DeepRain

Thomas Gnädig, Etienne Gramlich, Tim Hardenacke, Merle Wolf

9. September 2019

Inhaltsverzeichnis

1 Abstract

Das Teamprojekt DeepRain bestehend aus fünf Studierenden der Masterstudiengänge MSI und BIT startete im Wintersemester 18/19 an der HTWG Konstanz. Das Ziel Projekts ist, innerhalb eines Jahres einen lauffähigen Algorithmus auf Github (thgnaedi/DeepRain) zur Verfügung zu stellen. Dieser soll mit Hilfe von Deep Learning in der Lage sein das Wetter, bzw. den Niederschlag, in Konstanz über den Zeitraum von einer Stunde vorherzusagen. Das einjährige Projekt wird von Prof. Oliver Duerr betreut. Eine möglichst genaue Wettervorhersage hat viele offensichtliche Vorteile. Beginnend mit dem täglichen Verlassen des Hauses und der Entscheidung; mit oder ohne den Regenschirm? Vermutlich treffen die meisten diese Entscheidung basierend auf der Wettervorhersage und vertrauen darauf, dass diese auch korrekt ist. Vorhersagen mit Deep Learning-Unterstützung finden auch eine sehr wichtige Verwendung in der Katastrophenvorhersage und dem Katastrophenmanagement (vgl. [HTRS19, S. 763]). Eine weitere wichtige Rolle spielt es bei der Energieversorgung, beispielsweise bei der Planung der Auslastung von Solar-Panels (vgl. [And, S. 2]). Die eigentliche Wettervorhersage wird jedoch bisher noch nicht mit Deep Learning-Techniken generiert, dies ist erst ab diesem Jahr bei einigen Anbietern angedacht (vgl. [Chr19]. Im

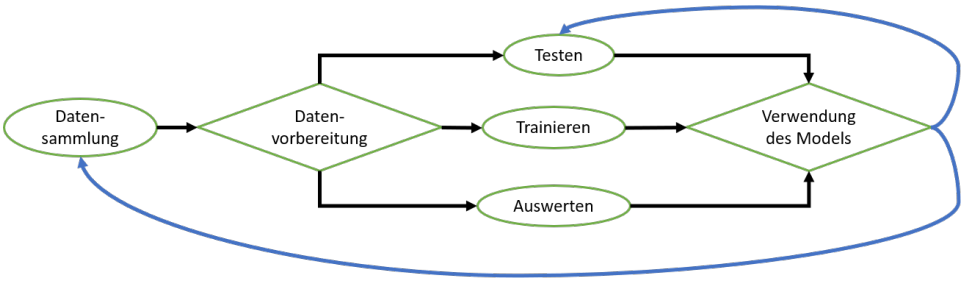


Abbildung 1: Wichtige Schritte im Prozess des DeepRain-Projekts, eigene Darstellung

Rahmen des Teamprojekts stand zu Beginn ein einarbeiten in die Prozesse des Maschine-Learnings an, sowie den Stand der Technik zu recherchieren. Daraufhin stand die Datenbeschaffung im Vordergrund und die Qualitätsbewertung der vorhandenen Daten. Im späteren Verlauf des Projekts ging es dann an die Entwicklung des Algorithmus. Der Ablauf dieser Arbeit lehnt sich an diesen, auch in Abbildung 1 dargestellten, Prozess an.

1.1 Deep Learning

Deep Learning basiert auf der Optimierung von künstlichen neuronalen Netzen (KNN) und ist eine Weiterentwicklung des Maschine Learning (vgl. [GT19, S. 1]). Ziel von neuronalen Netzen ist es, eine ähnliche Lösungsweise zu ermöglichen, wie im menschlichen Gehirn. Kern ist hier das "Lernen" und das Lösen von Problemen. Neuronale Netze haben sich vor allem wegen ihrer Fähigkeit der Verarbeitung von ständig wachsenden Datenmengen und -komplexität etabliert (vgl. [WEB18, S. 373]). In einem künstlichen neuronalen Netzwerk müssen im Vorhinein keine Vermutungen über Zusammenhänge festgelegt werden, denn diese werden während des Lernprozesses vom Netz ermittelt (vgl. [BEPW18, S. 581]). Ein Neuronales Netz ist so aufgebaut, dass es einen bekannten Input gibt, welcher durch einen sogenannten Hidden-Layer läuft. Der Hidden-Layer besteht aus Neuronen in denen das Lernen stattfindet und schließlich ein Output generiert wird. Dabei können einige Variablen, die den Output beeinflussen, festgelegt werden. Die Neuronen modifizieren dabei die Daten und leiten diese anschließend weiter (vgl. [WEB18, S. 373]). Es kann sich auch zwischen einer biologischen oder einer technischen Simulation von Neuronen entschieden werden (vgl. [htt14]). Die technische Simulation wird zur Datenverarbeitung und Mustererkennung meist bevorzugt (vgl. [htt14]). Dieser Zusammenhang wird in Abbildung 2 dargestellt. In dieser Abbildung wird auch ersichtlich, wodurch sich ein Deep Neural Network von einem neuralen Netz unterscheidet. Hier wird gleich mit mehreren Hidden-Layers, die hintereinander liegen, gearbeitet. Die Verwendung mehrerer Hidden-Layers führt dabei laut ([BEPW18, S. 581]) zu erfahrungsgemäß korrekteren Lernergebnissen. Das „Lernen“ hängt dann vom Grad der Aktivierung der einzelnen Neuronen ab. Dieser wird durch die bisherigen Ergebnisse durch eine ermittelte Gewichtung einzelner Neuronen bestimmt. Diese Gewichtung wird im Verlauf des Lernens im Netz im besten Fall immer weiter optimiert (vgl. [BEPW18, S. 586]). In unserem Projekt soll mithilfe von neuronalen Netzen, das Lernen aus bisherigen Wetterdaten ermöglicht werden, sodass eine möglichst korrekte Wettervorhersage der kommenden Stunde generiert wird.

1.2 State-of-the-art Wettervorhersage mit Deep Learning

Obwohl für Deep Learning, Machine-Learning, sowie Neuro- und Lingual-Programming einiges an Literatur zu finden ist, steht die Nutzung zur Wettervorhersage noch am Anfang (vgl. [Chr19]). So gibt es auf der Plattform GitHub zum aktuellen Stand (August 2019) gerade einmal eine Handvoll Repositories zum Thema Wettervorhersage mit Deep Learning. Verfahren werden verbessert, die Lernverfahren den Menschlichen Lernen immer ähnlicher, die Entwicklung ist stetig steigend und wie bereits erwähnt sind noch viele Bereiche wie u.a. die Wettervorhersage ergründbar ([Wic17, S. 103]).

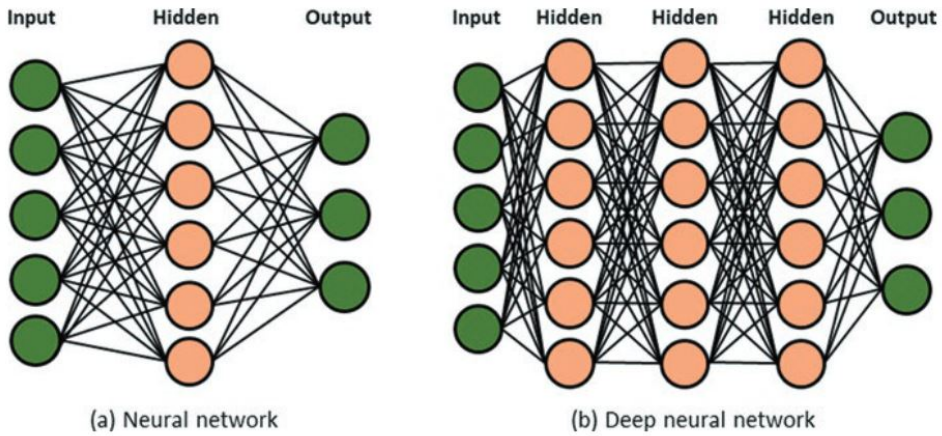


Abbildung 2: Unterscheidung Neurales Netzwerk und Deep-neurales Netzwerk, Quelle: ANN for deep learning (Akerkar 2019, S. 35)

Im Bereich der Wettervorhersage bietet sich, in Anlehnung an Abbildung 3, in den meisten Fällen das Supervised Learning an, da eine möglichst genaue Vorhersage erreicht werden soll. Aber auch das nicht-überwachte Lernen kann interessante Schlüsse zulassen, um eventuell unbekannte Muster zu finden (vgl. [WEB18, S. 371]).

2 Daten

In diesen Kapitel geht es um die Beschaffung und Vorbearbeitung der für das Training des Netzes erforderlichen Daten.

2.1 Quelle

Die verwendeten Radardaten stammen von den Radarstationen des Deutschen Wetterdienstes (im Folgenden DWD genannt) und geben die Stärke des Niederschlags an. Die Messstationen bilden Radien um die jeweiligen Stationen und bedecken dadurch Deutschland und einen kleinen Bereich der Nachbarländer. Die Radar-Messungen werden in stündlicher oder fünf minütiger Auflösung bereitgestellt, wir verwenden für unser Projekt die feinere fünf minütige Auflösung. Zurzeit liegen die Daten von Januar 2001 bis einschließlich Januar 2018 vor. Für unser Projekt verwenden wir 18 ganze Jahre, von 2001 bis 2017.[1]

[1] `https://opendata.dwd.de/climate_environment/CDC/grids_germany/5_minutes/radolan/reproc/2017_002/bin/`

2.1.1 Crawler

Da die Daten monatsweise in geschachtelten Archiven gepackt sind, haben wir einen Crawler geschrieben, der die Daten zuerst herunterlädt und auspackt. Mit den Kommandozeilenoptionen[2] kann gesteuert werden, ob die stündlichen oder minütlichen Daten heruntergeladen werden und wohin die Binärdateien entpackt werden sollen. Wir empfehlen, die Binärdaten auf eine Btrfs[3]-formatierte Partition zu entpacken, da die Daten (wegen häufig auftretender Nullen bzw. kein Regen) leicht komprimierbar sind und die minütlichen Daten sonst mehr als ein Terabyte belegen würden.

2.2 Preprocessing

Um das Binärformat des DWD einzulesen, benötigt man die Python-Bibliothek „Wradlib“, die man über den Package-Manager von Anaconda installieren kann. Dann kann man die Datensätze als Deutschlandkarte mit einer Auflösung von 1100x900 Pixeln rastern.

Das Preprocessing geschieht in zwei Durchläufen: Zuerst wird das Maximum des Niederschlags bestimmt, das Minimum wird als 0 (kein Regen) angenommen. Danach werden die Werte auf einen Bereich gespreizt.

Beim zweiten Durchgang werden die Daten mithilfe des globalen Maximalwertes auf einen Wertebereich zwischen 0 und 255 umgerechnet, damit die Datensätze in einem Bildformat mit einer Bittiefe von 8 Bit gespeichert zur weiteren Verarbeitung werden können. Wir haben uns für das PNG-Format entschieden, weil es verlustfrei komprimiert und von plattformübergreifenden Bibliotheken gelesen und geschrieben werden kann. Vor dem Speichern werden die Werte mit dem Faktor 4 multipliziert, um Werte über 255 auf den Maximalwert abzuschneiden. Dadurch verwerfen wir Ausreißer und das Training funktioniert besser. Der Faktor 4 wurde empirisch bestimmt, weil das Berechnen eines Histogramms über alle 18 Jahre zu aufwendig gewesen wäre.

