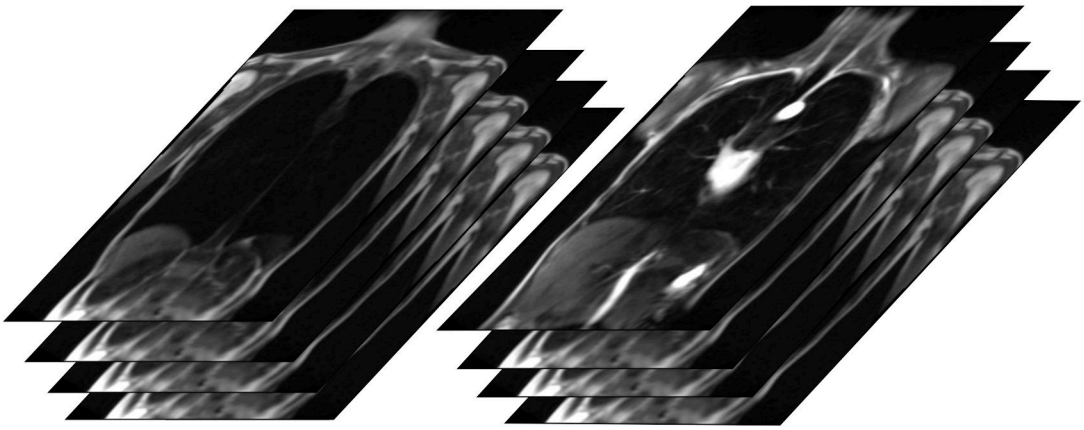
Hector Seminar

Karlsruher Institut für Technologie

KI-gestützte MRT-Bildanalyse



Abschlussdokumentation der Kooperationsphase 2022/23

Durchgeführt am MMS KIT
Betreut durch Dr. Arnd Koeppe und Julian Grolig

Fabian Habermann, Paul Schepperle, Kurs KA17

Inhaltsverzeichnis

Abstract

There are numerous diseases, all diagnosed as COPD, (Chronic Obstructive Pulmonary Diseases). In order to diagnose such a disease, an MRI-scan is often performed, to observe how a contrast agent (a simple injection that can be easily detected with an MRI) passes through the body, specifically the lungs of a suspected patient. If any parts of the lung are obstructed, the MRI-scan will reveal them, because the contrast agent does not pass through those parts of the lung. Due to the large size of an MRT-scan (approximately 65.000.000 voxels) the process of searching for potential issues is very time-consuming for the radiologist.

To simplify this process for radiologists, an artificial intelligence (AI) tool was developed to find the ideal timestamp. After this timestamp is identified, the radiologist can analyze the flow without having to search through two dimensions for the timestamp at which the fluid enters the lungs. The AI tool converts the large 4D scan into an array of 32 values (one for each timestamp) using various mathematical methods. This process is referred to as preprocessing. The preprocessed data is then evaluated by an neural network, that was trained with 515 real MRI-scans and their corresponding, manually evaluated timestamps.

1 Einleitung

Chronisch obstruktive Lungenkrankheiten (engl."chronic obstuctive pulmunary diseases"), kurz COPD, sind Lungenerkrankungen, bei denen die Atemwege chronisch verengt sind.[1]Dadurch verschlechtert sich die Sauerstoffaufnahme und die physische Leistungsfähigkeit der Erkrankten nimmt ab. In Deutschland leiden heute ca. 6,8 Millionen Menschen an einer COPD, etwa jeder vierte Erwachsene erkrankt im Laufe seines Lebens an COPD.[2] Zur Identifizierung von COPD werden MRT-Aufnahmen herangezogen, bei denen den Patienten vorher ein Kontrastmittel injiziert wird, welches auf den MRT-Aufnahmen deutlich aufgehellte Bereiche erzeugt. Auf diesen Aufnahmen wird unter anderem nach dunklen Stellen in der Lunge gesucht, welche schlecht durchblutet werden, aber auch die Fließgeschwindigkeit wird untersucht. Um diese Analyse zu vereinfachen, wird ein KI-Tool am Institut für Angewandte Materialien-Mikrostruktur-Modellierung und Simulation (IAM-MMS) am Karlsruher Institut für Technologie (KIT) entwickelt, die den Ankunftszeitpunkt des Kontrastmittels im Herzen automatisch bestimmt. Diese KI-Tool hat die Struktur eines neuronalen Netzes und wird mit Python programmiert.

2 Material und Methoden

2.1 Magnetresonanztomographie

2.1.1 Funktionsweise

Die Aufnahme der für das Training notwendigen Bilder stammen aus der multizentrischen, deutschlandweiten COPD-Studie COSYCONET.[3] Die Patienten werden dazu auf einer Liege mit dem ganzen Körper in einen Magnetresonanztomographen (MRT-Scanner,Abbildung 1) geschoben. Im Bereich des MRT-Scanners liegt ein sehr starkes Magnetfeld vor. Dabei richten sich die Spins der Wasserstoffkerne des menschlichen Gewebes in Richtung des Magnetfeldes aus, da sie sich selbst wie kleine Magnete verhalten. Mit Hilfe eines sehr starken, aber kurzen elektromagnetischen Pulses werden die Spins aus ihrer Gleichgewichtslage gebracht, in die sie sich nach kurzer Zeit wieder zurückbewegen.[4] Diese Relaxation kann mit Hilfe von Spulen gemessen werden, welche dann mit Hilfe eines aufwendigen Verfahrens zu dreidimensionalen Darstellungen zusammengesetzt werden.

Abbildung 1: MRT-Scanner[4]

Um die Durchblutung auf den Aufnahmen zu erkennen, verwendet man ein spezielles Verfahren, bei dem man den Patienten zusätzlich ein Kontrastmittel injiziert. Dieses Kontrastmittel erzeugt aufgehellte Bereiche auf den MRT-Aufnahmen. Da das Kontrastmittel den Blutkreislauf mitdurchläuft, kann man damit den Blutdurchfluss verschiedener Gefäße erkennen. Als Grundlage dienen in der vorliegenden Arbeit die MRT-Aufnahmen des Thorax-Bereichs aus der COSYCONET-Studie.

