HTWG Konstanz

Fakultät für Informatik

DeepRain

Regenvorhersage mit Neuronalen Netzen und die Visualisierung dieser in einer App

MSI AS - Master Informatik, Autonome Systeme

Autoren: Simon Christofzik
Paul Sutter
Till Reitlinger

Version vom: 11. September 2020

Betreuer: Prof. Dr. Oliver Dürr

Zusammenfassung

Ziel der Vorliegenden Arbeit ist zu prüfen, ob es mit begrenzten Ressourcen möglich ist eine Regenvorhersage zu berechnen und diese den Nutzern bereitzustellen. Für die Berechnung der Regenvorhersage wurden Neuronale Netze eingesetzt. Die hierfür benötigten historischen und aktuellen Radardaten wurden vom Deutschen Wetterdienst bezogen und anschließend analysiert und aufbereitet. Des Weiteren wurde eine App entwickelt, in der die Regenvorhersagen visualisiert werden. Außerdem bietet Sie die Möglichkeit, bei bevorstehenden Regen, den Benutzer zu benachrichtigen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In Quasi allen Zweigen der Wirtschaft spielt das Wetter eine Rolle welche das Handeln und den wirtschaftlichen Erfolg beeinflusst. Angefangen beim Bäcker, dessen Personalplanung und Brötchenmenge vom Grillwetter abhängt, bis zur Landwirtschaft, in welcher der Anbau und Erntezeitpunkt auf Grundlage der Wettervorhersagen getroffen werden muss. Außerdem finden Wettervorhersagen selbstverständlich auch im privaten Sektor große Anwendung und werden von den meisten Menschen täglich verwendet. Während die Vorhersage von Temperaturen und Sonnenstunden mit Physikalischen Modellen inzwischen sehr zuverlässig funktionieren, sind Regenvorhersagen verhältnismäßig schwer zu machen. Nicht umsonst werden Regenvorhersagen in den meisten Fällen nur mit Wahrscheinlichkeiten angegeben. Das liegt vor allem an der immens großen Anzahl von Faktoren welche den Regen beeinflussen. Laut dem Deutschen Wetterdienst sind aktuell nur 80% der Regenvorhersagen korrekt, bei den Temperaturen hingegen sind es über 90% [Mingels, 2016]. Der Rechenaufwand für Wettervorhersagen ist Enorm aber aufgrund der beschriebenen Relevanz werden die nötigen Ressourcen eingesetzt um ausreichend gute Wettervorhersagen zu ermöglichen.
In Deutschland gibt es mit dem Deutschen Wetterdienst eine Behörde die meteorologischen Dienstleistungen für die Allgemeinheit erbringt. Diese werden im Folgenden auch in Anspruch genommen. Neuronale Netzte können dabei helfen in diesen komplexen und teilweise chaotisch wirkenden Vorgängen Muster zu erkennen und so Regenvorhersagen zu berechnen.

2 Gesamtsystem

In folgender Abbildung ist die Datenpipeline von dem Server des DWD bis zur Darstellung der Vorhersagen in der App zu sehen.
Der DWD stellt alle fünf Minuten die neusten Regendaten für Deutschland im Binär Format auf dem Opendata Server des DWD zur Verfügung. Diese werden heruntergeladen und in der Datenaufbereitung verwendet, um PNGs zu generieren (Siehe Kapitel 4). Dieses PNGs werden im Anschluss verwendet, um eine Regenvorhersage zu machen (Siehe Kapitel 5). Der Output der Regenvorhersage ist wieder ein PNG. Dieses PNG wird in der Firebase gespeichert und auf allen Geräten mit der installierten App angezeigt (Siehe Kapitel 6). Der Übersichtlichkeit halber haben wir das gesamte Projekt in die drei Komponenten “Datenbeschaffung & Vorverarbeitung”, “Vorhersage” und “App & Datenbankhandling” aufgeteilt. Die Komponente “Datenbeschaffung & Vorverarbeitung” reicht vom Download der Binärdaten beim DWD bis zu den daraus generierten PNG’s. In der Komponente “Vorhersage” werden die Regenvorhersage mithilfe von neuronalen Netzen gemacht. In der Komponente “App & Datenbankhandling” Geht es um das Datenmanagement

