 2023.
 - [16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
 - [17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
 - [18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
 - [19] scikit-learn contributors. scikit-learn: Machine learning in python. <https://scikit-learn.org/stable>, 2023. Accessed: August 29, 2023.
 - [20] Daniel Berrar et al. Cross-validation., 2019.
 - [21] Wikipedia contributors. Fine-tuning (deep learning) — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Fine-tuning_\(deep_learning\)&oldid=1168828327](https://en.wikipedia.org/w/index.php?title=Fine-tuning_(deep_learning)&oldid=1168828327), 2023. [Online; accessed 29-August-2023].
 - [22] Chonghui Guo, Hailin Li, and Donghua Pan. An improved piecewise aggregate approximation based on statistical features for time series mining. In *Knowledge Science, Engineering and Management: 4th International Conference, KSEM 2010, Belfast, Northern Ireland, UK, September 1-3, 2010. Proceedings 4*, pages 234–244. Springer, 2010.
 - [23] Duraisamy Sundararajan. *The discrete Fourier transform: theory, algorithms and applications*. World Scientific, 2001.
 - [24] MSSM Basir, RC Ismail, KH Yusof, NIA Katim, MNM Isa, and SZM Naziri. An implementation of short time fourier transform for harmonic signal detection. In *Journal of Physics: Conference Series*, volume 1755, page 012013. IOP Publishing, 2021.

- [25] mlearnere. Learning from audio: The mel scale, mel spectrograms, and mel frequency cepstral coefficients. <https://towardsdatascience.com/learning-from-audio-the-mel-scale-mel-spectrograms-and-mel-frequency-cepstral-coefficients-2021>. [Online; accessed 29-August-2023].
- [26] Wikipedia contributors. Mel scale — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Mel_scale&oldid=1170474440, 2023. [Online; accessed 29-August-2023].
- [27] Luís Aguiar-Conraria and Maria Joana Soares. The continuous wavelet transform: Moving beyond uni-and bivariate analysis. *Journal of Economic Surveys*, 28(2):344–375, 2014.
- [28] Wikipedia contributors. Convolution — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1171637764>, 2023. [Online; accessed 29-August-2023].
- [29] PyWavelets contributors. PyWavelets: Wavelet transforms in python. <https://pywavelets.readthedocs.io/en/latest/ref/cwt.html>, Year. Accessed: August 29, 2023.
- [30] Marco Thiel, M Carmen Romano, and Jürgen Kurths. How much information is contained in a recurrence plot? *Physics Letters A*, 330(5):343–349, 2004.
- [31] Zhiguang Wang and Tim Oates. Imaging time-series to improve classification and imputation. *arXiv preprint arXiv:1506.00327*, 2015.
- [32] Wai-Ki Ching and Michael K Ng. Markov chains. *Models, algorithms and applications*, pages 1–18, 2006.
- [33] Wikipedia contributors. Stochastic matrix — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Stochastic_matrix&oldid=1172743157, 2023. [Online; accessed 29-August-2023].
- [34] pyts contributors. pyts: A python package for time series classification. <https://pyts.readthedocs.io/en/stable/generated/pyts.image.MarkovTransitionField.html#pyts.image.MarkovTransitionField>, Year. Accessed: August 29, 2023.
- [35] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Temporal cycle-consistency learning, 2019.
- [36] Xiang Li, Wei Zhang, Hui Ma, Zhong Luo, and Xu Li. Degradation alignment in remaining useful life prediction using deep cycle-consistent learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(10):5480–5491, 2021.
- [37] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, Masahiro Nomura, and Masaaki Onishi. Multiobjective tree-structured parzen estimator. *Journal of Artificial Intelligence Research*, 73:1209–1250, 2022.

- [38] Patrick Nectoux, Rafael Gouriveau, Kamal Medjaher, Emmanuel Ramasso, Brigitte Chebel-Morello, Noureddine Zerhouni, and Christophe Varnier. Pronostia: An experimental platform for bearings accelerated degradation tests. In *IEEE International Conference on Prognostics and Health Management, PHM'12.*, pages 1–8. IEEE Catalog Number: CPF12PHM-CDR, 2012.
- [39] Edwin Sutrisno, Hyunseok Oh, Arvind Sai Sarathi Vasan, and Michael Pecht. Estimation of remaining useful life of ball bearings using data driven methodologies. In *2012 ieee conference on prognostics and health management*, pages 1–7. IEEE, 2012.
- [40] Sheng Hong, Zheng Zhou, Enrico Zio, and Kan Hong. Condition assessment for the performance degradation of bearing based on a combinatorial feature extraction method. *Digital Signal Processing*, 27:159–166, 2014.
- [41] Yaguo Lei, Naipeng Li, Szymon Gontarz, Jing Lin, Stanislaw Radkowski, and Jacek Dybala. A model-based method for remaining useful life prediction of machinery. *IEEE Transactions on reliability*, 65(3):1314–1326, 2016.
- [42] Sangram Patil, Aum Patil, Vishwadeep Handikherkar, Sumit Desai, Vikas M Phalle, and Faruk S Kazi. Remaining useful life (rul) prediction of rolling element bearing using random forest and gradient boosting technique. In *ASME international mechanical engineering congress and exposition*, volume 52187, page V013T05A019. American Society of Mechanical Engineers, 2018.
- [43] Sumyung Gang, Ndayishimiye Fabrice, Daewon Chung, and Joonjae Lee. Character recognition of components mounted on printed circuit board using deep learning. *Sensors*, 21(9):2921, 2021.
- [44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [45] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [46] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumley. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2880–2894, 2020.