2.1.2 Datensätze

Es liegen 527 Aufnahmen verschiedener Patienten vor. Nicht von allen Aufnahmen liegen die Maximalzahl an Voxeln in allen Raumrichtungen vor. Zur Normierung der Datensatzdimensionen werden die fehlenden Werte dieser Voxel mit Null definiert. Die 527 , mit Nullen aufgefüllten, Aufnahmen haben jeweils ein Format von $46 \cdot 60 \cdot 256 \cdot 256$ Voxel (ein Voxel ist ein Gitterpunkt in einer mehrdimensionalen Struktur). Jeder Datensatz enthält die Informationen von insgesamt 30 Zeitschritten. Die Daten jedes Zeitschritts wiederum enthalten 60 Längenwerte (x-Richtung), 256 Breitenwerte (y-Richtung) und 256 Höhenwerte (z-Richtung). Damit kann mit entsprechender Software eine Darstellung der MRT-Aufnahme in Sagital- (zx), Frontal- (yz) bzw. Transversalebene (xy) erstellt werden (siehe Abbildung 2). Die verwendeten Darstellungen in dieser Dokumentation sind anatomische Frontalschnitte. Die räumliche Auflösung eines Datensatzes beträgt in etwa 0,5 cm in z-Richtung und 0,25 cm in x- und y-Richtung. Die pro Aufnahme rund 65 Millionen Einzelwerte wären als Eingabewerte zu viele, um das neuronale Netz zu trainieren. Deswegen werden die Aufnahmen noch vorverarbeitet, um die Datenmenge zu reduzieren. Der Ankunftszeitpunkt des Kontrastmittels für jede Aufnahme wird manuell vorher ermittelt, um als Lösung für den Trainingsdatensatz verwendet zu werden.

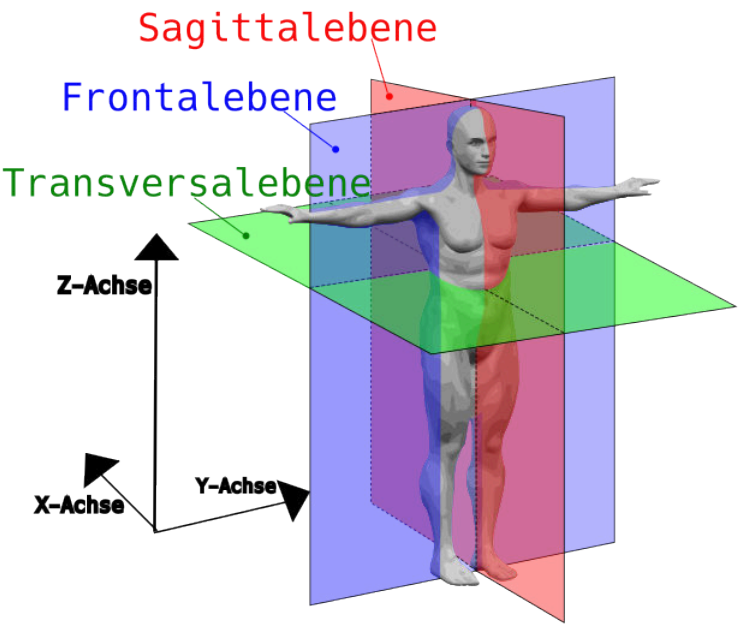


Abbildung 2: Medizinische Ebenen[5]

2.2 Datenaufbereitung

2.2.1 Features

Auf einem Teil der Aufnahmen ist überhaupt kein Kontrastmittel erkennbar. Daher werden diese aussortiert. Für jede der dadurch übrig bleibenden 515 Aufnahmen werden nun 2 sogenannte *Features* (Datensätze, mit denen das neuronale Netz trainiert wird) kreiert. Das erste Feature enthält pro Zeitschritt einen repräsentativen Helligkeitswert (*Image*) und das Zweite stellt die Veränderung desselben dar (*Changes*).

2.2.2 Vorverarbeitungsschritte

Die Aufnahmen liegen als Bilder im .tif Format vor, werden aber mit `skimage` (einer *Python-Bibliothek*)zu mehrdimensionalen Arrays umgewandelt. Es sind bereits ein paar Methoden zur Vorverarbeitung implementiert, aber viele müssen angepasst werden und manche selbst entwickelt werden. Für die komplette Entwicklung wurde Visual Studio Code und Python auf einem Computer des IAM-MMS verwendet. Alle Funktionen die entwickelt oder modifiziert wurden, sind im Anhang hinterlegt. Die Aufbereitung der Daten erfolgt dabei in 4 Schritten (siehe Abbildung 3).
Im ersten Schritt werden bei jeder Aufnahme die Werte für die vorderen und hinteren Frontalebenen verworfen. Dafür wird die Methode `squish` verwendet. Dadurch fallen Frontalebenen heraus, auf denen man nicht das Innere des Brustkorbes sehen kann. Die Bilder haben somit nur noch 8 Werte in z-Richtung.
Im zweiten Schritt werden die Datenwerte in x-Richtung jeweils durch ihr Maximum ersetzt. Somit besitzt der Datensatz einer Aufnahme nur noch die Dimension $30 \cdot 8 \cdot 1 \cdot 256$, was anschaulich einer Reduktion von vier auf drei Dimensionen entspricht. Diese Reduktion wird mit der Methode `reduce` durchgeführt, wobei man den Modus `max` benutzt. Der Modus von `reduce` gibt an, welcher Operator zur Datenreduktion verwendet wird; der Modus `max` steht für den Maximalwert. Andere Modi sind `mean` für den Mittelwert oder `min` für das Minimum. Da es um besonders helle Punkte geht (helle Voxel haben höhere Zahlenwerte), führt der Modus `min` nicht zu guten Ergebnissen. Deshalb wird einmal `mean` und zweimal `max` verwendet.
Als Drittes wird für jeden Wert in y-Richtung der Maximalwert in dieser Richtung gebildet. Der daraus resultierende Datensatz hat somit effektiv nur noch zwei Dimensionen.
Für den abschließenden Schritt ermittelt man für jeden Zeitschritt den Mittelwert der so verbleibenden 8 Werte in z-Richtung. Hierbei wird ebenfalls die Methode `reduce` benutzt, allerdings im Modus `mean`. Somit hat sich der Datensatz auf die Dimension $30 \cdot 1 \cdot 1 \cdot 1$ reduziert. Es liegen also pro Patientenaufnahme 30 zeitabhängige Werte vor. Diese 30 Werte werden als *Feature Image* abgespeichert (siehe Abbildung 3).
Für das zweite *Feature* werden jeweils die räumlichen Werte zweier benachbarter Zeitschritte innerhalb einer Aufnahme benutzt. Pro Voxel eines Zeitschrittes wird die Helligkeitsdifferenz zum gleichen Voxel, der einen Zeitschritt weiter ist, gebildet. Somit hat man nur noch 29 Zeitschritte, wobei jeder einzelne Zeitschritt als Änderungsaufnahme der Orginalaufnahmen zu verstehen ist. Mit diesen Daten werden

nun die gleichen 4 Vorverarbeitungsschritte wie für das erste *Feature* durchgeführt. Diese nun 29 Werte werden als *Feature Changes* gespeichert.