2.2.1 Trainingsdaten

Da sich unsere Aufgabenstellung mit einer Regenvorhersage für Konstanz befasst, sind die aktuell gespeicherten Bilder noch deutlich zu groß. Daher wird aus dem gespeicherten Bild nur ein kleiner Bereich um Konstanz herum ausgeschnitten. die Position von Konstanz wurde bereits zuvor in Kapitel 3.2 bestimmt. Um diese Position wird nun ein Gebiet von 200x200 Pixeln extrahiert. Das so erzeugte Bild enthält dann alle Regenfronten, die das Wetter der nächsten 30 Minuten beeinflussen könnten. Um den Rechenaufwand zu reduzieren, werden die Bilder dann auf eine Größe von 64x64 Pixeln herab

[2] `https://github.com/thgnaedi/DeepRain/tree/master/DWD_Crawler`

[3] `https://en.wikipedia.org/wiki/Btrfs`

skaliert. Für eine Vorhersage haben wir uns dazu entschieden, fünf Zeitschritte zu verwenden. Es wird also das aktuelle Wetter, sowie die vergangenen 20 Minuten berücksichtigt. Als Label dienen dann n Zeitschritte, für den ersten Versuch sind $n = 1$, später soll auch weiter in die Zukunft vorhergesagt werden, hierzu werden $n = 7$ Zeitschritte verwendet.

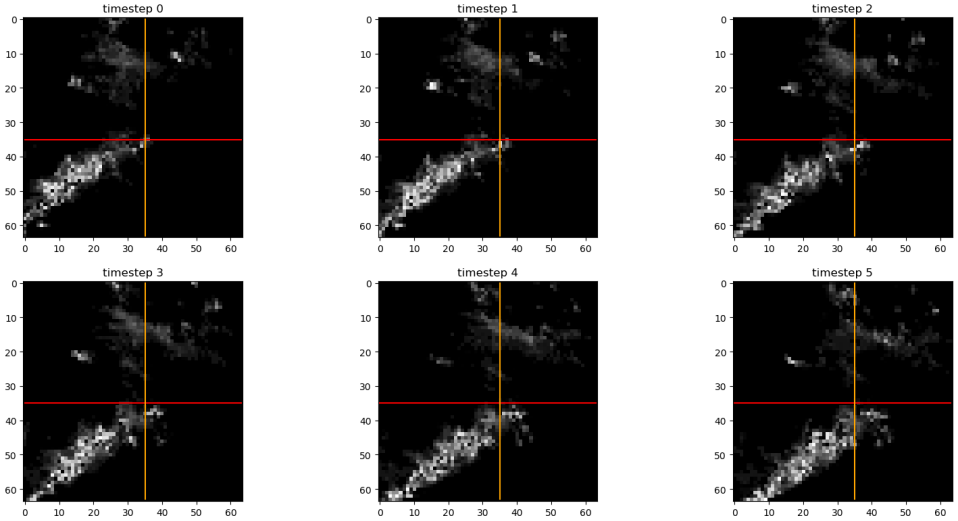


Abbildung 3: Die Grafiken timestep 0 bis timestep 4 sind die 5 eingehenden Daten, die als timestamp 5 bezeichnete Grafik entspricht dem zu lernenden Label. Die Bilder sind jeweils 5 Minuten voneinander entfernte Radarbilder. Die orange und rote Linien dienen nur zur besseren Darstellung der Bewegung.

Die Daten für eine einfache, fünf minütige Vorhersage sind in Abbildung 3 dargestellt.

Für das Training können allerdings nicht alle Daten verwendet werden. Es kann vorkommen, dass eine Radarstation keine Daten liefert; diese Bilder eignen sich nicht für das Training. Sehr häufig regnet es gar nicht erst, die Grafik ist also komplett schwarz und ohne Struktur; Diese Samples sind ebenfalls nicht für das Training geeignet. Als letzte Einschränkung gilt, dass über alle fünf eingehenden Zeitschritte ein Mindestmaß an Regen zu sehen sein muss um in das Trainingsset aufgenommen zu werden. Für die Label gibt es keine Einschränkung, da sowohl die Fortbewegung von Regen, als auch das verschwinden gelernt werden soll. Jeder Zeitschritt wird maximal einmal in das Trainingsset aufgenommen, was einmal als Label verwendet wurde, wird keinesfalls in einem anderen Sample als als Eingabedatum verwendet. Beim Aufteilen des Sets zwischen Trainings und Validierungsset ist es wichtig, dass nicht zufällige Samples ausgewählt werden, damit garantiert ist, dass keine ähnliche Wetterlage bereits gesehen wurde. Die so entstandenen Daten werden vor Eingabe in das Netz noch auf Werte zwischen 0 und 1 normiert. Ab jetzt können die Daten für das Trainieren verwendet werden.

2.3 Herausforderungen in diesem Kapitel

Die größte Herausforderung in diesem Kapitel war zweifelsfrei die große Datenmenge, die wir verarbeitet haben. Die Rohdaten der 18 Jahre in 5-Minuten-Auflösung hätte unseren zugewiesenen Speicher gesprengt. Mit Btrfs, dass die Dateien on-the-fly komprimiert und de-dupliziert, passten die Daten doch auf unsere Festplatte. Auch mussten wir eventuelle Ausreißer empirisch entfernen, weil das Berechnen des Histogramm über so viele Daten zu aufwändig wäre.

Darüber hinaus lagen die Archive des DWD im Tar-Format vor, das keine Checksumme anbietet und somit erst beim Entpacken bemerkt werden kann, dass beim Download einer Datei ein Fehler auftrat. Bei den großen Tar-Archiven des DWD ist leider mehrmals ein Download-Fehler aufgetreten, der sehr spät auffiel.

3 Daten-Aufbereitung

3.1 RADOLAN-Daten

Die Daten des DWD werden über das Routineverfahren RADOLAN (Radar-Online-Aneichung) erfasst, dass durch eine Kombination von Niederschlagstationen und Wetterradar hochauflösende Niederschlagsdaten produziert.

Die Open-Source Bibliothek wradlib[4] stellt für diese RADOLAN-Daten eine stereographische Projektion zur Verfügung. Dies bedeutet, dass die Erdkugel zu einem Koordinatensystem aufgespannt wird, dessen Ursprung im Nordpol liegt (siehe Abbildung 4).

[4] `https://docs.wradlib.org/en/stable/index.html`

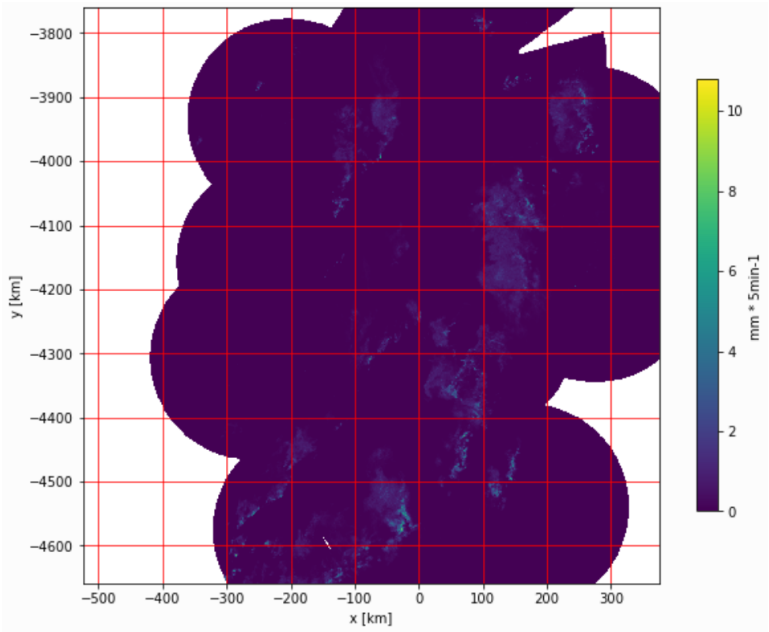


Abbildung 4: Ausschnitt Deutschlands im Koordinatensystem

Der resultierende Ausschnitt für Deutschland kann mit Hilfe von wradlib als 1100x900 Array ausgelesen werden, dessen Werte einer 1 Kilometer Grid-Box entsprechen.

3.2 Location Konstanz

Um für Konstanz und Umgebung sinnvolle Vorhersagen treffen zu können, muss der Standort von Konstanz auf den Daten markiert werden. Dazu müssen die geographischen Koordinaten von Konstanz in X- und Y-Koordinaten des Koordinatensystems umgewandelt werden.

Geographische Koordinaten

Latitude	**Longditude**
47.66033	9.17582

XY - Koordinaten

X-Koordinate	**Y-Koordinate**
-4602.6447	-66.4622

Die umgewandelten XY-Koordinaten müssen nun innerhalb des Arrays gefunden werden. Dazu wurde über den Array iteriert und die Werte mit den errechneten Werten verglichen. Anschließend muss die Location auf der

Karte markiert und dargestellt werden[5] (siehe Abbildung 5).

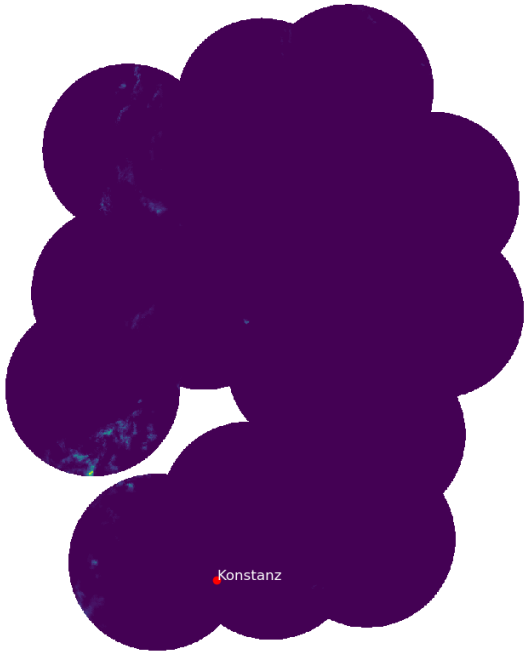


Abbildung 5: Location von Konstanz im Ausschnitt Deutschlands

3.3 Herausforderungen

Änderungen der Auflösung des Formats anpassen, richtige Datenquelle in wradlib finden. XY-Koordinaten innerhalb des Arrays finden und darstellen.

4 Datenanalyse

Es wurden im Projekt unterschiedliche Daten auf ihre Quantität, Qualität und Verwendbarkeit untersucht. In diesem Schritt betrifft die Datenqualität erstmal die äußerlichen Merkmale, die tatsächliche Ähnlichkeit und der Vergleich von Regendaten folgt in einem späteren Kapitel. Für das Projekt

[5] https://github.com/thgnaedi/DeepRain/blob/master/Data/Location%20Konstanz/DeepRain_WRADLIB.ipynb

werden zum einen die tatsächlichen Niederschlagsdaten in Konstanz zum späteren Vergleich benötigt, zum anderen Radardaten, beides möglichst in ein bis zehnminütiger Auflösung und möglichst von aktuell bis einige Jahre in die Vergangenheit reichend. Wetterradardaten werden von öffentlichen und privaten Anbietern erhoben und sind in unterschiedlichen Qualitäten und unterschiedlicher Datenmenge verfügbar. Der E-Mail-Verkehr mit Meteogroup, einem privaten Wetterdienstleiter zeigte, dass die Daten der Wetterstationen Konstanz Südkurier und Konstanz Dettingen von uns nicht verwendet werden konnten. Die Daten von Windy sind zwar öffentlich zugänglich, jedoch nicht downloadbar, welches eine Voraussetzung für verwendete Daten ist. Die Daten die Anbieters Kachelmann sind bis ins Jahr 2015 online verfügbar jedoch müssten hier die Nutzungsrechte eindeutig geklärt werden. Die umfangreichsten Radardaten werden von der Deutschen Wetterstation zur Verfügung gestellt, daher wurde sich im Team für die Verwendung dieser Daten entschieden.