A. Visualisierung

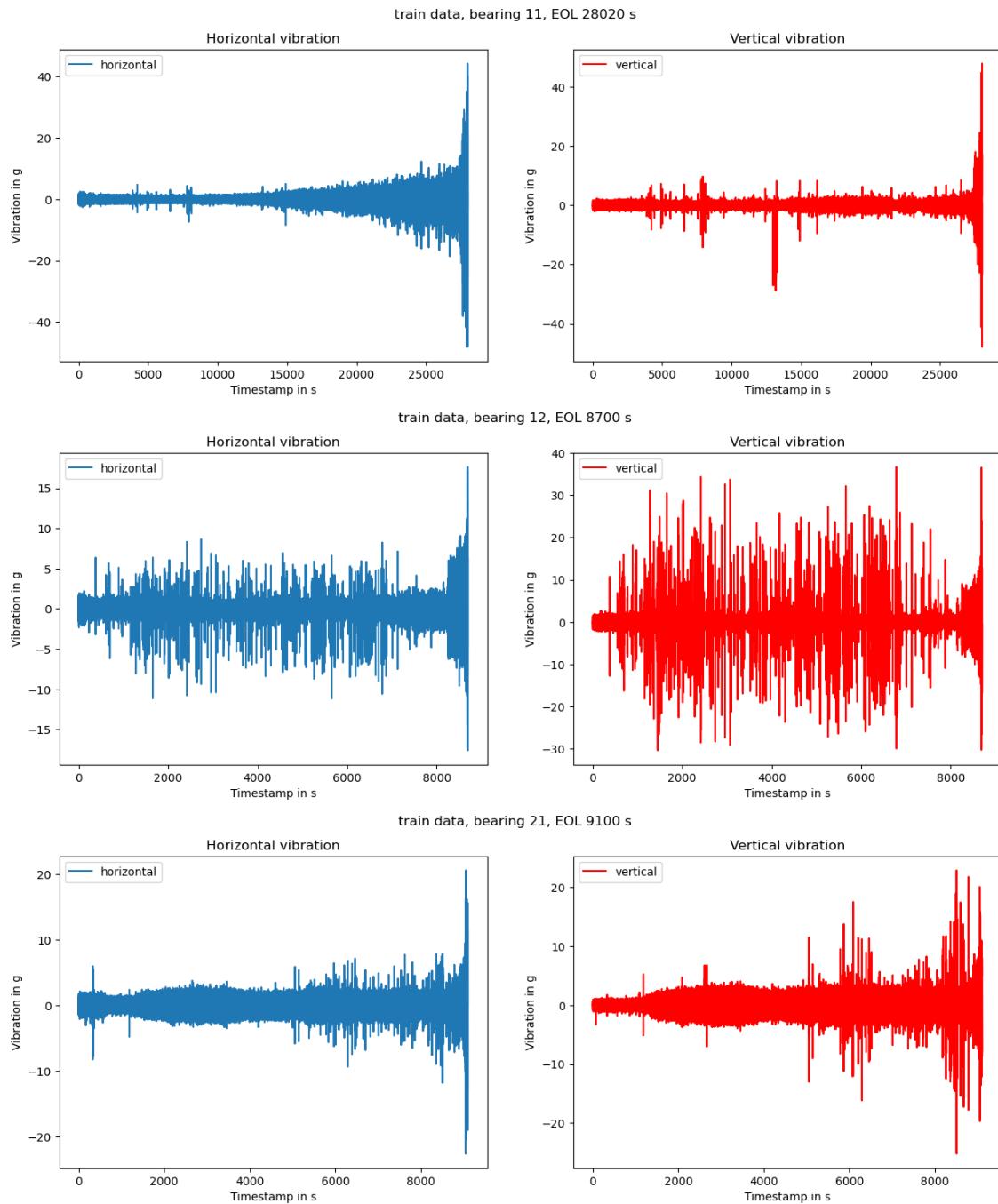


Abbildung A.1.: Visualisierung von Bearing1_1, Bearing1_2 und Bearing2_1

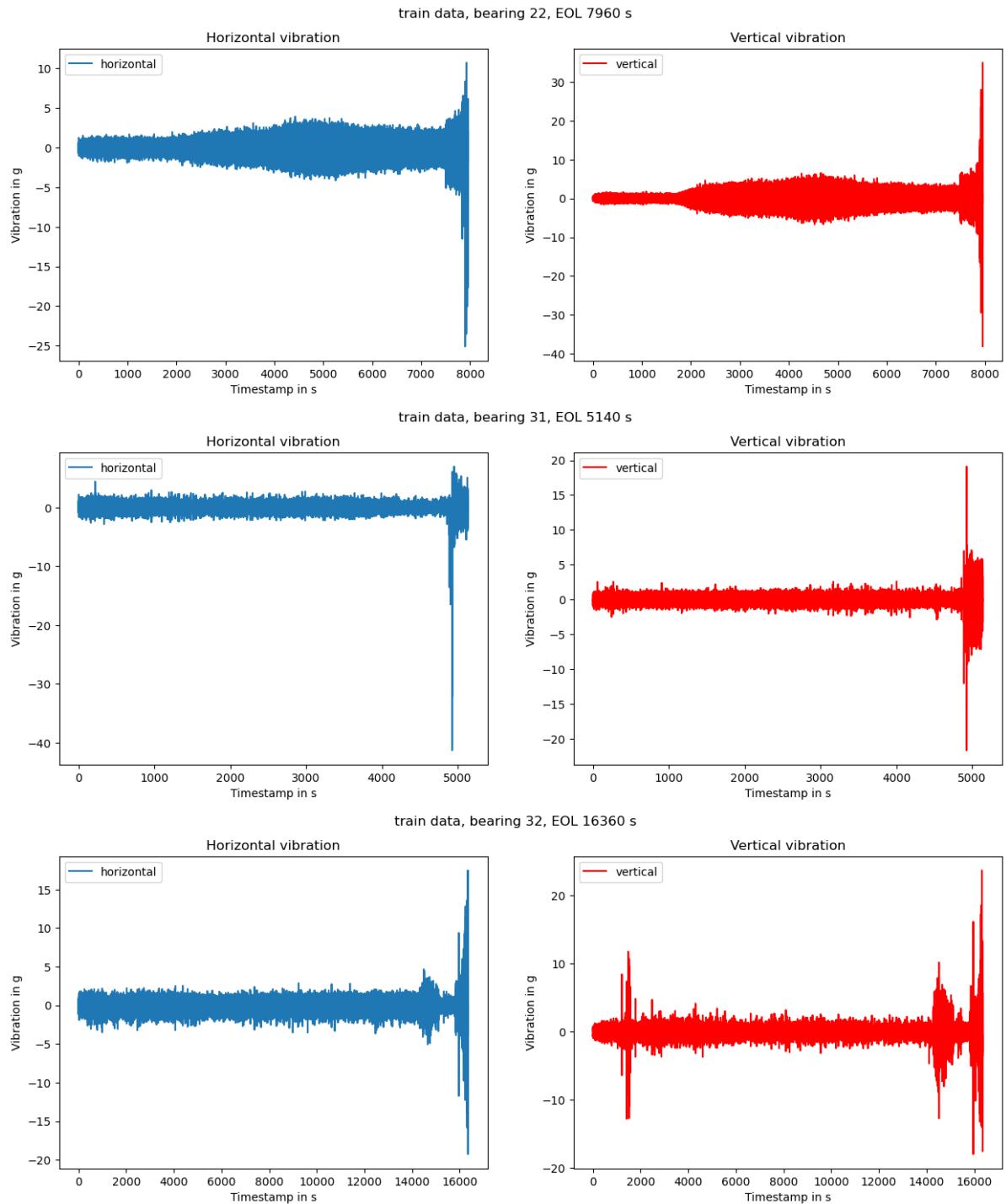


Abbildung A.2.: Visualisierung von Bearing2_2, Bearing3_1 und Bearing3_2