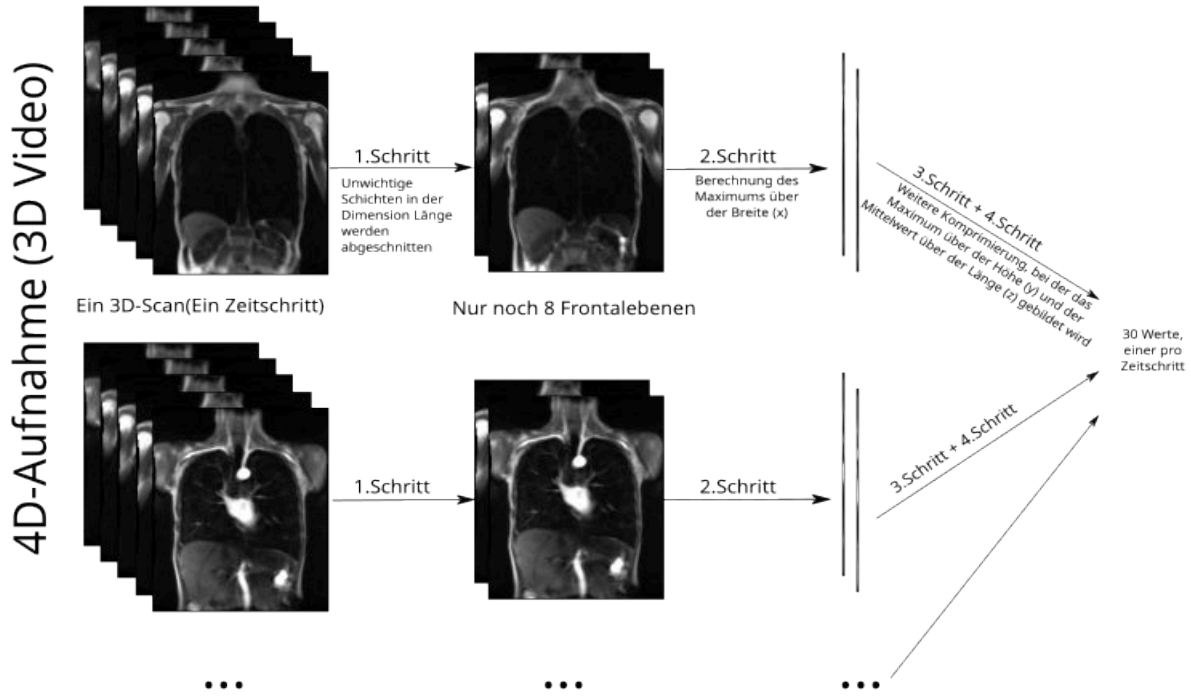


Abbildung 3: Aufbereitungsschritte für das *Feature Image*.

2.3 Neuronale Netze

2.3.1 Grundlagen

Ein neuronales Netz besteht aus Knoten, die über Kanten miteinander verbunden sind. Diese Struktur des neuronalen Netzes wird oft mit dem Aufbau des Gehirn verglichen, wobei man die Kanten mit den Axonen und Dendriten und die Knoten mit Zellkernen vergleichen kann. In der Abbildung 4 repräsentieren die Punkte einzelne Knoten und die Striche die Kanten.
Die Ausgaben einzelner Knoten dienen als Eingabe der Knoten der darauffolgenden Schicht des neuronalen Netzes. Dabei wird ihr Einfluss durch die Gewichtung der Kanten bestimmt. Der Wert eines Knotens (Ausnahme sind Knoten in der Eingabeschicht) wird durch die Summe aller Knotenwerte der Schicht davor, multipliziert mit der jeweiligen Gewichtung der Kanten, errechnet.
Auf den so erhaltenen Wert wird eine Aktivierungsfunktion (siehe Abbildung 5) angewendet, um die Werte zwischen 0 und 1 zu normieren und um eine Triggerfunktion zu implementieren. Die verwendete Aktivierungsfunktion heißt Sigmoidfunktion und kann als $f(x) = \frac{1}{1+e^{-x}}$ geschrieben werden. Für das Training des neuronalen Netzes werden zunächst die Gewichtungen der Kanten zufällig festgelegt. Beim Training werden die Gewichtungen nach und nach angepasst.
Die Ausgabe des neuronalen Netzes in der Ausgabeschicht wird über die Metrik mit der erwarteten

Antwort verglichen. Diese Trainingsart wird als überwachtes Lernen(engl.*supervised learning*) bezeichnet. Das heißt, dass das Netz nicht selbstständig, sondern mit vorgegebenen Lösungen trainiert wird. Bei diesem Training werden die Gewichtungen auf den Kanten langsam durch Backpropagation (engl.*backpropagation*) angepasst . Je nach Größe der Gesamtabweichung und Anteil einer einzelnen Gewichtung am Gesamtfehler, wird die Gewichtung einer Kante entsprechend so angepasst, das die Abweichung etwas geringer wird. Der Korrekturwert wird mit der Lernrate multipliziert. Mit einer sehr kleinen Lernrate beeinflusst ein Training die Gewichtungen also nur geringfügig. Die sogenannte Dropoutrate gibt an, welcher Prozentsatz der Knoten beim Training ausgelassen werden. Dies wird gemacht, um eine Überanpassung(engl.*Overfitting*) zu vermeiden, d. h. um zu vermeiden, dass das Netz zwar Trainingsaufnahmen sehr gut auswertet, aber es bei den ungesehenen Testdaten zu schlechten Vorhersagen kommt.