4.1 Verifizierung der Daten mit statista und Wetterkontor

In den Daten des DWD hat die Konstanzer Wetterstation die ID: 02712. Aus dem Zip-File mit der enthaltenen Textdatei können somit die nun aufgezeigten Daten für die Untersuchung entnommen werden. Zu den wichtigsten Informationen gehören hier das Qualitätsniveau der Daten (zwischen 1 und 16) als QN, das Messdatum im Format YYYYMMDDHHMM, die Niederschlagshöhe und den Niederschlagindikator. Die Aktuellen Daten wurden mit einem Python-Script eingelesen und ein paar kleine Statistiken zum Validieren der Daten vorgenommen. Zum Vergleich wurden hierfür die monatlichen Niederschlagsangaben von Statista.com und Wetterkontor.com genommen, allerdings bezogen die sich auf ganz Deutschland.

Die Historischen Daten des DWD sind auch Monatlich als Textdatei gespeichert. Für den Vergleich wurde das Jahr 2018 betrachtet. In den historischen Daten fallen neue Spalten im Header auf, diese sind aber „ungenutzt“ bzw. über alle (geprüft auf Jahr 2017) Einträge identisch. Diese sind: QN, RTH_01 und RWH_01. Die Messwerte wurden wie auch zuvor monatsweise zusammenaddiert das Ergebnis des Vergleichs ist in Tabelle 1 zu sehen. Auffällig ist, dass einige Messwerte sehr gut übereinstimmen. So ist der Niederschlag in den Monaten März und April fast gleich in der referenzierten Messung und der Messung des DWDs. In anderen Monaten, wie im Juni ist der relative Fehler mit 26.79% sehr hoch. Das könnte an gemessenen Extremwerten in den Daten des DWDs liegen, die in der Größenordnung in den Messungen der anderen Anbieter nicht auftauchen. Die hierzu verwendeten Skripte und Quellen sind auf GitHub wiederzufinden[6].

[6] `https://github.com/thgnaedi/DeepRain/tree/master/WetterStation_KN`

Monat	**ref.** l/m^2	**Messung**	**Fehler (abs.)**	**Fehler (rel.)**
Januar	40,1	37,39	-2,71	6,76%
Februar	34,5	37,02	2,52	7,30%
März	40,2	40,3	0,1	0,25%
April	132,4	132,25	-0,15	0,11%
Mai	56,4	52,58	-3,82	6,77%
Juni	95	69,55	-25,45	26,79%
Juli	168,9	171,38	2,48	1,47%
August	155,7	148,42	-7,28	4,68%
September	65	55,23	-9,77	15,03%
Oktober	36,3	36,93	0,63	1,74%
November	76,8	78,49	3,69	4,93%
Dezember	79,5	78,52	-0,98	1,23%

Tabelle 1: Auswertung des absoluten und relativen Fehlers zwischen den Regendaten verschiedener Quellen

4.2 Korrelation der Radardaten mit den Regenmessungen

In diesem Schritt sollte geprüft werden, wie hoch die gemessenen Radardaten mit den Niederschlagsdaten korrelieren. Im optimalen Fall sollten alleine auf Grundlage dieser Analyse Rückschlüsse auf die Lage von Konstanz auf dem Radarbildern durch eine besonders hohe Korrelation der Daten möglich sein. Als Monat für die Untersuchung wurde der Juni 2016 gewählt, da es sich um einen sehr Regenreichen Monat gehandelt hat. Einen wichtigen Anteil an der Korrelationsanalyse hat die Datenaufbereitung in Anspruch genommen. Um einen Vergleich der Niederschlagsdaten mit den Radardaten zu ermöglichen ist es erforderlich, dass die Daten jeweils für dieselben Zeitabstände verfügbar sind. Da die Regenmessungen in minütiger Auflösung vorliegen, wurden sie mit Hilfe zweier Python-Skripte auf eine Datei mit 5-minütiger und eine mit stündlicher Auflösung angepasst. Nun wurde ein Notebook geschrieben, dass die Pixelwerte der Punkte um Konstanz also den Pixelwert 842.406 herum einliest und mit dem Zeitpunkt der Messung und dem tatsächlich gemessenen Niederschlag zu diesem Zeitpunkt in einer Numpy-Matrix speichert. Hier musste eine Einschränkung in Kauf genommen werden, da das einlesen jedes einzelnen Pixels bei 900 x 1100 Pixeln nicht möglich war und daher circa ein Umkreis von 20 Pixeln um Konstanz herum ausgewählt wurde. Bei der ersten Analyse wurde mit 156.377 ein falscher Pixel verwendet, was auf Unklarheiten bei der Achsenbestimmung zurückzuführen war. Die erstellte Numpy-Matrix wurde dann noch auf fehlende Werte untersucht, dieser Anteil hat sich als verschwindend gering herausgestellt, das heißt, dass die Daten bis auf ganz wenige Ausreißer vollständig sind. Für die Analyse der Daten wurde dann ein R-Skript geschrieben. In diesem wurden die Datuminformationen extrahiert, sodass eine Einteilung in Jahr, Monat, Tag, Stunde und

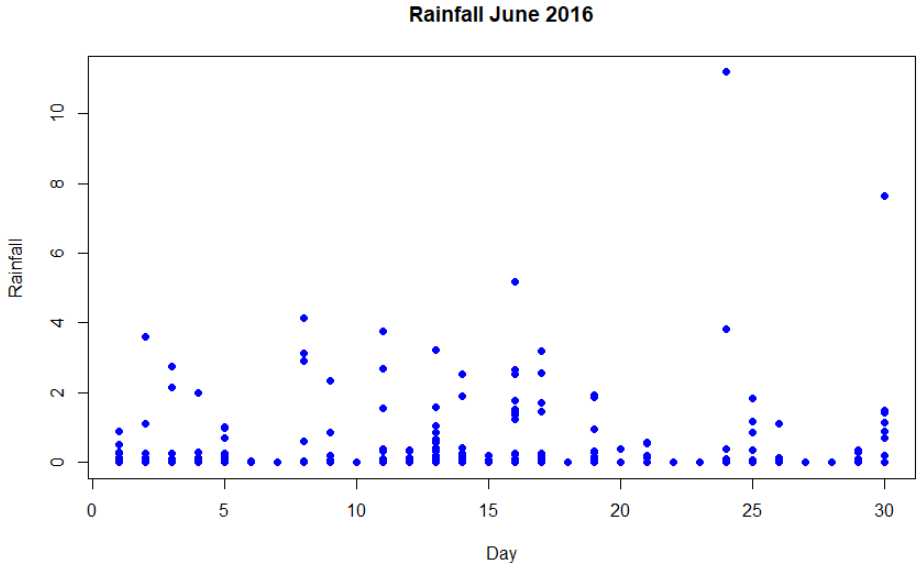


Abbildung 6: Regenfallmengen im Juni 2016 nach Tag

Minute der Datensätze möglich wurde. Dann wurden die die Regenmengen der gemessenen Niederschlagswerte je Tag dargestellt. Hier zeigte sich eine recht gleichmäßige Verteilung der Regenwerte mit einigen gut identifizierbaren Ausreißern z.B. am den 24. Juni mit über zehn Liter pro Quadratmeter, siehe 6. Bei der Werteverteilung der Pixel aus den Radarbildern fällt auf, dass es sehr viele null-Werte und am zweitmeisten den Wert 80 gibt. Werte, die dazwischen existieren sind „10“, „20“, und „40“. Bezüglich der Niederschlagsdaten könnte hier auch das eher niedrige Qualitätsniveau der Stufe 3 Einfluss gehabt haben. Leider konnte zwischen den beiden Datentypen keine Korrelation festgestellt werden. Hierfür kommt eine Reihe von möglichen Gründen in Frage. Da die eigene Betrachtung der Daten, also der händische Abgleich von Zeiten mit Starken Regen in beiden Datengruppen nahelegt, dass die Daten tatsächlich nicht korrelieren kommt in Frage, dass die gemessenen Daten der Wetterstation Konstanz Fehlmessungen beinhalten.

5 Netzwerkarchitekturen

Für den ersten Kontakt mit einem Neuronalen Netz befassten wir uns mit dem MNIST Datensatz. Hierbei geht es um eine Klassifizierung von handschriftlichen Zahlen in die zugehörigen zehn Klassen. Um Rechenzeit zu sparen, da uns anfangs noch keine GPU zur Verfügung stand, haben wir das Netz sehr klein gehalten. Das Netz für welches wir uns entschieden haben, besteht aus einem Convolutional Layer mit 32 3x3 Kerneln, diese werden im späteren

Kapitel 5.1 genauer erläutert. Anschließend folgt ein 2x2 MaxPooling Layer, auch darauf gehen wir später noch ein. Um die Klassifikation vornehmen zu können, wird anschließen ein Flatten Layer eingebunden und ein Fully-Connected Layer mit 128 Output-Neuronen. Diese 128 Features werden dann über ein weiteres Fully-Connected Layer auf die 10 Ausgabeklassen verrechnet. Als Aktivierungsfunktion dient ein softmax, welcher dafür sorgt, dass die Summe aller Ausgaben eins ergibt. Somit können die Outputs als Wahrscheinlichkeiten gedeutet werden, die wahrscheinlichste Klasse wird dann als Vorhersage verwendet. Das hier erwähnte Fully-Connected Layer ist die einfachste Art eines Layers, es besteht aus mehreren Eingabeneuronen und einigen Ausgabeneuronen. Jedes Ausgabeneuron ist hierbei mit jedem Eingabeneuron verbunden. Um den Wert eines Ausgabeneurons zu berechnen wird dann einfach eine gewichtete Summe aller Eingaben berechnet. Die hierbei verwendeten Gewichte werden über die Fehlerfunktion gelernt.

Für die kurzzeitige Wettervorhersage entschieden wir uns für zwei grundlegende Aufgabenstellungen. Die erste Herangehensweise war das Vorhersagen weiterer Radarbilder in der Zukunft. Die Architektur muss also mehrere zusammengehörende Radarbilder als Eingabe verarbeiten und als Ausgabe wieder ein oder mehrere Zeitschritte liefern. Für diese Aufgabenstellung eignet sich sowohl ein klassisches CNN (Kaptel 5.1), als auch ein UNet(Kaptel 5.2). Um uns zwischen diesen Architekturen zu entscheiden nahmen wir einen kurzen Test vor, in welchem beide Architekturen mittels MSE einen Zeitschritt (5 Minuten) vorhersagen sollten.

Die Abbildung 7 zeigt die Lernkurve der beiden Architekturen auf die identische Problemstellung. Das UNet lernt in diesem Beispiel deutlich besser, weshalb die Vorhersage von Radarbildern in Zukunft mit einem UNet behandelt wird. Beide UNet Architekturen haben zu Beginn eine deutlich schlechtere Vorhersage als das CNN, bereits nach 60 Epochen ist die einfache UNet Architektur gleich gut, während das grüne UNet die Performance sogar übertrifft und nach 100 Epochen die besten Ergebnisse liefert.
Die zweite Herangehensweise ist eine Klassifikation, hierbei geht es nicht darum, das exakte Radarbild vorherzusagen, sondern einzuordnen, ob es regnet oder nicht. Diese Aufgabe wurde als einfache Klassifikation für Konstanz, als auch als pixelweise Klassifikation, für alle eingehenden Pixel durchgeführt. Für die Aufgabe der Klassifikation jedes Pixels wurde wieder ein UNet verwendet. Bei der Aufgabe der einfachen Klassifikation für Konstanz kamen beide Architekturen zum Einsatz.