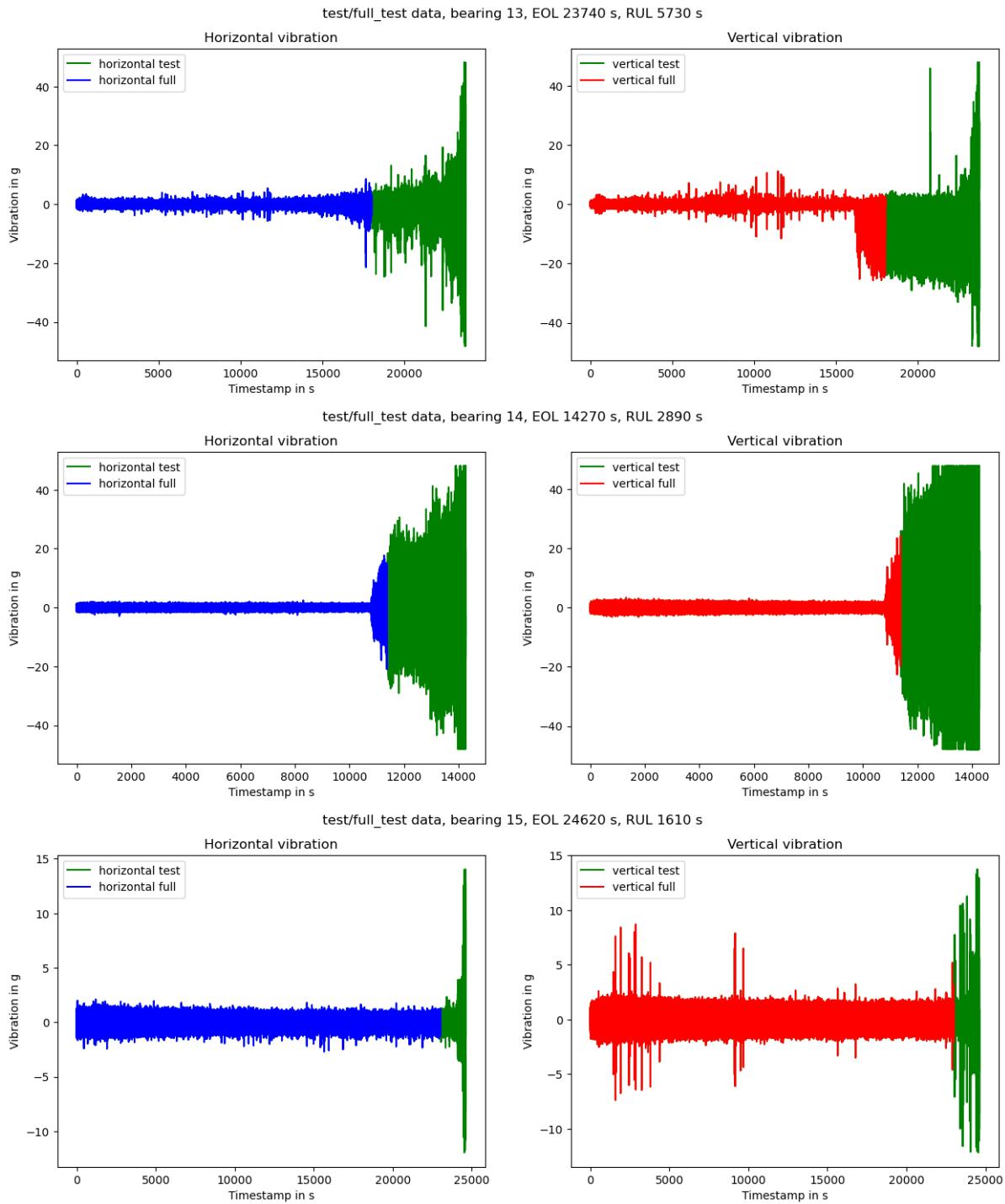


Abbildung A.3.: Visualisierung von Bearing1_3, Bearing1_4 und Bearing1_5

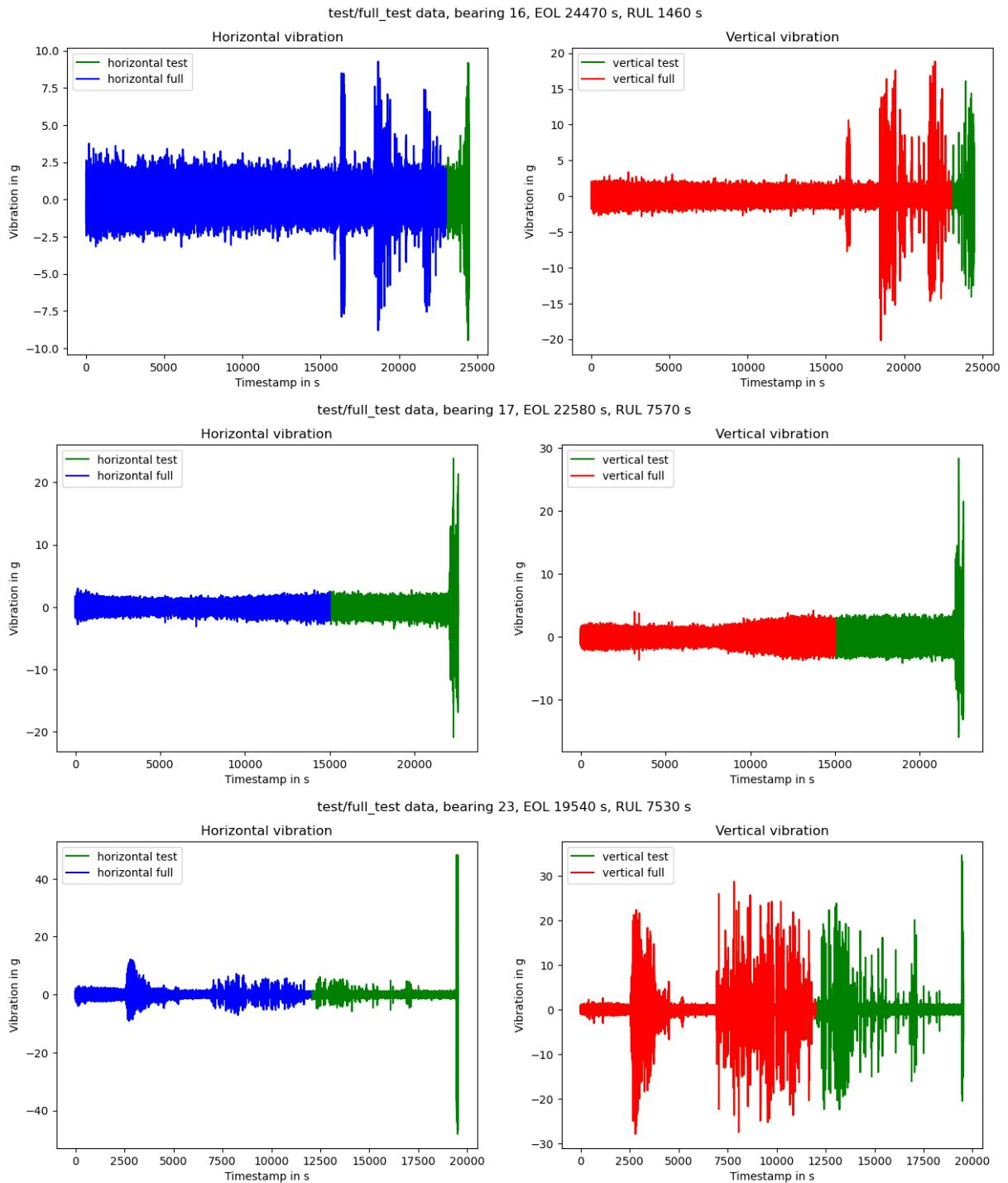


Abbildung A.4.: Visualisierung von