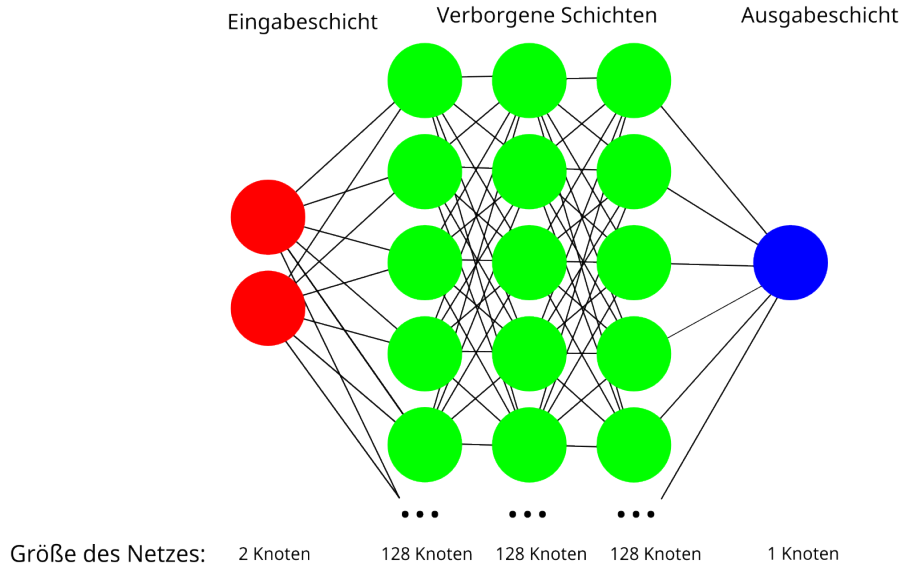


Abbildung 4: Aufbau eines neuronales Netzes mit zwei Eingabeknoten, drei verborgenen Schichten mit jeweils 128 Knoten und einem Ausgabeknoten

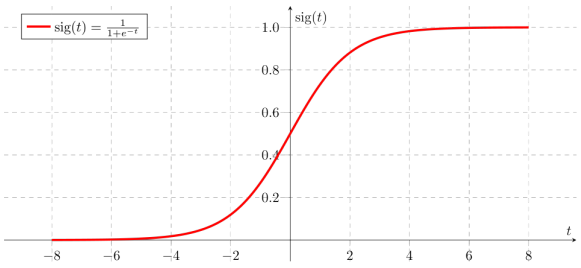


Abbildung 5: Diagramm der Sigmoidfunktion[10]

2.3.2 Matrizenmultiplikation

Matrizen sind besonders gut geeignet um die Gewichtungen der Kanten des neuronalen Netzes mathematisch darzustellen. Zudem sind sie algorithmisch besonders effizient anzuwenden. Eine Matrix repräsentiert alle Kanten zwischen 2 aufeinanderfolgenden Schichten. Für die Verarbeitung einer Eingabe und für das Training wird deswegen Matrizenmultiplikation benötigt.[7] In Abbildung 6 ist ein einfaches neuronales Netz zu sehen.

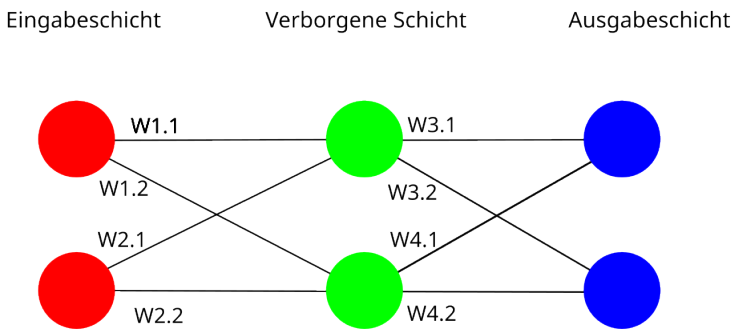


Abbildung 6: Aufbau eines neuronales Netz mit Gewichtungen

Die allgemeine Formel für die Berechnung der Ausgabe aus einer Eingabe für dieses Netz wäre:

$$Output = M_1 \times (M_2 \times Input)$$

Die Gewichtung dieses neuronalen Netzes wären die Gewichtungsmatrix(M_1) zwischen Eingabeschicht und verborgener Schicht :

$$M_1 = \begin{pmatrix} W1.1 & W2.1 \\ W1.2 & W2.2 \end{pmatrix}$$

Die Matrix(M_2) zwischen verborgener Schicht und Ausgabeschicht ergibt sich damit zu:

$$M_2 = \begin{pmatrix} W3.1 & W4.1 \\ W3.2 & W4.2 \end{pmatrix}$$

Wenn das neuronale Netz nun einen Vektor I als Eingabe bekommt, wird die Ausgabe folgendermaßen berechnet:

$$\begin{aligned} Ausgabe &= \begin{pmatrix} W_{3.1} & W_{4.1} \\ W_{3.2} & W_{4.2} \end{pmatrix} \times \left[\begin{pmatrix} W_{1.1} & W_{2.1} \\ W_{1.2} & W_{2.2} \end{pmatrix} \times \begin{pmatrix} I_1 \\ I_2 \end{pmatrix} \right] \\ &= \begin{pmatrix} W_{3.1} & W_{4.1} \\ W_{3.2} & W_{4.2} \end{pmatrix} \times \begin{pmatrix} W_{1.1} \cdot I_1 + W_{2.1} \cdot I_2 \\ W_{1.2} \cdot I_1 + W_{2.2} \cdot I_2 \end{pmatrix} \\ &= \begin{pmatrix} W_{3.1} \cdot (W_{1.1} \cdot I_1 + W_{2.1} \cdot I_2) + W_{4.1} \cdot (W_{1.2} \cdot I_1 + W_{2.2} \cdot I_2) \\ W_{3.2} \cdot (W_{1.1} \cdot I_1 + W_{2.1} \cdot I_2) + W_{4.2} \cdot (W_{1.2} \cdot I_1 + W_{2.2} \cdot I_2) \end{pmatrix} \end{aligned} \tag{1}$$

2.3.3 LSTM

Da die Berechnung der Ausgabewerte von neuronalen Netzen auf Matrixmultiplikation basiert, muss die Eingabe immer die gleiche Vektordimension besitzen. Jedoch sind auf manchen Aufnahmen mehr Zeitschritte vorhanden als auf anderen. Um dieses Problem zu lösen, gibt es RNNs (*Recurrent Neuronal Networks*). Diese funktionieren so, dass unterschiedlich viele sequentielle Werte nacheinander in das Netz eingeben werden, und diese miteinander addiert werden. Abbildung 7 zeigt wie man vom Grundaufbau eines RNNs zur Arbeitsform gelangt.