5.1 CNN

Ein Convolutional Neural Networks im klassischen Sinne ist ein Netzwerk mit mehreren Convolutional-Layer, häufig in Verbindung mit Pooling-Layer. Ein Convolutional-Layer besteht aus mehreren Filtern. Die Filter berechnen ein Output in Abhängigkeit mehrerer benachbarter 'Pixel'; die Größe der

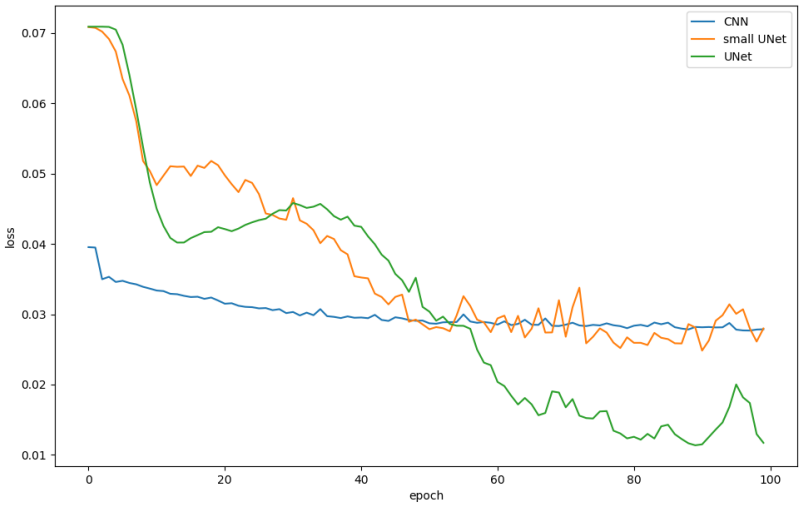


Abbildung 7: Gezeigt ist die Lernkurve der Validierungsdaten auf 100 Epochen mit unterschiedlichen Netzarchitekturen. Die verwendeten Trainingsdaten sind 1000 synthetische Bilder welche eine wandernde Regenfront simulieren sollen. Das CNN ist etwa gleich gut wie das einfache UNet. Eine tiefere UNet Architektur (grün) erreicht eine noch bessere Performance. Die Architekturen sind in nachfolgenden Kapiteln genauer erläutert.

berücksichtigten Region hängt von der Filter- bzw. Kernelgröße ab. Vorteile von Convolutional Layers sind zum einen, dass Nachbarschaften berücksichtigt werden, was gerade bei Bildern sehr sinnvoll ist, aber auch, dass der Speicherbedarf sehr gering ist, da nur eine kleine Anzahl an Gewichten für das komplette Bild verwendet werden müssen. Ein Pooling-Layer reduziert die Feature-Größe indem jeweils nur das stärkste Signal einer Region weitergegeben werden.

Für die Vorhersage der Radardaten haben wir uns für ein sehr einfaches CNN (Convolutional Neural Network) entschieden. Als Eingabe erwartet es mehrere Zeitschritte um darauf eine Vorhersage zu treffen. Der Input ist also dreidimensional wobei die dritte Dimension die Zeitschritte und die anderen Dimensionen die Bildauflösung beschreiben. Die Eingabe wird durch sechzehn Convolution Kernel der Größe 5x5 verrechnet. Anschließend folgen zweiunddreißig weitere 5x5 Kernel. Des weiteren ist ein optionales Dropout Layer eingebunden, welches nur zu Testzwecken verwendet wurde. Die Performance hatte sich dadurch aber nicht bemerkenswert verändert. Abschließend kommt ein Kernel, welcher die nun entstandenen Features zu einem Bild zusammenfasst. Hierzu wird ein 3x3 Kernel verwendet. Alle Layer sind mit einer ReLu als Aktivierungsfunktion ausgestattet. Die Performance ist

weniger gut, als ein UNet kann aber mit der sehr einfachen UNet Architektur mithalten. Zu sehen ist das in Abbildung 7 wo die blaue Kurve dem Lernverhalten der hier beschriebenen CNN Architektur entspricht.

Für Klassifikation des Konstanz-Pixels entschlossen wir uns dazu, unser ursprüngliches CNN zum Lernen des MNIST Datensatz wiederzuverwenden. Dieses Netz besteht lediglich aus zweiunddreißig 3x3 Kerneln mit ReLu Aktivierungsfunktion. Anschließend folgt ein 2x2 MaxPooling und ein Flatten Layer. Die nun „flachen“ Daten werden durch ein FullyConnected Layer auf 30 Neuronen reduziert. Darauf folgt ein Dropout Layer mit 20% welches Overfitting verhindern soll. Abschließend folgt ein Layer, dass die Features auf drei Ausgabe-Neuronen verrechnet. Damit die Klassifizierung einfacher zu interpretieren ist, wird als Aktivierungsfunktion ein SoftMax verwendet.

5.2 UNet

Ein Unet ist eine spezielle Form eines CNN. Es hat seinen Namen durch die 'U-Förmige' Architektur (siehe Abbildung 8).

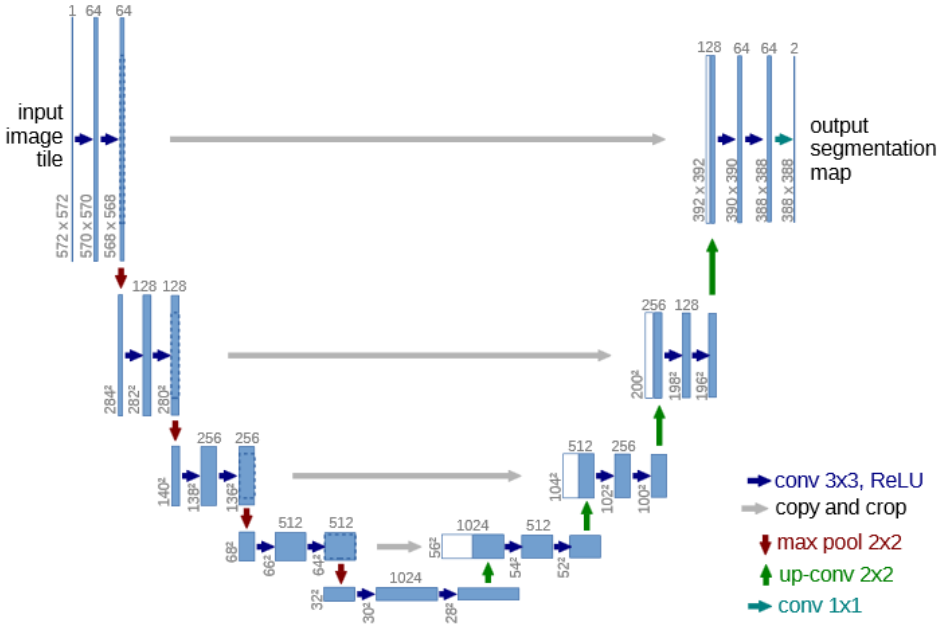


Abbildung 8: Die hier abgebildete Architektur entstammt dem Paper zur biomedizinischen Bildsegmentierung [RFB15]. Hier abgebildet ist eine typische UNet-Architektur, welche von den Eingabebildern (oben links) zu der Ausgabe (oben rechts) die Features stets verkleinert (rote Pfeile) und im späteren Verlauf auch wieder vergrößert (grüne Pfeile). Da bei dem wieder vergrößern der Daten nicht alle Informationen wiedergewonnen werden können, gibt es auch Querverbindungen (graue Pfeile) welche die zuvor errechneten Features an ein späteres Layer weitergibt.

Das Unet besteht aus mehreren Convolutional Layern, mit anschließendem Pooling. Hierdurch wird die Featuregröße immer weiter reduziert. Ab einem bestimmten Punkt wird die Featuregröße wieder aufgeblasen, um am Ende für den Output die gleiche Größe wie für den Input zu besitzen. Beim 'Aufblasen' kommt ein Upsampling Layer zum Einsatz. Da allerdings beim Vergrößern der Features nicht alle Informationen wiederhergestellt werden können, gibt es auch noch horizontale Verbindungen, durch welche zuvor erstellte Features später weiterverwendet werden können. Diese Architektur ist sehr ähnlich zu einem Autoencoder und wurde erstmals für biomedizinische Zwecke verwendet. Da sich mit dieser Architektur mehrere Bilder einlesen und auch beliebig viele Bilder ausgeben lassen, versuchen wir damit die Wettervorhersage für mehrere Zeitschritte zu lösen.

Um eine geeignete Architektur auszuwählen haben wir ein einfaches Testszenario aufgebaut, welches unterschiedliche Netzarchitekturen zu lösen hatten. Die hierfür verwendeten Daten waren aus jeweils 100x100x5 Pixeln großen synthetischen Daten aufgebaut. Das erste U-Net besteht aus lediglich einem Upsampling Layer. Die Architektur sieht wie folgt aus: Anfangs verarbeiten zweiunddreißig 5x5 Convolutional Layer mit Padding die Eingabe. Diese wird über ein 3x3 MaxPooling reduziert. Die reduzierten Daten werden durch ein 3x3 Upsampling wieder in die ursprüngliche Größe zurück gewandelt und mit den Features vor dem MaxPooling verbunden. Die so entstehenden 100x100x64 Features werden anschließend über einen 1x1 Convolutional Layer zu einem Ausgabebild zusammengefasst. Die zu diesem Netz gehörende Lernkurve ist in Abbildung 7 zu sehen. Die Performance unterscheidet sich kaum zu der eines klassischen CNNs. Daher versuchen wir die Architektur tiefer zu gestalten.

Das finale UNet wie es in Abbildung 7 in grün zu sehen ist besteht aus mehreren Ebenden. Zunächst wird die Eingabe durch ein Convolutional-Layer mit zehn 3x3 Kerneln und anschließender ReLU als Aktivierungsfunktion geleitet. Die so entstehenden Features werden als Conv01 bezeichnet und später wieder verwendet. Die Features werden anschließend durch ein MaxPooling-Layer der Größe 2x2 verkleinert. Anschließend folgt ein weiteres Convolutional-Layer mit zwanzig 3x3 Kerneln. diese Features werden im weiteren als Conv02 bezeichnet. Auch hierauf folgt wieder ein 2x2 MaxPooling-Layer mit anschließend einem weiteren Convolutional-Layer mit zwanzig Kerneln der Größe 3x3. Diese Features werden als Conv03 weiterverwendet. Die tiefste Ebene in dieser Architektur besteht wieder aus einem Convolutional-Layer mit zwanzig Kerneln der Größe 3x3. Auch hier folgt ein 2x2 MaxPooling. Diese Features heißen Conv04. Ab jetzt werden die Features wieder größer. Ein Upsampling-Layer der Größe 2x2 sorgt dafür, dass die Features mit den zuvor erstellten Conv04 Features verbunden werden können. Dies geschieht durch ein Concatenate-Layer, was die beiden Tensoren zu einem Größeren verbindet. Die so entstandenen Features werden wieder durch ein Upsampling-Layer der größe 2x2 vergrößert und mit den Conv03 Featu-

res verbunden. Auch diese Daten werden wieder über ein Upsampling-Layer der größe 2x2 verrechnet und anschließend mit den Conv02 Features verbunden. Abschließend werden die Daten durch ein weiteres 2x2 Upsampling-Layer verrechnet und mit den Conv01 Features verbunden. Abschließend kommt ein als Ausgabe ein Convolutional-Layer mit einem 1x1 Kernel der alle Features zu einem Graustufenbild zusammenfasst. Alle Hier erwähnten Convolutional-Layer haben eine ReLU als Aktivierungsfunktion, wodurch negative Werte ausgeschlossen werden. Des weiteren besitzen diese Layer auch Padding um die Bildgröße beizubehalten.

Alle oben genannten Architekturen werden mit dem Adam-Optimizer trainiert. Je nach Aufgabenstellung dient als Fehlerfunktion der MSE, beim Vorhersagen der Radardaten, oder die Categorical Cross-Entropy, beim Lösen des Klassifikationsproblems.