Bearing1_6, Bearing1_7 und Bearing2_3

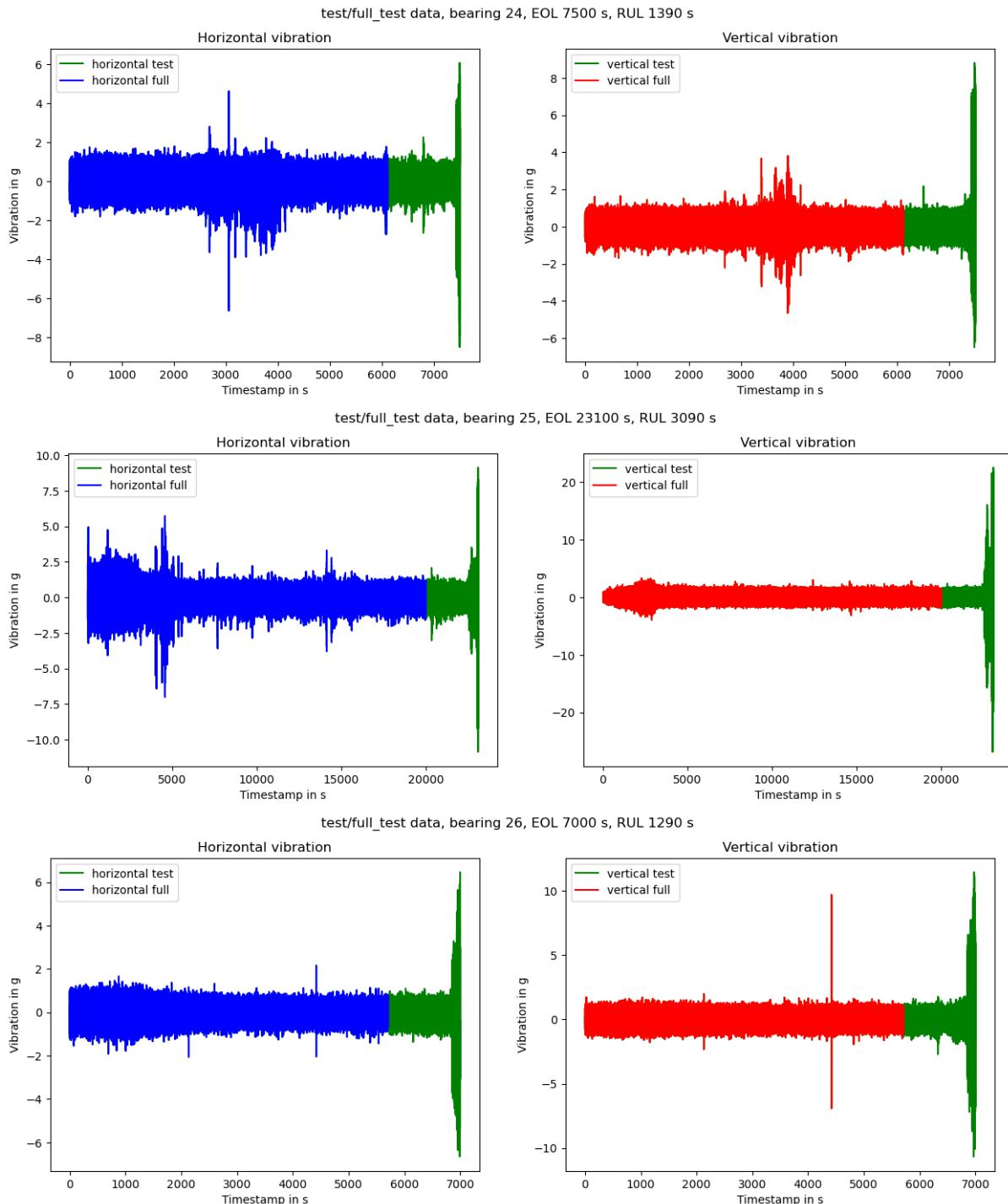


Abbildung A.5.: Visualisierung von Bearing2_4, Bearing2_5 und Bearing2_6

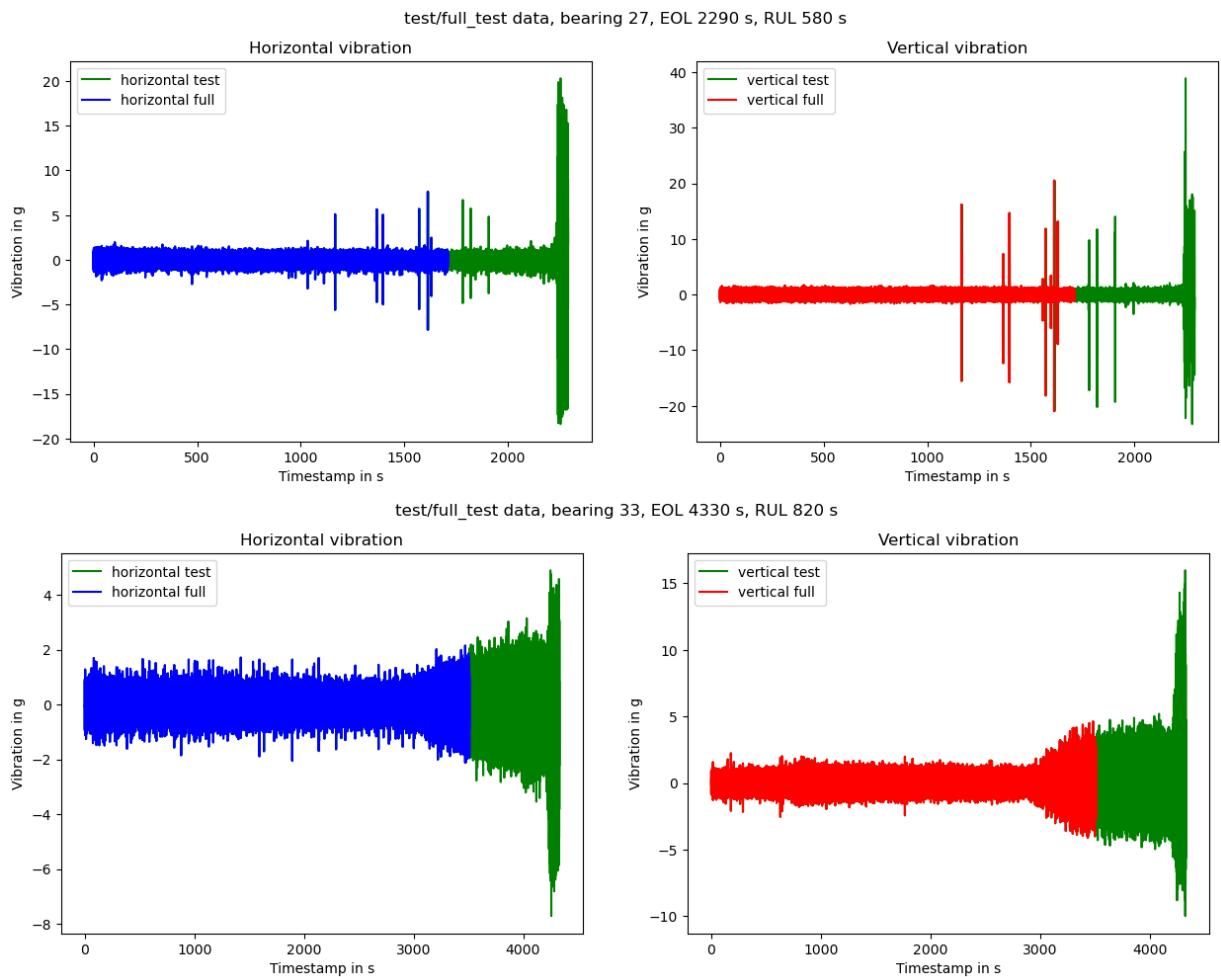


Abbildung A.6.: Visualisierung von Bearing2_7 und Bearing3_3