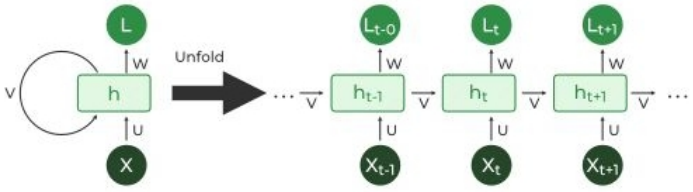


Abbildung 7: Aufbau eines RNNs[8]

Jeder Eingabeknoten gibt seinen Ausgabewert an den nächsten Eingabeknoten weiter, wo er mit der Eingabe verrechnet wird. Das passiert für alle Eingabewerte. Dabei werden immer die gleichen Gewichtungen verwendet (d. h. auch bei 100 Eingaben gibt es meist nur 3 Gewichtungen). Der wesentliche Unterschied zu normalen neuronalen Netzen ist, dass die Ausgabe des n-ten Knoten nicht nur von der Eingabe aller Knoten vor ihm abhängt, sondern auch von Eingaben nach ihm. Da RNNs

aber nur sehr schwer zu trainieren sind (aufgrund des *exploding/vanishing gradient problem*) verwendet man häufig an ihrer Stelle LSTMs (*long-/short-term memory*). Sie funktionieren ähnlich wie RNNs, aber haben zwei Wege Informationen weiter zu geben (einen für Langzeiterinnerungen und einen für Kurzzeiterinnerungen). Graphik 8 zeigt: im Gegensatz zu den RNNs gibt es zwei Werte die ein Knoten weitergibt. Der obere Weg ist das Langzeitgedächtnis, von dem jede Zelle einen variablen Anteil vergisst und etwas hinzufügt, während der untere Weg das Kurzzeitgedächtnis repräsentiert. Dieses ist die Ausgabe der Zelle und stark vom Langzeitgedächtnis abhängig, beeinflusst aber das Langzeitgedächtnis der nächsten Zelle. Die Verwendung von LSTMs bietet den Vorteil, dass es nicht mehr nötig ist, Bilder mit weniger Zeitschritten aufzufüllen.

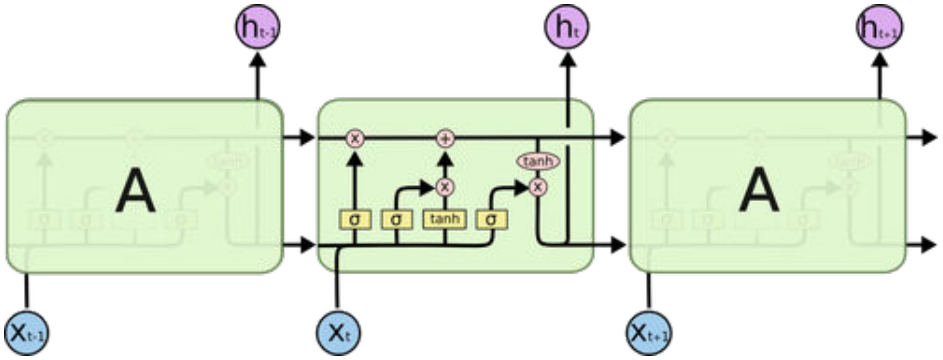


Abbildung 8: Aufbau einer LSTM Schicht mit 3 Zellen[9]

2.4 Training

Die Zuverlässigkeit eines neuronalen Netzes und das effektive Training hängen unter anderem von der Anzahl der Schichten, Anzahl der Knoten, der Lernrate sowie der Dropoutrate ab. Die Suche nach den vielversprechendsten Parametern (sogenannte Hyperparameter) wird mithilfe eines *Hyperband Search* Ansatzes durchgeführt. Dabei werden am Anfang die Grenzen und Schrittweiten der einzelnen Parameter festgelegt, es wird also festgelegt, welche Werte ein Parameter besitzen kann. Es werden nun alle möglichen Kombinationen an Parameterwerten in einem Turniersystem gegeneinander ausprobiert, wobei schon nach 20 Epochen getestet wird. Die besten Parameterkombinationen werden dann für das eigentliche Training weiterverwendet. Meistens nimmt man mehr als eine Parameterkonstellation, da bei längerem Training sich manche Parameterkombinatonen besser oder schlechter eignen.
Das reguläre Training mit den so gefundenen, gut geeigneten Hyperparametern wird mit 300 Epochen durchgeführt.
Um eine Aussage darüber zu treffen, wie gut ein neuronales Netz ist, wird das neuronale Netz nach dem Training mit Validierungsdaten von Patientenaufnahmen überprüft, die nicht für das Training verwendet werden. Von den insgesamt vorliegenden Patientenaufnahmen werden durch einen statistischen Prozess 70 % in Trainingsdaten, 15 % in Validierungsdaten und 15 % in Testdaten unterteilt.