6 Regenradar-Vorhersage

Ziel dieser Aufgabe ist es, mit Hilfe eines UNets eine kurzzeit-Regenvorhersage zu erreichen. Die hier verwendeten Daten sind im Kapitel 2.2.1 genauer erklärt. Ziel des Netzes ist es die Baseline zu schlagen. Als Baseline zählen wir die Vorhersage des zeitlich letzten Bildes. Es wird bei der Baseline also davon ausgegangen, dass sich an dem Regenverhalten nichts ändert. Im ersten Unterkapitel wird das UNet verwendet um eine fünf-minuten Vorhersage zu machen. Es werden verschiedene Erweiterungen an dem bisherigen UNet angewandt, um eine eventuelle Steigerung der Performance zu erreichen. Im zweiten Unterkapitel soll eine Vorhersage bis zu 35 Minuten in die Zukunft möglich werden. Die Auswertung wird im jeweiligen Kapitel vorgenommen.

6.1 Vorhersage von fünf Minuten

Um eine Radar-Vorhersage für fünf Minuten zu erreichen, werden Daten wie in Kapitel 2.2.1 in Abbildung 3 gezeigt verwendet. Die Eingabedaten bestehen aus fünf aufeinander folgende Bilder, welche bis zu 20 Minuten in die Vergangenheit reichen. Das Radarbild im Label ist fünf Minuten in der Zukunft.

Die verwendete Architektur entspricht der, des in Kapitel 5.2 beschriebenen UNets. Die verwendete Fehlerfunktion ist der MSE. Das Training beschränkt sich auf 1900 Samples aus dem Jahr 2016, zum Validieren wurden die letzten 50 Samples aus dem Dezember genommen. Die Validierungsdaten kommen somit in keiner Weise während des Trainings vor.

Die erste Analyse der Lernkurve 9 zeigt, dass das Netz den Fehler (MSE) während der ersten 20 Epochen sehr stark verbessern kann. Auch bei der letzten Epoche scheint der Validation-Loss noch weiter zu fallen. Für dieses Einstiegsbeispiel soll das Ergebnis aber zunächst genügen. Es gilt noch zu

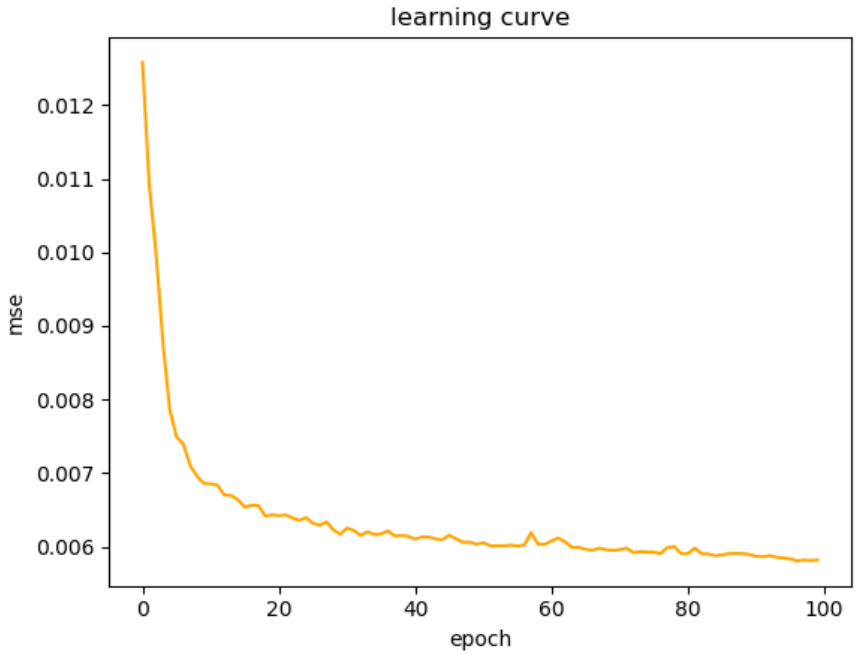


Abbildung 9: Die dargestellte Lernkurve bezieht sich auf die fünf Minütige Radarvorhersage mittels MSE. Zum Trainieren wurden 1900 Samples aus dem Jahr 2016 verwendet. Abgebildet ist der Validation-Loss welcher auf weitere 50 Samples berechnet wurde. Die ersten Epochen ist eine sehr starke Verbesserung des Fehlers zu erkennen, auch nach 100 Epochen scheint die Fehlerkurve noch zu fallen.

klären, ob das Netz auch tatsächlich eine sinnvolle Vorhersage liefert und ob es in der Lage ist die Baseline zu schlagen.

Die Ausgabe sowie die Baseline sind in Abbildung 10 für ein Validierungssample zu sehen. Das Netz hat die Bewegung der Regenfront gut vorhergesagt, allerdings ist die Regenintensität meist unterschätzt. Besonders den starken Niederschlag, kann das Netz nicht vorhersagen. Das ist in der linken Abbildung an den weißen Punkten unten links gut zu erkennen. Das Netz sagt hier eher einen dunkleren Grauton als das Label vorher. Um dennoch bewerten zu können, ob das Netz im Schnitt besser als die Baseline ist, wird für beide Vorhersagen der mittlere quadratische Fehler zum Label (im Bild 'timestep 5') berechnet. Das Resultat zeigt für die Baseline einen Fehler von 0.00165 und für das Netz etwa 0.0008. Die Baseline macht im Schnitt also einen doppelt so großen Fehler wie unser Netz. Diese Berechnung wurde für alle 50 Validierungssamples vorgenommen. Das Resultat ist in Abbildung 11 zu sehen.

Einige Validierungsdaten konnten nahezu ohne Fehler vorhergesagt werden. Dies sind Bilder, bei welchen kein Regen mehr in dem Label zu sehen

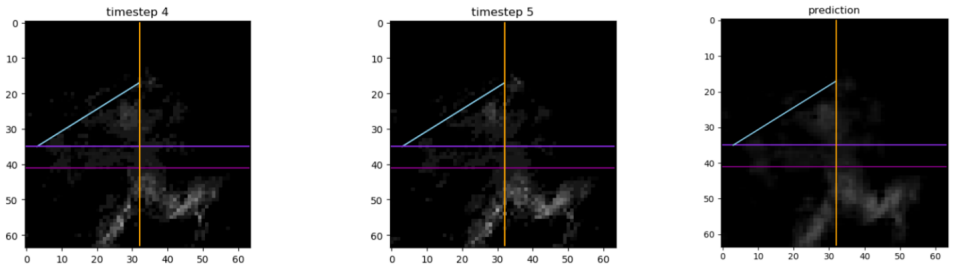


Abbildung 10: Hier abgebildet sind mehrere Radarbilder. Je heller ein Pixel dargestellt ist, desto stärker regnet es an dieser Stelle. Das hier als timestep 4 bezeichnete Bild ist das letzte Eingangs-Bild und spiegelt somit die Baseline wieder. Rechts daneben abgebildet ist das Label, welches fünf Minuten später repräsentiert. Ganz rechts ist die Vorhersage des Netzes zu sehen. Die eingezeichneten Linien dienen nur der Orientierung um ein Vergleich der Bilder sowie das schätzen der Bewegung der Regenfront zu ermöglichen. Es ist deutlich zu sehen, dass die Vorhersage des Netzes stark verschwommen ist. Die Bewegung der Regenwolken wurde allerdings sehr gut vorhergesagt.

ist. Hat auch die Baseline keinen Fehler gemacht, war bereits im letzten Zeitschritt kein Regen mehr vorhanden. Alle anderen Samples beinhalten Regenwetter in den Labels. Hier ist die Baseline auch durchgehend schlechter. Dies kommt vor allem daher, dass sich Regenfronten sowohl verschieben, als auch die Regenmenge innerhalb fünf Minuten deutlich wechseln kann. Selbst eine Verschiebung von lediglich einem Pixel resultiert in einem deutlichen Fehler.

Als nächsten Schritt wollen wir die Performance des Netzes noch weiter steigern. Hierzu werden verschiedene Ansätze versucht. Die zugehörigen Lernkurven sind in Abbildung 12 zu sehen. Die bisher verwendete Architektur ist in Blau eingezeichnet. Der erste Versuch war es, das Output-Layer statt mit einem 1x1 Kernel mit einem 3x3 Kernel auszustatten, um regionale Einflüsse besser in das Ergebnis mit aufzunehmen. Das Ergebnis hiervon ist ein vor allem Anfangs deutlich schnelleres Sinken des Fehlers. Nach 80 Epochen ist die Performance nur geringfügig besser. Ein weiterer Versuch ist das Erweitern des Netzwerkes um eine neue Ebene. Diese Ebene besteht wieder aus 20 3x3 Kernel, das nun tiefere Netzwerk ist in Grün dargestellt und erreicht die selbe Performance wie das ursprüngliche Netz. Eine deutliche Vergrößerung der neuen Schicht (von 20 auf 60 Kernel) kann die Performance minimal verbessern. Die zugehörige Lernkurve ist in Rot eingezeichnet. Ein anderer Ansatz ist das Ersetzen der ReLU Aktivierungsfunktionen durch Sigmoid Aktivierungsfunktionen. Hierdurch erhöht sich der Rechenaufwand enorm, da die ReLU lediglich negative Werte abschneidet und eine Sigmoid-Funktion für jeden Wert einen neuen Funktionswert berechnen muss. Die Performance ist in Violett eingezeichnet und zeigt keine Verbesserung. Auch die Fehlerfunktion wird testweise ausgetauscht, statt dem MSE wird der MAE verwendet, welcher lediglich ohne Quadrate arbeitet. Aber auch hier

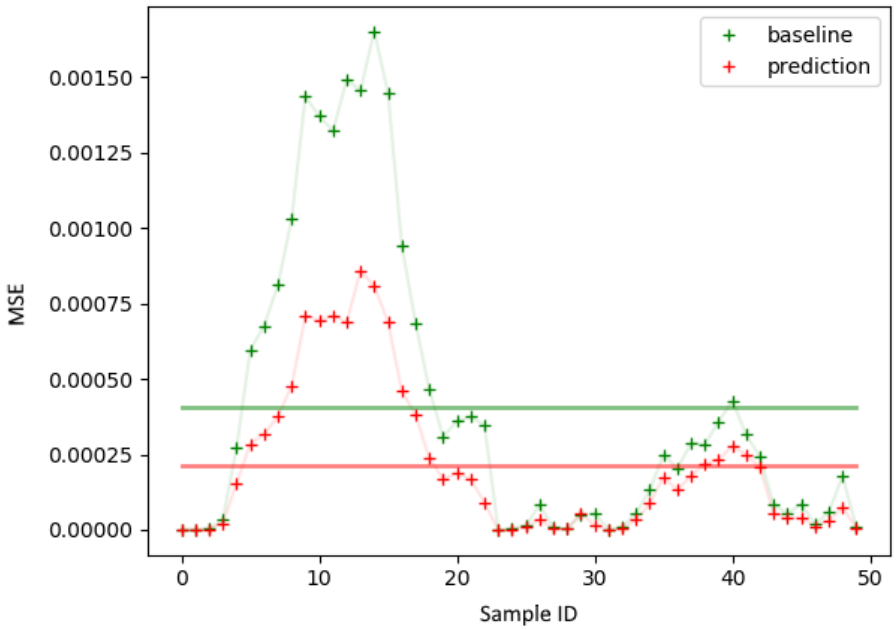


Abbildung 11: Auf der X-Achse sind die 50 Validierungssamples aufgelistet. Die Y-Achse entspricht dem MSE für ein Sample (Quadratische Abweichung über alle Pixel gemittelt). Für jedes Sample wurde der MSE jeweils für die Vorhersage des Netzes (rot) und der Vorhersage der Baseline (grün) berechnet. Die Performance der Baseline konnte überboten werden.