B. Modellstruktur

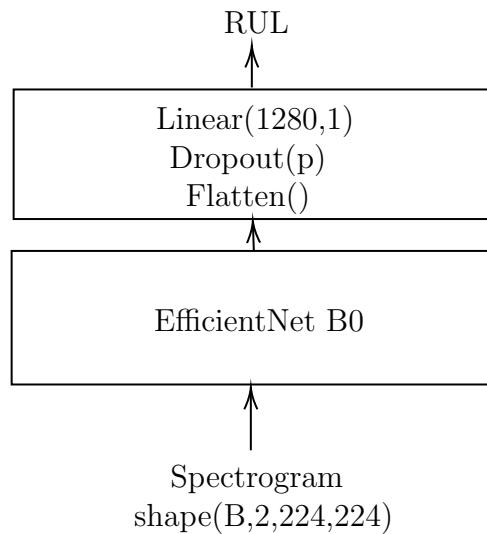


Abbildung B.1.: Modellstruktur vom Ansatz Im_CNN

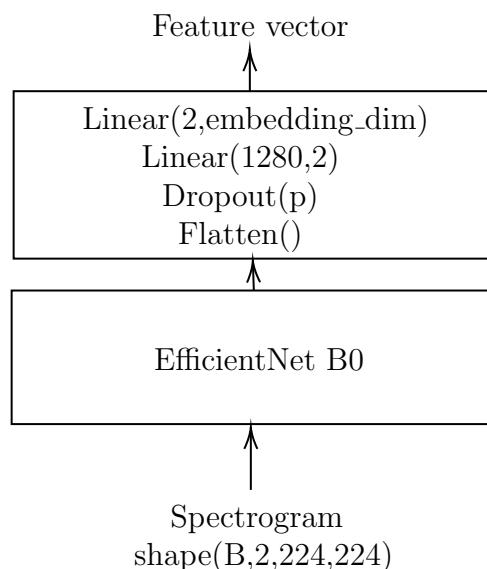
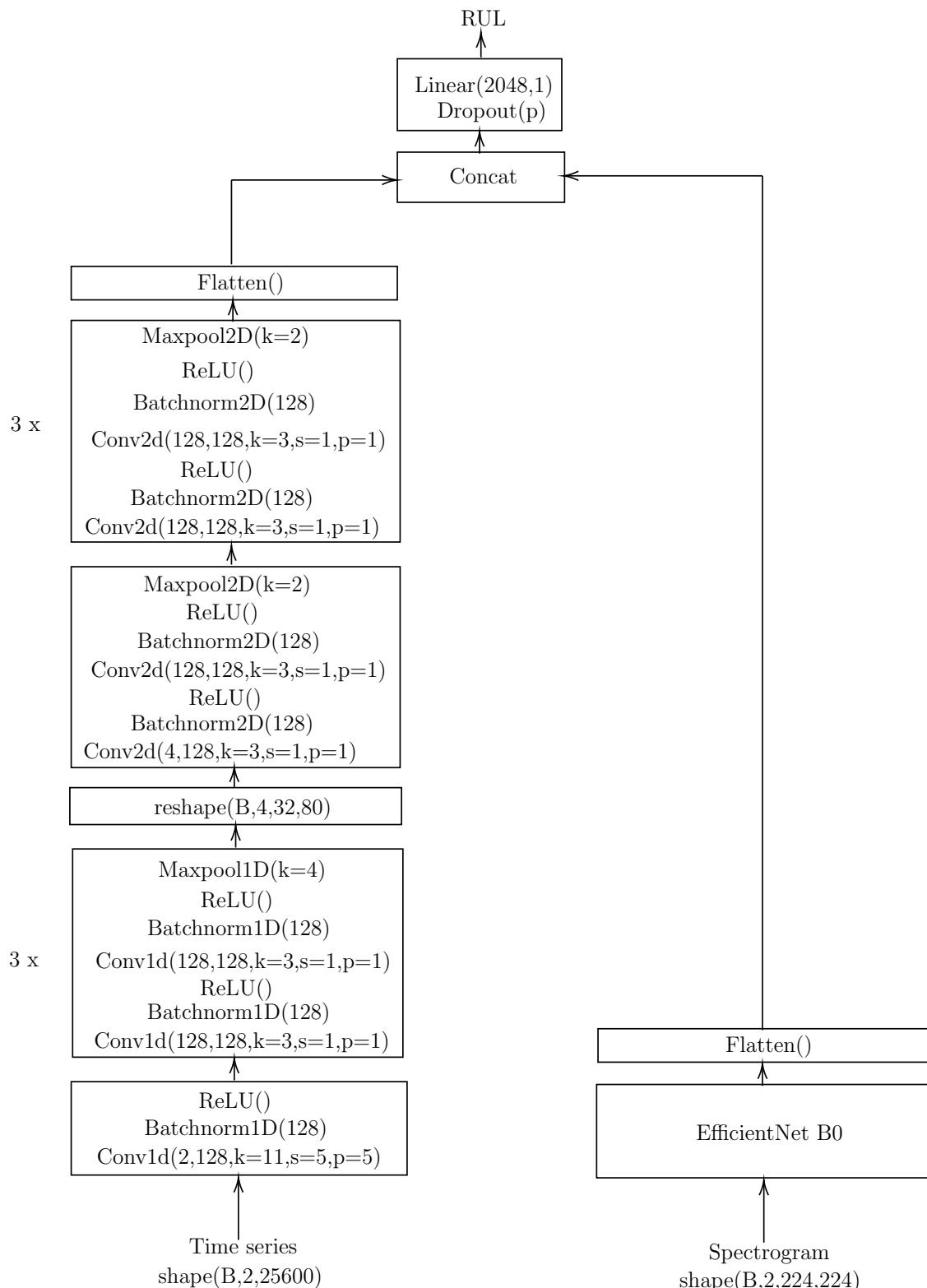