2.5 Metriken

Die Metrik ist ein Verfahren zur Berechnung der Abweichung des Ergebnisses des neuronalen Netzes (engl.: *criterion*) zu den vorgegebenen Lösungen.
Um die Bewertung vornehmen zu können werden die Aufnahmen zuerst manuell ausgewertet, indem der Ankunftszeitpunkt des Kontrastmittels im Herzen bestimmt wird.
Nun wird für jede Patientenaufnahme eine zeitabhängige Funktion definiert. Vor dem Ankunftszeitpunkt hat die Funktion einen Ausgabewert von 0,5. Ab dem Ankunftszeitpunkt ist der Funktionswert 1. Da ein Großteil der Aufnahmen nicht die maximale Anzahl an Zeitschritten hat, fällt der Funktionswert nach einigen Zeitschritten auf 0 ab. Das neuronale Netz soll nun durch das Training mit dem vorverarbeiteten Datensatz einen ähnlichen Graphen ausgeben.
Der MAE (engl.: *Mean Absolute Error*) gibt an, wie weit der berechnete Wert vom gewünschten Ergebnis entfernt ist. Das *Mean* bedeutet, dass der Durchschnittsfehler von allen 30 Ausgabeknoten berechnet wird. *Absolute* bedeutet, dass dieser Fehler immer positiv ist, egal ob das Netz über oder unter der vorgegebenen Lösung liegt. Der MAE gibt also an, wie gut im Durchschnitt, zwei Kurven übereinander liegen.
Eine andere in dieser Arbeit verwendete Metrik (*Peakdifferenz*) berechnet die Differenz zwischen dem Zeitschritt, welcher manuell als Ankunftszeitpunkt ermittelt wird, und dem Zeitschritt, den das neuronale Netz als Ankunftszeitpunkt ausgibt(siehe Abbildung 9). Im Graphen ist diese Differenz zu erkennen, indem man die Zeitschrittdifferenz zwischen dem ersten Zeitschritt, bei dem die manuell definierte Funktion den Ausgabewert 1 hat, und dem Zeitschritt, bei dem das neuronale Netz den Ausgabewert 1 ausgibt, bildet. Diese Metrik wird verwendet, da für die Aufgabenstellung der Ankunftszeitpunkt des Kontrastmittels interessant ist. Für den MAE würde jeder einzelne Zeitschritt einen Einfluss haben, z.B. würde auch bewertet werden, ob das neuronale Netz bei den späteren Zeitschritten immer auf Null bleibt, was für die Aufgabenstellung aber uninteressant ist.

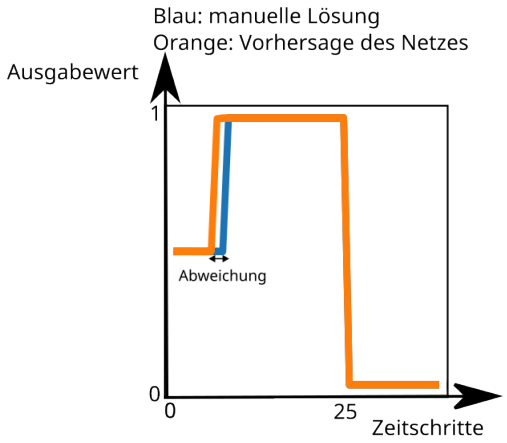


Abbildung 9: Beispielausgabe des neuronalen Netzes

3 Ergebnisse

3.1 Hyperparametersuche

Für die Hyperparametersuche wird der MAE als Metrik verwendet und nur 20 Epochen trainiert. Die 5 besten Kombinationen sind in der folgenden Tabelle aufgelistet.

#	Kontenzahl	verborgenen Schichten	Lernrate	Dropoutrate	MAE nach 20 Epochen
1	64	3	0,003	0,1	0,0231
2	128	3	0,003	0,1	0,0253
3	32	3	0,003	0,1	0,0450
4	256	4	0,003	0,65	0,0454
5	256	5	0,001	0,45	0,0463

Nur die ersten beiden Netze werden weiter trainiert, da schon die dritte Hyperparameterkombination einen deutlich höheren MAE aufweist. Als vielversprechendste Hyperparameterkombinationen ergeben sich damit ein Netz mit 3 verborgenen Schichten und je 64 oder 128 Knoten. Die Lernrate ist bei 0,003 am vielversprechendsten und bei der Dropoutrate ein Wert von 0,1.

3.2 Ergebnisse der verschiedenen Metriken

3.2.1 MAE

Diese Metrik lag bei unseren besten Ergebnissen bei ca. 0,01. Da die Durchschnittsabweichung des gesamten Verlaufs der zeitabhängigen Funktion für die Fragestellung nicht interessant ist, sondern der Abstand des ersten Peaks, also die Zeitschrittdifferenz der Ankunftszeitpunkte des Kontrastmittels, haben wir die Netze aber nicht damit bewertet; der MAE wird nur für die Hyperparametersuche beachtet.

3.2.2 Peakdifferenz

Es wurden 2 neuronale Netze mit verschiedenen Hyperparametern trainiert und getestet. Diese Modelle haben beide eine Lernrate von 0,003 , drei Schichten und eine Dropoutrate von 0,1. Sie unterscheiden sich lediglich in der Anzahl der Knoten pro Schicht, wie es sich aus den beiden besten Ergebnissen der Hyperparametersuche ergeben hat.
Die Modelle werden über 300 Epochen mit je 360 Trainingsaufnahmen trainiert. Bei der Überprüfung mit den Validierungsdaten, zeigt das Modell mit 128 Knoten nach 300 Epochen eine Abweichung von 0,507. Dies bedeutet, dass das Netz im Durchschnitt 0,507 Zeitschritte neben der manuellen Lösung liegt, es also sehr oft den genauen Zeitpunkt trifft oder einen Zeitpunkt daneben liegt. Das andere, kleinere Modell, hatte auf diesem Datensatz eine Abweichung von 0,532. Danach werden beide Netze mit den Testdaten getestet, also dem Datensatz, den das neuronale Netz noch nie gesehen hat. Hier liegt die Abweichung für das Netz mit 64 Knoten bei 0,881 und die für das Netz mit 128 Knoten bei 0,753.

Bei den Testdaten sieht die Verteilung der Abweichung aber anders aus, da auf einer Aufnahme der Ankunftszeitpunkt nicht klar erkennbar ist. Hier liegen beide Netze weit, meistens 12 oder 13 Zeitschritte neben dem vorgegebenen Ankunftszeitpunkt.
Bei der genaueren Analyse dieser Aufnahmen zeigt sich, dass das neuronale Netz eine wahrscheinlich bessere Lösung anzeigt, als das manuell ermittelte Verfahren. Dies kann man deutlich am Vergleich der beiden Aufnahmen erkennen (siehe Abbildung 10 und 11). Das vorliegende neuronale Netz ist dementsprechend nur so gut wie die Menschen, die den Datensatz klassifizieren. Lässt man diese Aufnahme aus dem Testdurchlauf heraus, liegt das Netz mit 64 Knoten bei 0,710 und das Netz mit 128 Knoten bei 0,579 durchschnittlichen Zeitschritten daneben.

Tabelle 1: Ergebnisse der Tests

Netze	Testdaten	Testdaten(ohne Ausreißer)
Netz mit 64 Knoten	0,881	0,710
Netz mit 128 Knoten	0,753	0,579

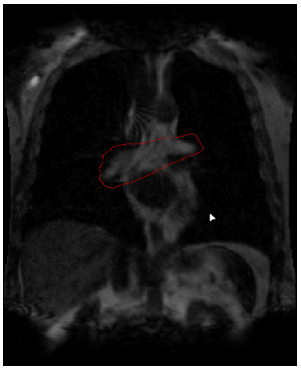


Abbildung 10: Manuell ermittelter Zeitschritt (6). In der Pulmonalartierie ist es hell.