(braune Kurve) ist keine Verbesserung zu erkennen. Als letztes wird der Adam-Optimizer ausgetauscht. Die Alternative ist der Nestreov-Optimizer, welcher in der Lernkurve (Pink) deutliche Sprünge aufweist. Eine nennenswerte Änderung konnte allerdings mit keiner Optimierung erreicht werden. Lediglich ein Unterschied in der Rechenzeit ist bemerkbar. Da die Netze zeitweise auch auf der CPU trainiert werden müssen, Da benötgte Hard- und Software nur teilweise während des Projekts zur Verfügung standen, wird weiterhin das Blaue bisher verwendete UNet verwendet.

6.2 Vorhersage von fünfunddreißig Minuten

Da die Baseline also mit dieser Architektur geschlagen werden kann, ist das nächste Ziel mehr als nur 5 Minuten vorherzusagen. Die hierfür verwendeten Daten bestehen aus 7 Zeitschritten im Label, also bis zu 35 Minuten in die Zukunft. Die Netzarchitektur muss hierfür lediglich am Output-Layer angepasst werden. Hierzu wird der 1x1 Kernel gegen sieben 1x1 Kernel ausgetauscht. Die Fehlerfunktion bleibt wie bisher der MSE, eine unterschiedliche Gewichtung der Zeitschritte wird nicht vorgenommen. Für das Training wird

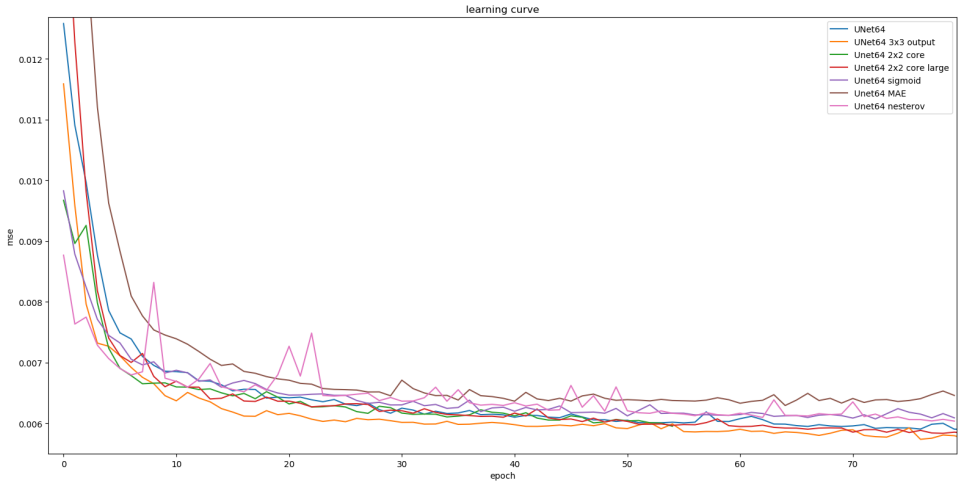


Abbildung 12: Um die Performance unserer UNet-Architektur weiter zu steigern wurden unterschiedliche Änderungen versucht. Die blaue Kurve zeigt das bisher verwendete Netz. Ein größerer Kernel am Output-Layer (gelb) verbessert die Performance ein wenig. Eine tiefere Netzwerkarchitektur (grün) steigert die Performance erst bei verwenden von sehr vielen Kernel (rot). Das Austauschen der Aktivierungsfunktionen gegen eine Sigmoid-Funktion (violett) ändert außer an der Berechnungsgeschwindigkeit nichts. Das Austauschen der Fehlerfunktion gegen den MAE (braun) sowie das Ändern des Optimierers zum Nesterov (pink) zeigen keine Verbesserung. Insgesamt werden nur deutliche Änderungen an der Rechenzeit festgestellt. Die Ergebnisse bleiben von der Performance vergleichbar.

nun auf alle zur Verfügung stehenden Daten zugegriffen. Dies entspricht 7500 Trainings- und 623 Validierungssamples.

Die zugehörige Lernkurve ist in Abbildung 13 zu sehen. Da die Performance auf das Validierungsset (blaue Kurve) nicht weiter verbessert wird, kann mit der Auswertung begonnen werden. Zunächst wird das Netz wieder mit der Baseline verglichen. Hierfür wird jeder Zeitschritt aus dem Label mit dem letzten Zeitschritt der Eingabe-Daten verglichen. Da die Baseline davon ausgeht, dass das Wetter so bleibt wie es ist, ist zu erwarten, dass die Baseline mit zunehmender Zeit stets schlechter wird. Auch das Netz sollte mit zunehmender Zeit Probleme bekommen die Daten noch korrekt vorherzusagen. Eine direkter Vergleich pro Zeitschritt zwischen Baseline und Netzvorhersage wird in Abbildung 14 dargestellt. Die obere Zeile der Abbildung enthält die Verteilung der Vorhersage, während die untere Zeile die der Baseline enthält. Die Spalten geben die Zeitschritte an, zwei untereinander liegende Histogramme stellen die selbe zeitliche Vorhersage dar. In den Histogrammen ist die Verteilung der Summe der quadratischen Fehler für alle Validierungsdaten dargestellt. Es wurde also pro Vorhersage eine Summe über alle quadratischen Abweichungen zwischen Vorhersage und Label, pro Zeitschritt, berechnet. Schwarz eingezeichnet ist der Mittelwert, die beiden

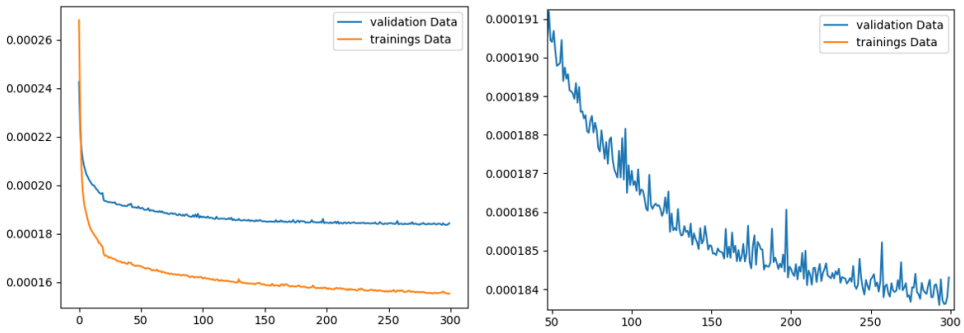


Abbildung 13: Die abgebildete Lernkurve zeigt, dass das Netz erst nach ca. 300 Epochen sich nicht weiter verbessert. Die rechte Abbildung ist ein Ausschnitt aus der Linken, bei welcher das Fallen der Lernkurve besser sichtbar ist.

grauen Linien entsprechen dem 25%-, bzw 75%-Quantil. Es ist deutlich zu erkennen, dass der Mittelwert des Fehlers mit zunehmender Zeit größer wird, bzw. nach rechts wandert. Dies war so auch zu erwarten, da mit zunehmender Zeit stets neue Faktoren Einfluss auf das Wetterverhalten haben. Interessanter ist der Vergleich der Spalten. Der Mittelwert des Fehlers ist durchgehend besser bei der Vorhersage des Netzes als bei der Baseline. Mit zunehmender Zeit wird dieser Unterschied allerdings sehr gering.

Den Zahlen nach ist das Ziel die Baseline zu schlagen also geschafft. Um aber wirklich ein Fazit ziehen zu können werden die Daten und Vorhersagen in nachfolgender Abbildung 15 und 16 dargestellt. Die hier verwendeten Daten und Label entstammen dem Validierungsset, welches Radardaten aus dem Jahr 2014 umfasst. Sie wurden in keinem Trainingsschritt verwendet. Die Abbildungen zeigen deutlich, dass das Netz die ersten fünf Minuten sehr genau vorhersagen kann. Das ausgegebene Bild hat unterschiedliche Niederschlagsmengen für jeden Pixel vorhergesagt. Mit zunehmender Zeit lässt diese Genauigkeit aber stark nach. So ist im letzten Ausgabebild sehr häufig für einen ganzen Bereich die selbe Niederschlagsmenge vorhergesagt. Es wirkt, als würde die Auflösung der Vorhersage mit zunehmender Zeit abnehmen. Durch die schwache Auflösung werden viele Pixel nicht ganz korrekt wiedergegeben, was den Fehler erhöht. Der so entstehende Fehler ist aber dennoch geringer, als der Fehler welchen die Baseline macht, indem sie exakte Niederschlagsmengen für jeden Pixel vorhersagt. Unser Netz schlägt die Baseline also nicht dadurch eine exaktere Vorhersage zu treffen, sondern durch eine verschwommene Regenfront, welche sich entsprechend der eingegangenen Daten weiter bewegt und dabei auch stets auseinander driftet.

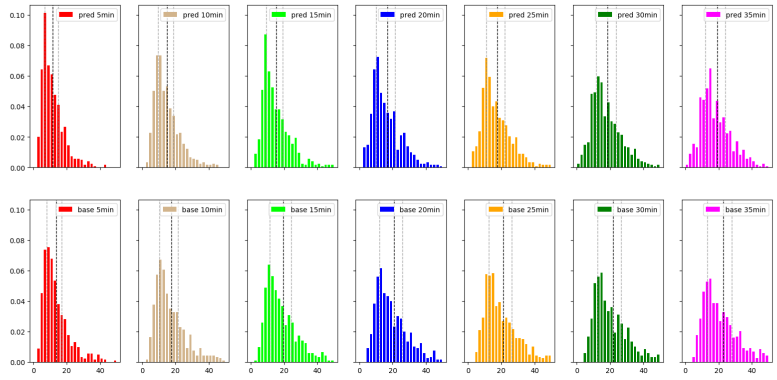


Abbildung 14: Ein Vergleich zwischen Baseline und Vorhersage. In den Spalten sind die zeitlichen Schritte von 5 bis 35 Minuten Vorhersage enthalten. In der oberen Zeile ist die Vorhersage des Netzes und in der unteren die der Baseline abgebildet. Die Histogramme zeigen die Summe der Quadratischen Fehler pro Zeitschritt. Die schwarzen Linien zeigen den Mittelwert der Verteilung, während die grauen Linien das 25%- sowie das 75% Quantil darstellen. Mit zunehmender Zeit wandert die Verteilung weiter nach rechts, die Vorhersagen stimmen also weniger mit dem Label überein.

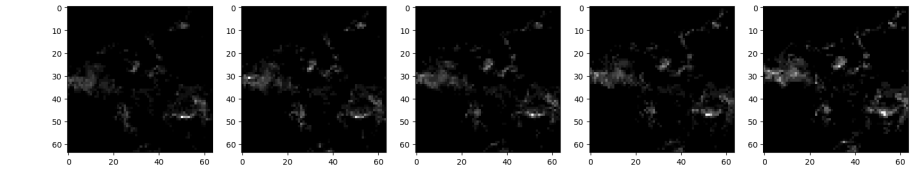
7 Klassifizierung

Statt die genaue Regenmenge in 35 Minuten vorherzusagen, stellten wir drei Kategorien auf: kein Regen ($= 0mm$), wenig Regen ($\leq 8mm$) und viel Regen ($> 8mm$). Diese Kategorien haben wir als One-Hot-Vector kodiert. ‘[1, 0, 0]‘ entspricht hierbei kein Regen, sodass man aus der ersten Dimension der Vorhersage einfach ein Vorschaubild generieren kann aus dem man gleich feststellen kann, ob es am jeweiligen Pixel regnet oder nicht.