Abbildung B.2.: Modellstruktur vom Ansatz Tcl_im_CNN

Abbildung B.3.: Modellstruktur vom Ansatz `Ts_im_CNN`

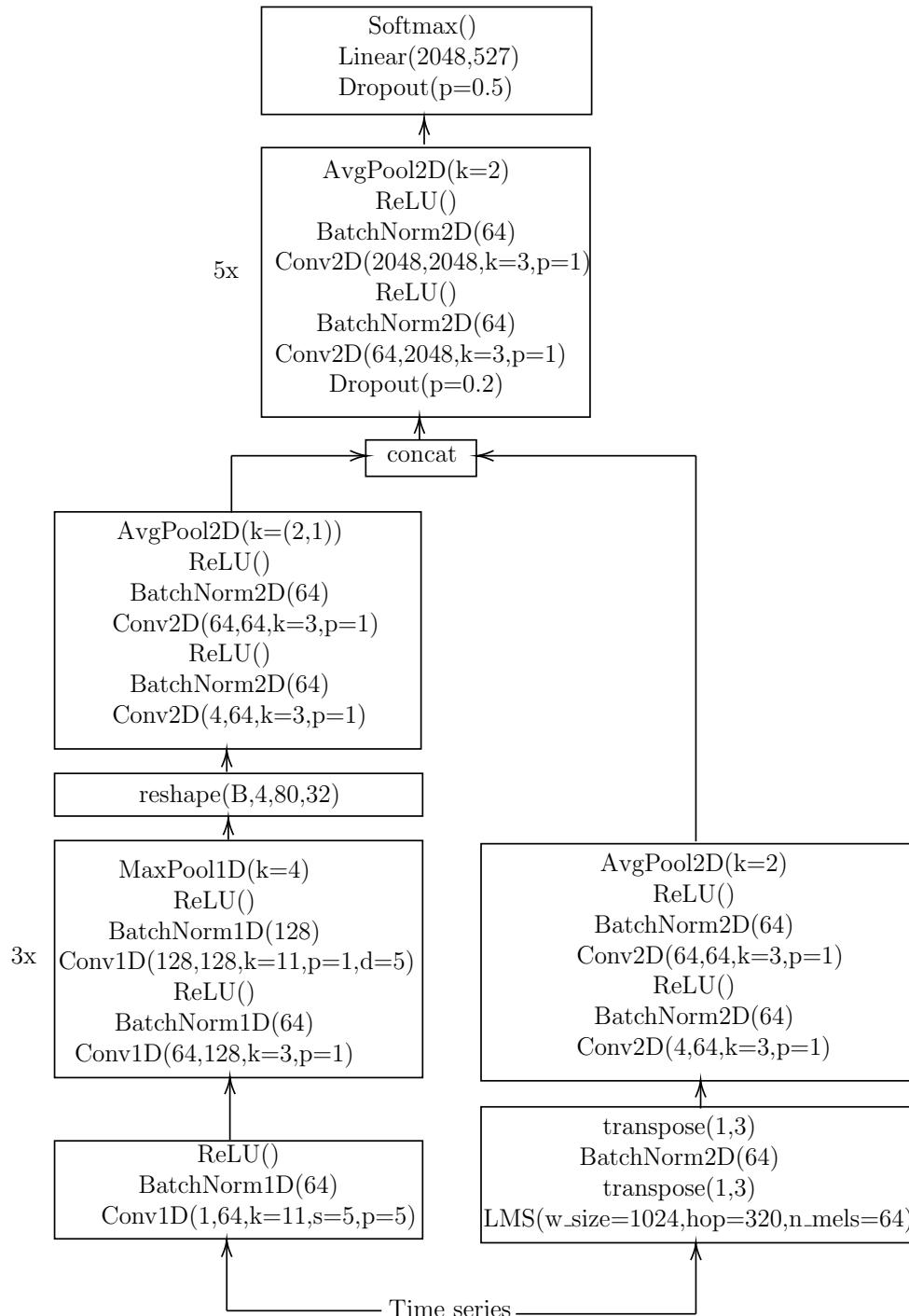


Abbildung B.4.: Modellstruktur vom PANN [46]

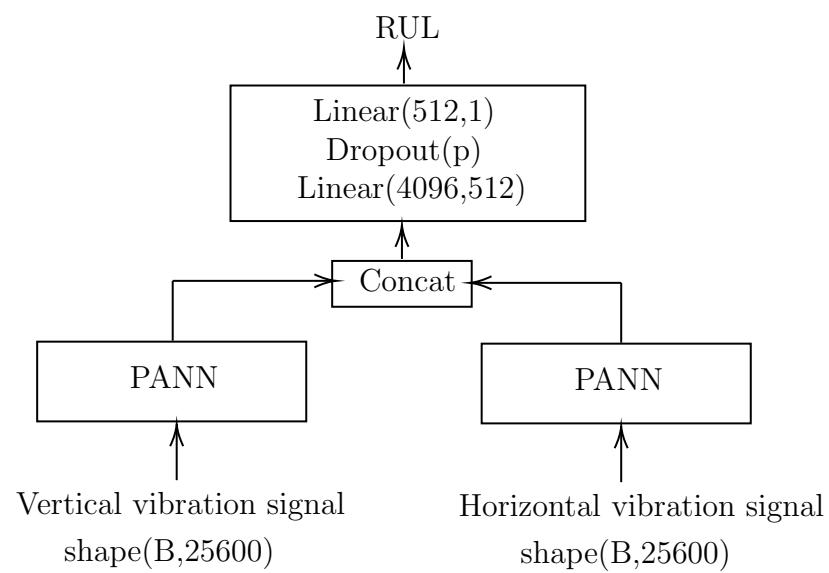


Abbildung B.5.: Modellstruktur vom Ansatz Pann

C. Ergebnisse

Tabelle C.1.: Ergebnisse von Methode Im-CNN

Method	Result				
	loss train	score train	loss test	score test	final score
STFT	0.06	0.49	1.22	0.31	0.05
LMS	0.01	0.79	0.23	0.21	0.11
CWT	0.01	0.74	0.21	0.25	0.03
RP	0.04	0.55	0.40	0.15	0.02
GAF	0.04	0.31	0.58	0.20	0.01
MTF	0.05	0.50	0.29	0.20	0.01
Method	Result CV				
	loss train	score train	loss test	score test	final score
STFT	-	-	0.15	0.37	0.14
LMS	-	-	0.17	0.32	0.08
CWT	-	-	0.19	0.31	0.10
RP	-	-	0.28	0.21	0.07
GAF	-	-	0.27	0.22	0.04
MTF	-	-	0.30	0.20	0.00