Abbildung 11: Zeitschritt des neuronalen Netzes (19). Es ist deutlich heller in der Pulmonalartierie

3.3 Fehlerdiskussion

Wie bereits bei den Ergebnissen gezeigt, sind die manuell ausgewerteten Lösungen nicht immer eindeutig richtig. Von den Testdaten war 1 von 76 Aufnahmen fehlerhaft ausgewertet. Bei den Trainings- und Validierungsdaten ist die Anzahl an fehlerhaft ausgewerteten Aufnahmen nicht klar, es sind aber mit hoher Wahrscheinlichkeit ebenfalls welche enthalten.
Weitere Ungenauigkeiten können durch die Einteilung des Fließprozesses des Kontrastmittel in feste Zeitschritte auftreten. Bei genauerer Untersuchung ist es bei vielen Aufnahmen oft nicht klar, welcher Zeitschritt bei zwei benachbarten Zeitschritten der bessere ist. Das heißt aber auch, dass, wenn das neuronale Netz hier eine Abweichung von einem Zeitschritt hat, das dies nicht eindeutig als ungenau einzustufen ist.
Eine weitere Fehlerquelle liegt in der Vorverarbeitung. Durch die starke Vorkomprimierung könnte auf manchen Aufnahmen das neuronale Netz Schwierigkeiten haben, den genauen Zeitschritt zu finden.
Das Endergebnis könnte auch durch die Hyperparamertersuche nicht optimal sein. Eine Hyperparametersuche mit längerer Trainingsdauer als 20 Epochen könnte andere, für längeres Training besser geeignete, Parameterkombinationen hervorbringen. Außerdem wird bei der Hyperparametersuche der MAE als Metrik verwendet. Wenn dafür stattdessen die Peakdifferenz als Bewertungsmetrik verwendet würde, könnten andere Parameterkombinationen als die Beste eingestuft werden.

4 Fazit und Ausblick

In der vorliegenden Arbeit konnte gezeigt werden, dass es möglich ist mit einem neuronalen Netz MRT-Aufnahmen auszuwerten. Dabei werden die Daten sehr stark vorverarbeitet und vereinfachte Metriken verwendet. Dennoch zeigte sich, dass bereits unter diesen vereinfachten Bedingungen ein neuronales Netz den Ankunftszeitpunkt des Kontrastmittels ziemlich genau erkennen kann. Man könnte die Fehlerquote noch weiter verbessern, indem man weniger oder komplizierte Vorverarbeitungsschritte benutzt oder das neuronale Netz größer (Vergrößerung der Knotenzahl und Anzahl verborgener Schichten) macht, dies würde aber viel mehr Rechenleistung erfordern. Das vorliegende Verfahren vereinfacht den Prozess der MRT-Auswertung, man kann sich aber eine zweite künstliche Intelligenz vorstellen, welche mit den Informationen der ersten eine Diagnose durchführt.
Auf diesem Netz aufbauend könnte man weitere KI-Tools programmieren, die weitere Analyseschritte automatisch übernehmen könnten, wie z.B. die Verteilung des Kontrastmittels in der Lunge zu analysieren.

5 Danksagung

Wir danken zuallererst unseren Betreuern am IAM-MMS Dr. Arnd Koeppe und Julian Grolig, welche uns stets gute Unterstützung gegeben haben. Wir danken desweiteren Prof. Dr. Britta Nestler, der Institutsleiterin am IAM-MMS, für die Erlaubnis am Institut zu arbeiten. Dank geht auch an Herrn Krieg vom Hector-Seminar für die Betreuung in den vergangenen 6 Jahren.

Einen besonderen Dank möchten wir an Dr. Hans-Werner Hector und Josephine Hector richten, die durch ihre Stiftung es überhaupt erst möglich gemacht haben, eine solche mathematisch-naturwissenschaftliche Förderung zu erhalten.

6 Anhang

6.1 Vorverarbeitung

```
# Define Data Preprocessing
preprocessor = Preprocessor(DataDefinition(input_features=[], output_features=[]))
preprocessor.add_preprocessing_step(
["squish_axis",["image","X",-3,4],{}]
)
preprocessor.add_preprocessing step(
["reduce",[["image"]],{"axes":["Z","Y"],"mode":"max", "rename":True}]
)
preprocessor.add_preprocessing_step(
["reduce",[["image"]],{"axes":["X"],"mode":"mean", "rename":True}]
)
preprocessor.add_preprocessing_step(
["expand",["label"],{"new_axis":"S","new_size":"image",
"mode":"onehot_switch","rename":True}]
)
```

6.2 Methode *squish*

```
def squish_axis(
    self,
    sample: dict,
    feature: str,
    axis: OptionsAxis = OptionsAxis.X,
    start_from_mid: int = -3,
    end_from_mid: int = 4
    ) -> tuple:
    """Reduces the amount of data by striping away the edges of the given Axis

    Args:
        sample (dict): the given samples
        feature (String): The Name of the targeted Feature
        axis (Axis): The Axis to reduce
        start (Int): Position relative to the middle of the Array
        (lower Indexes will be deleted)
        end (Int): Position relative to the middle of the Array
        (higher Indexes will be deleted)

    Returns:
        sample (tuple): the new, squished samples
        datadefinition (Datadefinition): the new datadefinition
```

```
    data_definition = copy.deepcopy(self.data_definition)
    axis = data_definition. features [feature] .data_format.index(axis)
    sample_feature = sample[feature]"""

    mid = int(sample _feature.shape[axis] / 2)
    start = mid + start_from_mid
    end = mid + end_from_mid + 1 # +1 so the last layer will be included
    assert start >= 0
    assert end <= mid * 2
    if axis == 1: # The Time axis normally
        new_sample = sample feature[:, start:end, :, :, :, :]
    if axis == 2: # The X axis normally
        new_sample = sample feature[:, :, start:end, :, :, :]
    if axis == 3: # The Y axis normally
        new_sample = sample feature[:, :, :, start:end, :, :]
    if axis == 4: # The Z axis normally
        new_sample = sample feature[:, :, :, :, start:end, :]
    # Update Variables
    sample[feature] = new_sample
    data_definition.feature_shape[feature][axis] = (
        start_from_mid * -1 + end_from_mid + 1
    ) # ALL samples have now the same size
return sample, data definition
```