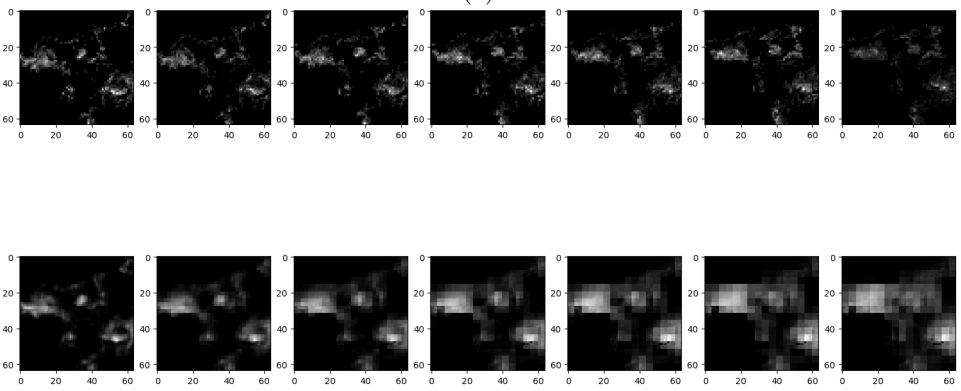
Für das Training mit Kategorien kann man nicht mehr den MSE verwenden, hier würde selbst nach 80 Epochen nur „kein Regen“ vorhergesagt. Stattdessen wurde als Loss-Funktion die „Categorical Crossentropy“ von Keras verwendet. Die binäre Crossentropy können wir nicht verwenden, da wir mehr als zwei Kategorien verwenden. Die „Categorical Crossentropy“ funktioniert relativ gut, aber es wird ein Blob vorhersagt, der etwas über den Bereich ragt, in dem es eigentlich regnet.

Danach wurde noch die Aktivierungsfunktion für den Output-Layer Sigmoid durch Softmax ersetzt. Dadurch erscheint das Vorschaubild etwas verwaschener, aber der Blob um das Regengebiet wird kleiner und die Differenz zum Referenzbild wird kleiner.

Wenn man die Aktivierungsfunktion der Hidden-Layer (von ReLu) zu Tanh verändert, verbessert sich auch die Kategorisierung: der Blob nähert sich weiter dem Regengebiet aus dem zu vorhersagendem Bild an, ist aber

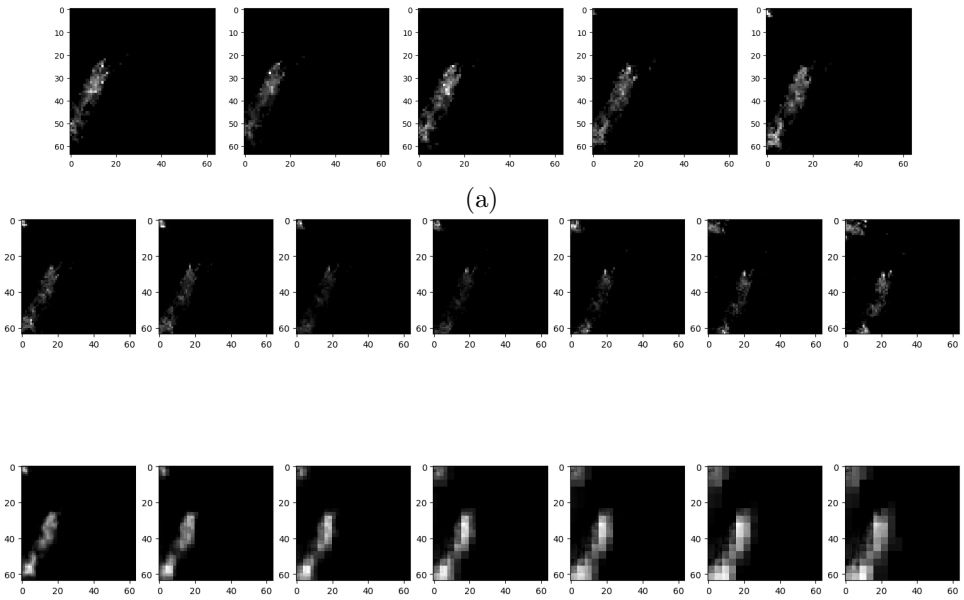


(a)



(b)

Abbildung 15: Die in (a) gezeigten Bilder entsprechen den eingehenden 25 Minuten an Radardaten. In (b) sind die 35 Minuten Label sowie darunter die 35 Minuten Vorhersagen abgebildet. Mit zunehmender Zeit wird die Vorhersage ungenauer. Während das erste Bild das Label noch gut wiedergeben kann, besteht das siebte nur noch aus sehr groben Strukturen.



(a)

(b)

Abbildung 16: Die in (a) gezeigten Bilder entsprechen den eingehenden 25 Minuten an Radardaten. In (b) sind die 35 Minuten Label sowie darunter die 35 Minuten Vorhersagen abgebildet. Im Gegensatz zu Abbildung 15 kann hier anhand der oberen linken Ecke sehr gut eine Bewegung wahrgenommen werden. Obwohl die Regenfront erst in den letzten beiden Eingabebildern erscheint, kann sie in den sieben Ausgabebildern vorhergesagt werden. Zwar ist auch hier die Vorhersage sehr ungenau, aber die Bewegungsrichtung ist korrekt vorhergesagt.

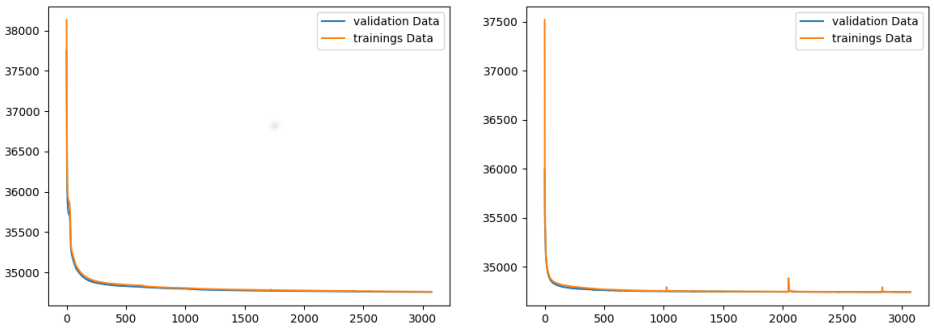
immer noch merkbar größer und franst an den Kanten aus.

Als nächstes wird die Metrik „categorical_accuracy“ verwendet, um die Vorhersage zu überwachen. Dadurch kann der Fortschritt beim Trainieren besser überwacht werden.

7.1 Training

Für das Training wurden alle Daten der 18 Jahre verwendet, partitioniert in Trainings- und Evaluationsdaten. Als Lossfunktion wurde der MSE verwendet, der den Unterschied über alle Pixel kumuliert, weswegen so hohe Werte in den Abbildungen vorkommen (1100x900 Pixel). Da das Training auf der GPU weniger als 10 Sekunden pro Epoche dauert, wurden gleich 3072 Epochen trainiert.

In Abbildung 17 sind die Lernkurven des Trainings nebeneinander dargestellt, links steht das Training des Netzes mit Softmax als Aktivierungsfunktion des Hidden Layer und rechts ist das selbe Netz mit TanH als Aktivierungsfunktion. Man sieht jedoch, dass es ab etwa 1500 Epochen (x-Achse) keine Verbesserungen mehr gibt.



(a) Lernkurve (Hidden layer: Softmax) (b) Lernkurve (Hidden layer: Tanh)

Abbildung 17: Gegenüberstellung zweier Lernkurven von verschiedenen Aktivierungsfunktionen der Hidden Layer. Auf der Y-Achse ist der kumulierte MSE über alle Pixel. Ab etwa der tausendsten Epoche gibt es kaum noch eine Verbesserung. Bei Tanh flacht die Kurve schneller ab.

7.2 Auswertung

Zu den Netzen mit den verschiedenen Aktivierungsfunktionen wurden jeweils eine Confustion-Matrix erstellt (siehe Tabellen 2 und 3). An beiden Matrizen kann man sehen, dass beide Aktivierungsfunktionen für einzelne Pixel statistisch ähnliche Ergebnisse liefern.

Bei beiden Matrizen gleicht sich, dass falls kein Regen vorhergesagt wird, in 96,1% respektive 96,3% auch tatsächlich kein Regen eintrifft. Wird wenig

Regen vorhergesagt, ist die Unsicherheit recht groß: zu rund 20% regnet es gar nicht oder zu etwa 16% regnet es stark. Zu einem Fünftel kann es also sein, dass hier die Vorhersage „Regen“ nicht eintrifft. Wird starker Regen vorausgesagt, trifft zu gut 68% auch starker Regen ein, zu fast 26% wenig Regen oder gut 5% kein Regen. Hier trifft die Vorhersage „Regen“ also zu 95% ein.

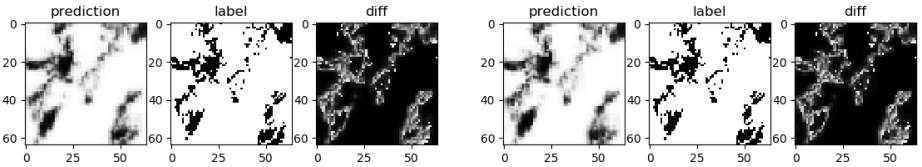
		Vorhersage		
		Kein Regen	**Wenig Regen**	**Viel Regen**
Daten	**Kein Regen**	2227245 (96,1%)	33383 (19,9%)	3762 (5,7%)
	Wenig Regen	81116 (3,5%)	106434 (63,4%)	17077 (25,9%)
	Viel Regen	9695 (0,4%)	27930 (16,7%)	45166 (68,4%)

Tabelle 2: Confustion-Matrix zur 35 Minuten Regenvorhersage (drei Kategorien, Aktivierungsfunktion der Hidden Layer: Softmax), Vorhersage für kein Regen sehr gut, die für wenig und viel Regen sind unsicherer.

		Vorhersage		
		Kein Regen	**Wenig Regen**	**Viel Regen**
Daten	**Kein Regen**	2225229 (96,3%)	35527 (20,4%)	3634 (5,4%)
	Wenig Regen	76988 (3,3%)	110399 (63,5%)	17240 (25,8%)
	Viel Regen	8849 (0,4%)	27969 (16,1%)	45973 (68,8%)

Tabelle 3: Confustion-Matrix zur 35 Minuten Regenvorhersage (drei Kategorien, Aktivierungsfunktion Hidden Layer: Tanh), die Werte sind ähnlich wie die der Aktivierungsfunktion Softmax (Tabelle 2).

Bei der Vorhersage von zusammenhängenden Niederschlagsmengen auf einer Karte gibt es Unterschiede zwischen den Aktivierungsfunktionen der Hidden Layer, während es auf die Wahrscheinlichkeit der einzelnen Pixel kaum Unterschiede gab. Das UNet mit TanH als Aktivierungsfunktion der Hidden Layer erzeugt größere (und leicht rundere) Blobs (Abbildung 18b). Bei Softmax werden schräge Kanten besser vorhergesagt (Abbildung 18a).



(a) Hidden layer activation: Softmax (b) Hidden layer activation: Tanh

Abbildung 18: Vergleich von Aktivierungsfunktionen der Hidden-Layer nach je 3072 Epochen. Aktivierungsfunktion der Output Layer ist jedes mal Softmax. Bei Tanh sind die Blobs rundlicher und haben härtere Kanten.

7.2.1 Training mit zwei Kategorien

Daraufhin versuchten wir die Vorhersage mit nur zwei Kategorien (Regen / kein Regen) durchzuführen um festzustellen, ob die Vorhersage auf diese Weise besser funktioniert als die mit den drei bisherigen Kategorien. Dazu trainierten wir noch einmal das Netz mit zwei Kategorien und stellen die Confusion Matrix auf (siehe Tabelle 4). Bei einem Threshold von 0 erhält man fast gleiche Wahrscheinlichkeiten wie bei 3 Kategorien für Regen und Kein Regen: Falls kein Regen vorhergesagt wurde, regnet es zu 96% nicht. Falls Regen vorhergesagt wurde, regnet es zu gut 80%.