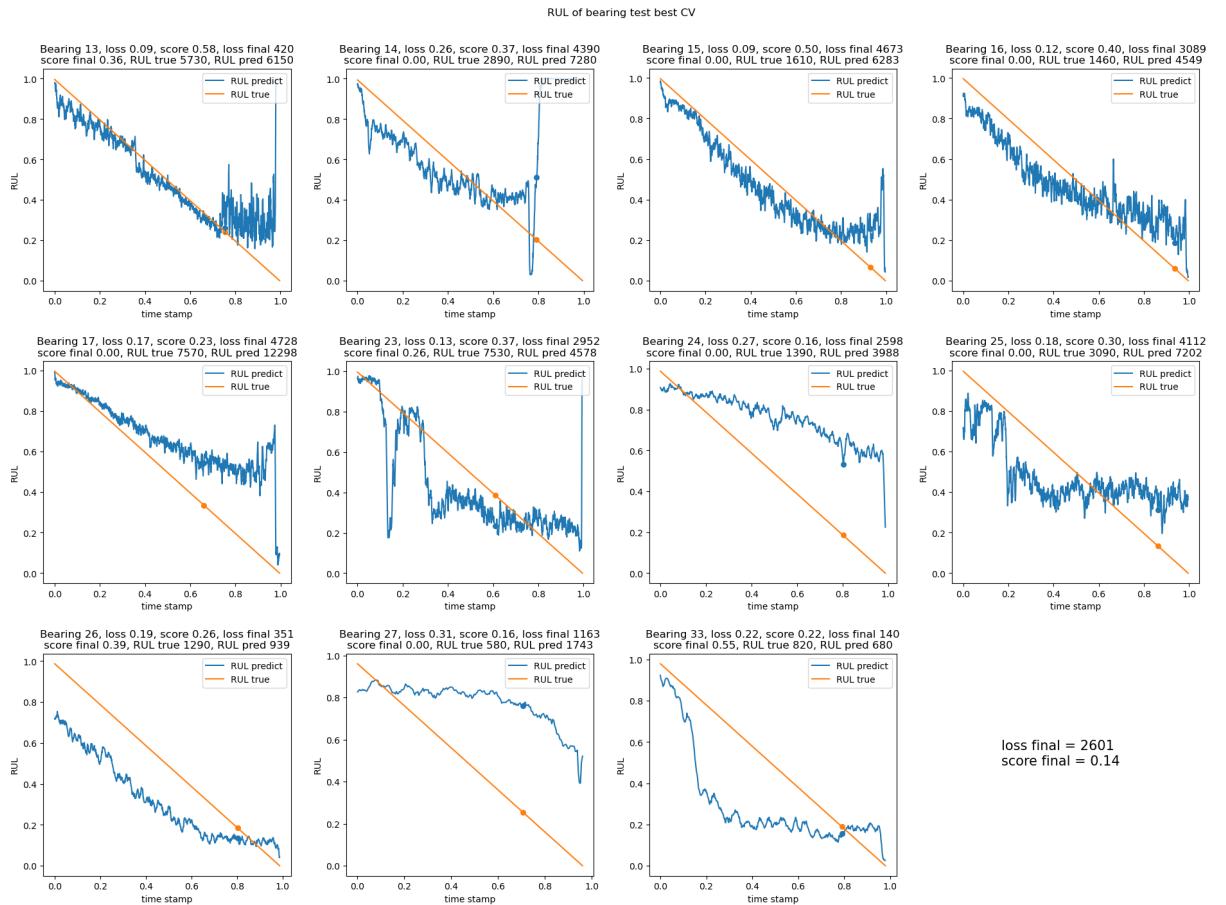


Abbildung C.1.: Visualisierung des besten Ergebnisses vom Ansatz Im_CNN

Tabelle C.2.: Kreuzvalidierungsergebnisse und beste Hyperparameter vom Ansatz Im_CNN

Method	Cross validation				Hyperparameter
	loss train	score train	loss val	score val	
STFT	0.05	0.52	0.03	0.65	dropout = 0.5389 weight_decay = 0.0307 lr = 0.0012 optimizer = SGD window_size = 3328 hop_size = 32
LMS	0.01	0.75	0.01	0.80	dropout = 0.1283 hop_size = 32 lr = 0.0005 optimizer = Adam window_size = 5120 n_mels = 306 weight_decay = 1.1319e-5
CWT	0.05	0.49	0.04	0.56	dropout = 0.3 weight_decay = 0.0032 lr = 0.00057 optimizer = Adam b = 2.05 fc = 2.4 scales = (0.9,40,195)
RP	0.04	0.51	0.07	0.37	dropout = 0.1243 threshold = None lr = 0.0580 optimizer = SGD time_delay = 1 dimension = 1 weight_decay = 0.0307
GAF	0.04	0.55	0.08	0.41	dropout = 0.7491 weight_decay = 0.0307 lr = 0.0005 optimizer = Adam method = difference
MTF	0.05	0.48	0.13	0.34	dropout = 0.2461 strategy = quantile lr = 0.0557 optimizer = Adam n_bins = 5 weight_decay = 1.7309e-5

Tabelle C.3.: Ergebnisse vom Ansatz Ts_im_CNN

Method	Result				
	loss train	score train	loss test	score test	final score
STFT	0.06	0.57	0.73	0.31	0.01
LMS	0.01	0.75	0.20	0.26	0.02
CWT	0.01	0.75	0.19	0.29	0.08
RP	0.05	0.57	0.31	0.31	0.01
GAF	0.05	0.57	0.55	0.28	0.01
MTF	0.05	0.52	0.25	0.32	0.03
Method	Result CV				
	loss train	score train	loss test	score test	final score
STFT	-	-	0.14	0.31	0.02
LMS	-	-	0.16	0.32	0.08
CWT	-	-	0.16	0.34	0.16
RP	-	-	0.14	0.34	0.10
GAF	-	-	0.14	0.29	0.07
MTF	-	-	0.15	0.34	0.08