6.3 Methode *reduce*

```
@register(PreprocessingContainer.REGISTERED FUNCTIONS)
@nested_preprocessing_function
def reduce(
    self,
    sample: dict,
    features: Iterable[str] = None,
    axes: Iterable[OOptionsAxis] = [OptionsAxis.F],
    mode: OptionsReduceMethods = OptionsReduceMethods.mean,
    rename: bool = False,
    ) -> tuple;
    """Reduces the dimensionality of the given axis using given reduction mode.
    # FIXME: Multifeature functionality doesnt work :(
    Args:
    sample (dict): dictionary of feature tensors
    features (Iterable[str], optional): list of feature names.
          Defaults to None.
    axis (str, optional): axis of feature tensor to split. Defaults to "F".
    mode (str, optional): how to reduce the given axis. Defaults to "mean".
```

```
        One of "mean", "std", "min", "max", or an index integer
    rename (bool,optional): if the feature should overwrite the given feature

    Returns:
    tuple: sample (dict): modified dictionary of feature tensors,
         data_definition(DataDefinition): modified data definition
    """
    data_definition = copy.deepcopy(self.data_definition)
    for axis in axes:
        assert axis != "N"
        # data _ definition = copy.deepcopy(self.data_definition)
        features = features or list(data_definition.features.keys())
        mode_str = mode
        for feature_name in features:
            feature = data_definition.features[feature_name]
            if axis == axes[0]:
                 ndarray = copy.deepcopy(sample[feature_name])
            else:
                 ndarray = copy.deepcopy(value)
            try:
                assert instance(ndarray, np.ndarray)
                index = feature.data_format.index(axis)
            except (AssertionError, ValueError):
                # Feature doesn"t have this axis
                continue
             # Update value
        if mode == OptionsReduceMethods.mean:
            value = np.mean(ndarray, axis=index)
        elif mode == OptionsReduceMethods.std:
            value = np.std(ndarray, axis=index)
        elif mode == OptionsReduceMethods.min:
            value = np.min(ndarray, axis=index)
        elif mode == OptionsReduceMethods.max:
            value = np.max(ndarray, axis=index)
        elif mode == OptionsReduceMethods.ravel:
            new_shape = [d for i, d in enumerate(ndarray.shape) if i != index]
            new_shape[0] = -1
            value = np.reshape(ndarray, newshape=new_shape)
        else:
            try:
                # Assume mode is an index (integer)
                element = int(mode)
                idx = [slice(None)] * len(np.shape(ndarray))
                idx[index] = element
```

```
                value = ndarray[tuple(idx)]
            except ValueError as e:
                help = "Must be one of 'mean', 'std', 'min', 'max', or an int."
                raise ValueError(f"Invalid mode: {mode}. {help}") from e
            mode_str = "index"
        # Update data definition
        base_new_feature_name = feature_name + f" {axes} {mode _str}"
        feature_index = len(
            [
                f
                for f in data_definition.features.keys()
                if base new _ feature name in f
            ]
        )
    new_Teature_name = base_new_feature_name + str(feature_index)
    if rename:
            new_feature_name = feature_name
    new_feature = self. update feature _after split(
            feature _name, axis=axis,data_definition=data_definition
    )

    new _feature.name = new_feature_name
    if new_feature.decode_str_to:
        new_feature.decode str_to = str(value.dtype)
    elif new_feature.dtype:
        new_feature.dtype = str(value.dtype)
    data_definition.add_feature(new_feature)
    # Save value
    sample[new_feature_name] = value
    # del data _definition.feature_shape[feature name]
    [data _definition.data_format.index(axis)]
    # data_definition.data_format = data_definition.data_format.replace(axis,"")
return sample, data definition
```

6.4 Model des Netzes

```
 # Neural network

# Model
def model_function(hp, data definition):
    # Hyper parameters
    num_units = hp.Choice("num_units", [32, 64, 128, 256, 512], default=128)
    num_layers = hp.Int("num_layers", 1, 5, default=3)
```

```
    dropout_rate = hp.Float("dropout", 0.0, 0.7, default=0.3)
    num_outputs = data_definition.output_shape[-1]
    layers = []
    for _in range(num_layers):
        layers.append(
                klayers.LSTM(num_units,recurrent_dropout=dropout_rate, return_sequences=True)
        )
    layers.append(klayers.TimeDistributed(
            klayers.Dense(num_outputs, activation="sigmoid"))
    )

return tf.keras.Sequential(layers)
```

7 Literatur

Literatur

[1] Helmholz Munich: https://www.lungeninformationsdienst.de/krankheiten/copd/grundlagen , abgerufen : 29.7.2023

[2] COPD-aktuell:https://patient.boehringer-ingelheim.com/de/copd-aktuell/wie-haeufig-ist-die-copd-eigentlich , abgerufen : 29.7.2023

[3] http://www.asconet.net/html/cosyconet/teilprojekt1?asconetsid=90aed6979eebd7bf7f418db6d00a3cac , abgerufen: 15.8.2023

[4] https://garmisch-partenkirchen.die-radiologie.de/kernspintomographie-mrt , abgerufen : 29.7.2023

[5] https://upload.wikimedia.org/wikipedia/commons/d/d6/Human_anatomy_Koerperebenen.svg , abgerufen : 29.7.2023

[6] Universitätsklinikum Giessen und Marburg:https://www.ukgm.de/ , abgerufen : 29.7.2023

[7] Wolfram Mathworld : https://mathworld.wolfram.com/MatrixMultiplication.html , abgerufen : 29.7.2023

[8] https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network , abgerufen : 7.8.2023

[9] https://colah.github.io/posts/2015-08-Understanding-LSTMs/ , abgerufen : 7.8.2023

[10] https://de.wikipedia.org/wiki/Sigmoidfunktion , abgerufen : 14.8.2023

Selbstständigkeitserklärung

Hiermit versichern wir, dass diese Arbeit unter Beratung durch Dr. Arnd Koeppe, Herr Julian Grolig und Norbert Krieg selbständig verfasst wurde und keine anderen als die angegeben Quellen und Hilfsmittel verwendet wurden.

Fabian Habermann	Paul Schepperle