		Vorhersage	
		Kein Regen	**Regen**
Daten	**Kein Regen**	2218754 (96,3%)	45636 (18,4%)
	Regen	85427 (3,7%)	201991 (81,6%)

Tabelle 4: Confustion-Matrix (Zwei Kategorien, Threshold: 0, Softmax), die Vorhersage von keinem Regen ist genau so gut wie bei drei Kategorien, die Kategorie für Regen wird besser vorhergesagt, als die einzelnen Kategorien für wenig oder viel Regen (vgl. Tabelle 3).

Verwendet man einen Threshold von 2 (mm) für die Generierung der Kategorien, verschlechtern sich die Wahrscheinlichkeiten leicht (siehe Tabelle 5). Falls Regen vorhergesagt wurde, ist die Kein-Regen-Wahrscheinlichkeit nun 19,6%, statt 18,4%. Die Wahrscheinlichkeit von Regen, falls kein Regen vorhergesagt wurde, wird dafür minimal besser: 3,3% statt 3,7%.

		Vorhersage	
		Kein Regen	**Regen**
Daten	**Kein Regen**	2272947 (96,7%)	39276 (19,6%)
	Regen	78475 (3,3%)	161110 (80,4%)

Tabelle 5: Confustion-Matrix (Zwei Kategorien, Threshold: 2, Softmax), die Kategorie „kein Regen“ wird leicht besser vorhergesagt, Regen etwas schlechter (vgl. Tabelle 4).

Erhöht man den Threshold weiter auf 4, verdeutlicht sich der Trend aus der letzten Confusion Matrix (siehe Tabelle 6): die Wahrscheinlichkeit, dass für Regen „kein Regen“ vorhergesagt wird, sinkt auf 2,7%, die Wahrscheinlichkeit, dass Regen vorhergesagt wird und nicht eintrifft, steigt auf 23,2%.

Man sieht also, dass die Vorhersage ob Regen eintrifft oder nicht, bei drei Kategorien und zwei Kategorien (bei einem Threshold von 0) fast gleich ausfällt und die Erhöhung des Thresholds die Genauigkeit senkt. Die genaue Regenmenge vorauszusagen hat sich als schwierig herausgestellt, aber man weiß zumindest (fast) sicher, dass man, wenn kein Regen vorhergesagt wird,

		Vorhersage	
		Kein Regen	**Regen**
Daten	**Kein Regen**	2376736 (97,3%)	25441 (23,2%)
	Regen	65381 (2,7%)	84250 (77,8%)

Tabelle 6: Confustion-Matrix (Zwei Kategorien, Threshold: 4, Softmax), die Kategorie „kein Regen“ wird wieder etwas besser vorhergesagt, Regen schlechter (vgl. Tabelle 5).

nicht nass wird. Da das Datenset sehr unbalanciert ist, ist die Trefferquote nur bedingt aussagekräftig. Auch ein Klassifikator, welcher immer nur kein Regen vorhersagt hätte eine gute Trefferrate. Um dieser Eingenschaft entgegen zu wirken, wird auch ein ROC-Analyse durchgeführt.

7.3 ROC-Kurve bei zwei Kategorien

Zur Überprüfung des Verhaltens der Genauigkeit bei Anpassung des Thresholds wird eine Receiver Operating Characteristic-Kurve (ROC) verwendet (siehe Abbildung 19). Diese dient zum Vergleich der Confusion Matrizen mit verschiedenen Thresholds und damit der Performance des Netzes.

Auf der X-Achse wird die True Positive Rate (Sensitivität) abgebildet. Diese beschreibt die Rate der richtig kategorisierten positiven Ereignisse (In unserem Fall: Kein Regen vorhergesagt und es regnet tatsächlich nicht).

$$TPR \quad = \quad TruePositives \quad / \quad (TruePositive + FalseNegatives)$$

Auf der Y-Achse wird die False Positive Rate abgebildet. Diese beschreibt die Rate der falsch kategorisierten negativen Ereignisse (In unserem Fall: Kein Regen vorhergesagt aber es regnet tatsächlich) und der Gesamtanzahl der negativen Ereignisse.

$$FPR \quad = \quad FalsePositives \quad / \quad (FalsePositive + TrueNegatives)$$

In einem Optimalen Szenario wäre der Wert der TPR gleich eins und der Wert der FPR gleich Null.

7.3.1 ROC-Kurve bei verschieden Trainierten Netzen

In diesem Beispiel wird der Threshhold der Daten angepasst, ab wann ein Pixel als Regen zählt. Die daraus entstehende Confusion Matrix wird dann in der ROC verwendet. Dies soll zur Veranschaulichung dienen, welcher Schwellwert auf den Label am sinnvollsten ist.

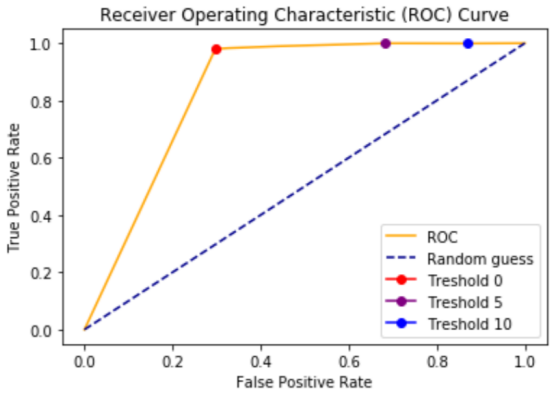


Abbildung 19: ROC-Kurve verschiedener Thresholds. Die Vorhersage wird schlechter mit steigendem Threshold.

Wie bereits vermutet, verschlechtert sich die Genauigkeit mit einer Erhöhung eines Thresholds, der mit Null daher den bestmöglichen Wert hat. Außerdem kann dadurch nachgewiesen werden, dass die Genauigkeit höher als beim zufälligen Auswählen ist.

7.3.2 ROC-Kurve bezüglich dem beste Netz

Da sich gezeigt hat, dass der Threshold 0 sich am besten eignet, wird nun die ROC auf die Ausgabe des Netzes, welches mit einem Threshold von 0 trainiert wurde, angewandt. Hierzu wird jeweils die Wahrscheinlichkeit, für die Klasse „Regen“ ausgewertet. Pixel welche als Regen klassifiziert sind haben hier ideal einen Wert nahe bei 1, regenfreie Pixel idealerweise einen Wert nahe bei 0. Wenn die beiden Klassen fehlerfrei voneinander zu trennen sind, ergibt sich daraus eine Area under the Curve (AUC) von 1. Bei einem Klassifikator, welcher die Klassen nicht trennen kann, erreicht eine AUC von 0,5. in unserem Fall (Abbildung 20) wird eine AUC von 0,955 erreicht, was auf ein recht gutes unterteilen der Klassen hinweist. Unser Klassifikator ist nicht ganz ideal, aber für eine 35 Minuten vorhersage doch recht zuverlässig.

7.4 Vergleich zu einfacher Klassifizierung mittels CNN

Ein sehr ähnlicher Test wurde auch mit einem einfachen CNN durchgeführt. Hierbei wurden die selben Eingangsdaten verwendet. Als Label wurde allerdings nur eine einfache Klassifizierung für den ’Konstanzpixel’ vorgenommen. Im Gegensatz zu dem UNet muss das CNN also lediglich für einen Pixel eine Klassifizierung vornehmen. Das Ergebnis dieses Versuchs zeigt, dass das CNN für eine vergleichsweise einfachere Aufgabe eine ähnliches bzw. leicht schlechteres Ergebnis liefert. Die Auswertung ist in Tabelle 7 abgebildet. Es zeigt sich, eine deutliche Schwäche bei der Vorhersage von Regen, hier sind

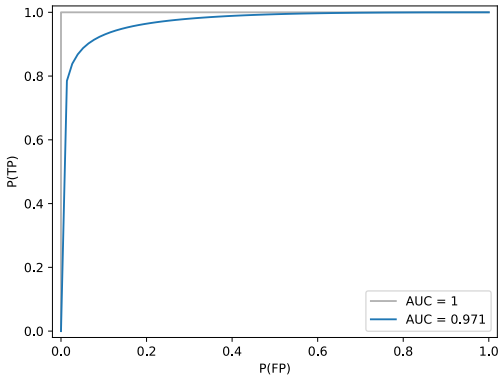


Abbildung 20: ROC-Kurve für den Threshold 0

lediglich 55% der vorhersagen korrekt. Auch bei der Klasse „kein Regen“ ist das UNet zuverlässiger.

		Vorhersage	
		Kein Regen	**Regen**
Daten	**Kein Regen**	410 (92%)	80 (45%)
	Regen	35 (8%)	98 (55%)

Tabelle 7: Confusion Matrix der Vorhersage des Konstanzpixels mit CNN. Die Sicherheit der Vorhersage von „kein Regen“ ist genau so gut wie beim UNet, die der Regenvorhersage deutlich schlechter.

7.5 Herausforderungen in diesem Kapitel

In diesem Kapitel gab es zwei große Herausforderungen: Zum Einen war es schwierig die richtigen Kategorien zu finden, und zum Anderen musste sich herausstellen, wie sich die verschiedenen Aktivierungsfunktionen auswirken.

Die drei Kategorien haben sich als suboptimal herausgestellt, da vor allem die dritte Klasse bei Weitem nicht genügend Samples für ein sinnvolles Training besitzt. Mit zwei Kategorien hingegen können wir zuverlässige Vorhersagen bezüglich Regen oder kein Regen geben.

Die Aktivierungsfunktionen haben sich nur auf die Form der Blobs der Niederschlagsmengen ausgewirkt (Abbildung 18), aber weniger auf die Regen-Wahrscheinlichkeit der einzelnen Punkte (siehe Tabellen 2 und 3).

8 Fazit

Eine kurzzeitige Regenvorhersage konnte sowohl mit Regression als auch mit Klassifizierung erreicht werden. Die Regression konnte die Baseline übertref-

fen und bis zu 35 Minuten vorhersagen.

Bei der Klassifizierung wurden die Daten in drei Kategorien eingeteilt wodurch wir gute Ergebnisse erzielen konnten, bei der Reduzierung auf zwei Kategorien konnte die Performance weiter erhöht werden. Auch hier sind zuverlässige Voraussagen bis 35 Minuten möglich.

Um die Performance weiter zu steigern könnte eine alternative Lossfunktion ausprobiert oder entwickelt werden, was aber im Rahmen des Teamprojekts nicht mehr möglich war.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[And] Andre Gensler et. al. Deep learning for solar power forecasting – anapproach using autoencoder andlstm neural networks. 2018.

[BEPW18] Klaus Backhaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber. Neuronale netze. In Klaus Backhaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber, editors, *Multivariate Analysemethoden*, pages 581–587. Springer Gabler, Berlin, 2018.

[Chr19] Christoph Fröhlich. Sonne, regen, schneefall: Warum zeigen alle wetter-apps etwas anderes an? wird es heute nochmal sonnig? muss ich für den wochenendtrip einen regenschirm einpacken? immer mehr menschen nutzen wetter-apps für die wettervorhersage. doch warum spuckt eigentlich jede andere ergebnisse aus?, 2019.

[GT19] Adrian Iustin Georgevici and Marius Terblanche. Neural networks and deep learning: a brief introduction. *Intensive care medicine*, 2019.

[HTRS19] Muhammad Hanif, Muhammad Atif Tahir, Muhammad Rafi, and Furqan Shaikh. Flood detection using social media big data streams. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*, volume 18, pages 761–771. Springer International Publishing, Cham, 2019.

[htt14] neuronale netze, 2014.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[WEB18] Andreas Welsch, Verena Eitle, and Peter Buxmann. Maschinelles lernen. *HMD Praxis der Wirtschaftsinformatik*, 55(2):366–382, 2018.

[Wic17] Christoph Wick. Deep learning. *Informatik-Spektrum*, 40(1):103–107, 2017.