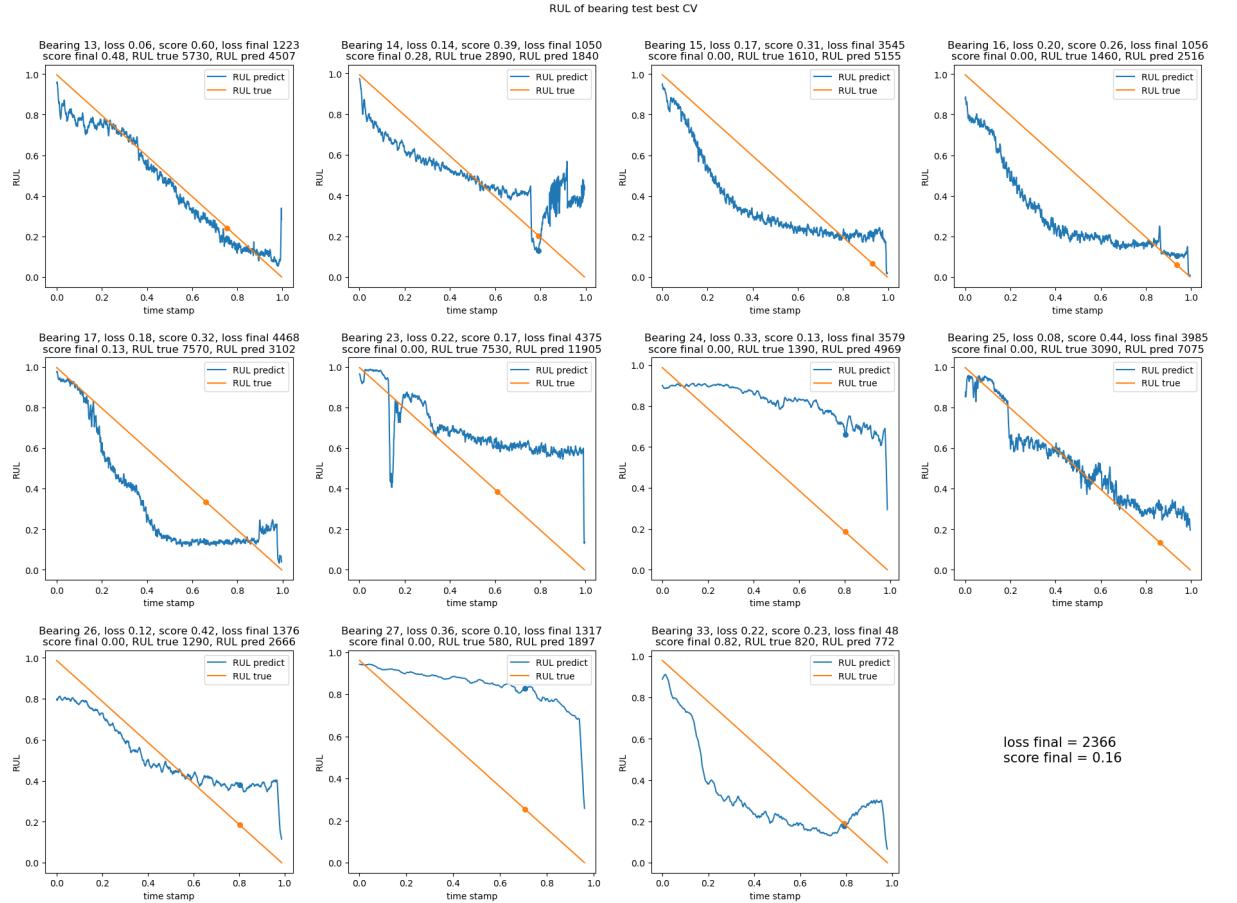


Abbildung C.2.: Visualisierung des besten Ergebnisses vom Ansatz Ts_im_CNN

Tabelle C.4.: Kreuzvalidierungsergebnisse und beste Hyperparameter vom Ansatz Tsim-CNN

Method	Cross validation				Hyperparameter
	loss train	score train	loss val	score val	
STFT	0.02	0.68	0.02	0.69	dropout = 0.3270 weight_decay = 0.0548 lr = 0.0027 optimizer = SGD window_size = 4352 hop_size = 256
LMS	0.01	0.74	0.01	0.78	dropout = 0.1906 hop_size = 32 lr = 0.0005 optimizer = Adam window_size = 5120 n_mels = 294 weight_decay = 1.1531e-5
CWT	0.03	0.64	0.02	0.68	dropout = 0.2065 weight_decay = 0.00014 lr = 0.0005 optimizer = Adam b = 2.9 fc = 2.35 scales = (1.3,40,100)
RP	0.02	0.69	0.02	0.69	dropout = 0.1299 threshold = distance percentage = 35 lr = 0.0020 optimizer = SGD time_delay = 1 dimension = 1 weight_decay = 0.0694
GAF	0.02	0.68	0.02	0.70	dropout = 0.3301 weight_decay = 0.0772 lr = 0.0021 optimizer = SGD method = summation
MTF	0.02	0.69	0.02	0.69	dropout = 0.1101 strategy = quantile lr = 0.0014 optimizer = SGD n_bins = 33 weight_decay = 0.0981

Tabelle C.5.: Ergebnisse vom Ansatz Pann

Method	Result				
	loss train	score train	loss test	score test	score final
LMS	0.03	0.66	0.14	0.32	0.19
Method	Result CV				
	loss train	score train	loss test	score test	score final
LMS	-	-	0.25	0.18	0.09

Tabelle C.6.: Kreuzvalidierungsergebnisse und beste Hyperparameter vom Ansatz Pann

Method	Cross validation				Hyperparameter
	loss train	score train	loss val	score val	
LMS	0.03	0.60	0.02	0.68	dropout = 0.1482 weight_decay = 0.0001 lr = 0.0003 optimizer = Adam

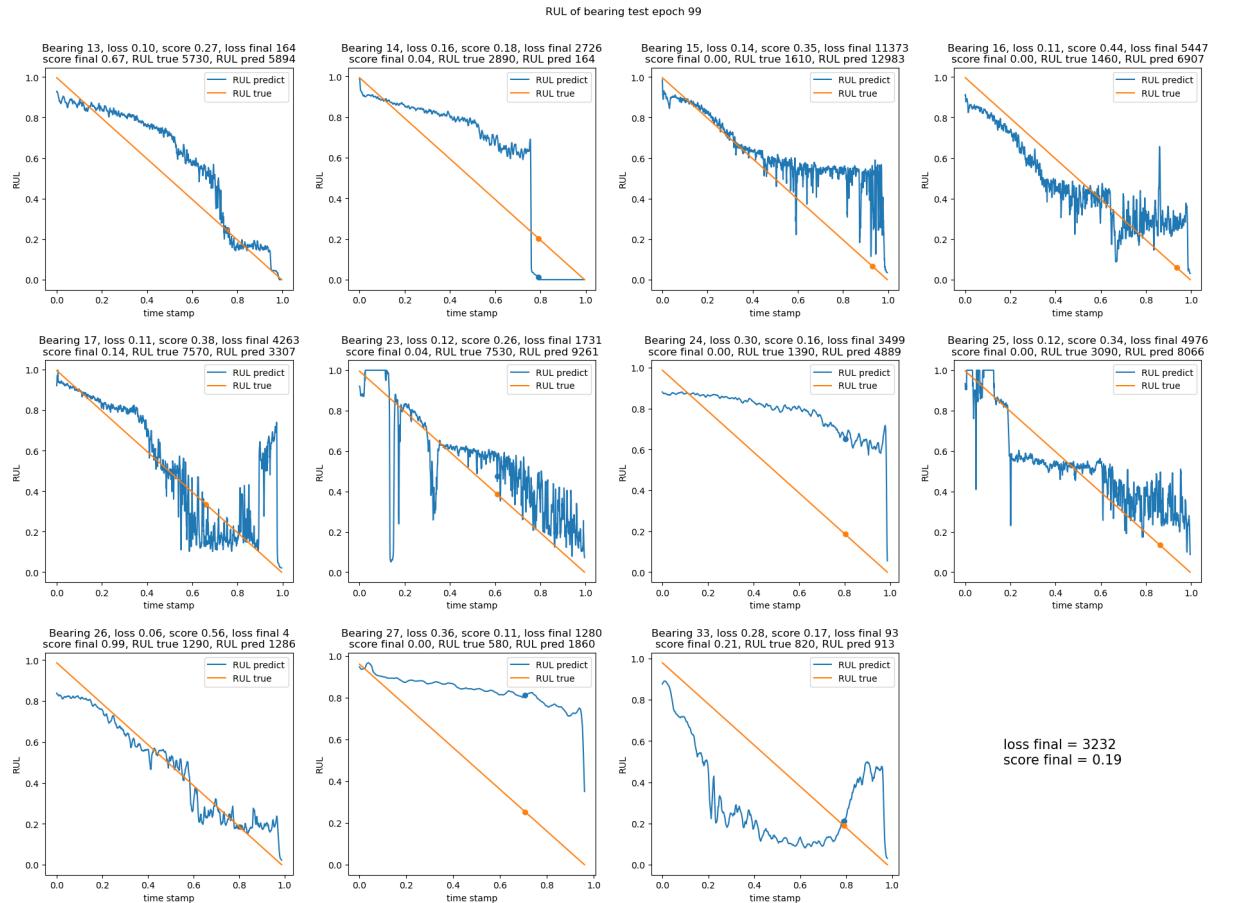


Abbildung C.3.: Visualisierung des besten Ergebnisses vom Ansatz Pann