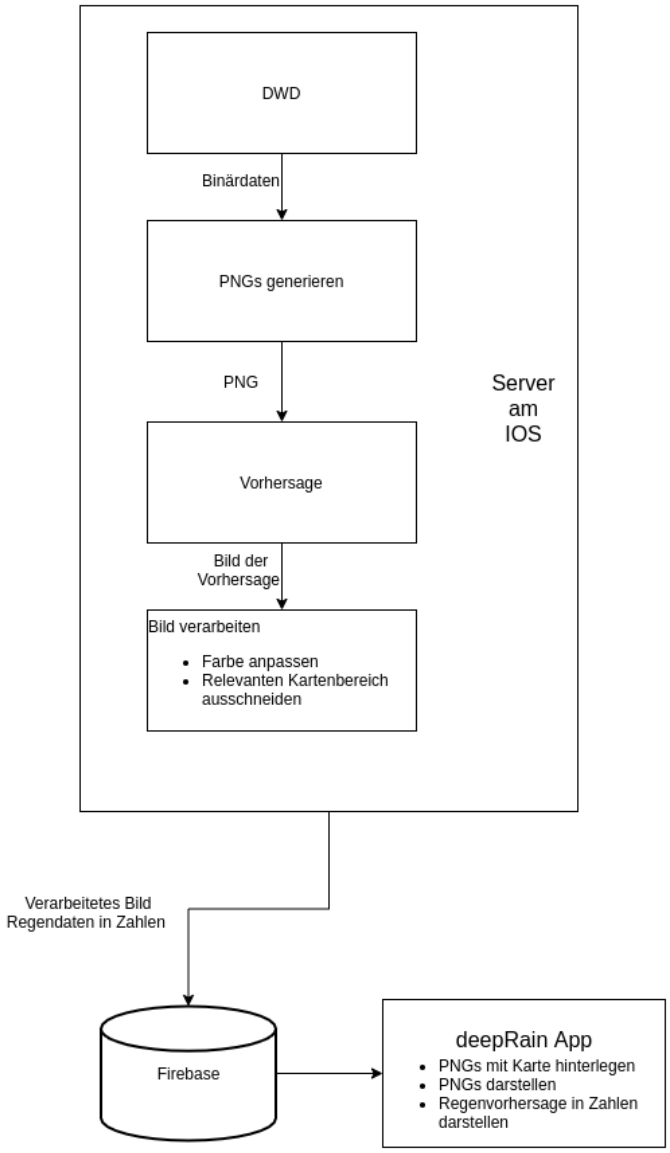


Abbildung 1: Das Gesamtsystem

mit der Firebase, und um die App, welche die Daten darstellt.

3 Die Daten

Die Datengrundlage für die Netze werden von dem DWD in Form des Radar online Aneichungs verfahren (RADOLAN) zur Verfügung gestellt. Das RADOLAN verfahren kombiniert die Messungen der 18 Radarstation mit den punktuellen Messungen von über 2000 Bodenniederschlagsstationen [Wetterdienst, 2020]. Eine dieser Stationen befindet sich in Konstanz.
Die RADOLAN Daten für die Netze werden über den Opendata Server vom DWD zur Verfügung gestellt. Die Binärdaten werden je nach Jahr in Form eines 1100x900 oder 900x900 Pixel Gitter über Deutschland gelegt. Dabei entspricht jeder Pixel 1km x 1km. Jeder Pixel in dem Pixel Koordinatensystem hat dabei zugehörige Höhen

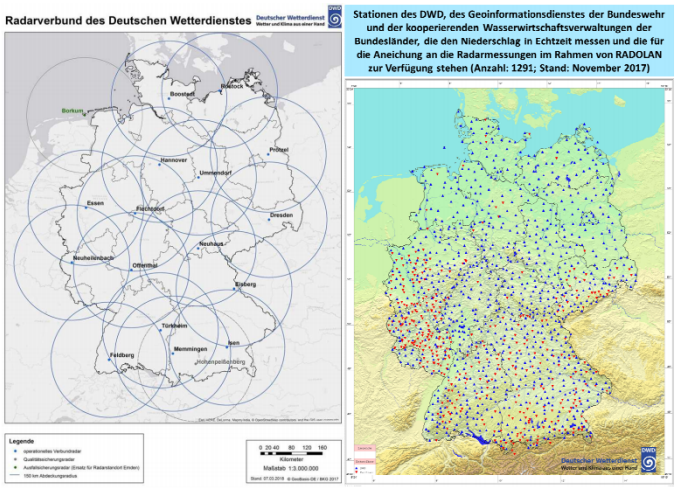


Abbildung 2: Übersicht über alle Boden und Radarstationen [DWD, 2019]

und Breitengrad Koordinaten. In Abbildung 3 ist der Aufbau der Binärdaten zu sehen.

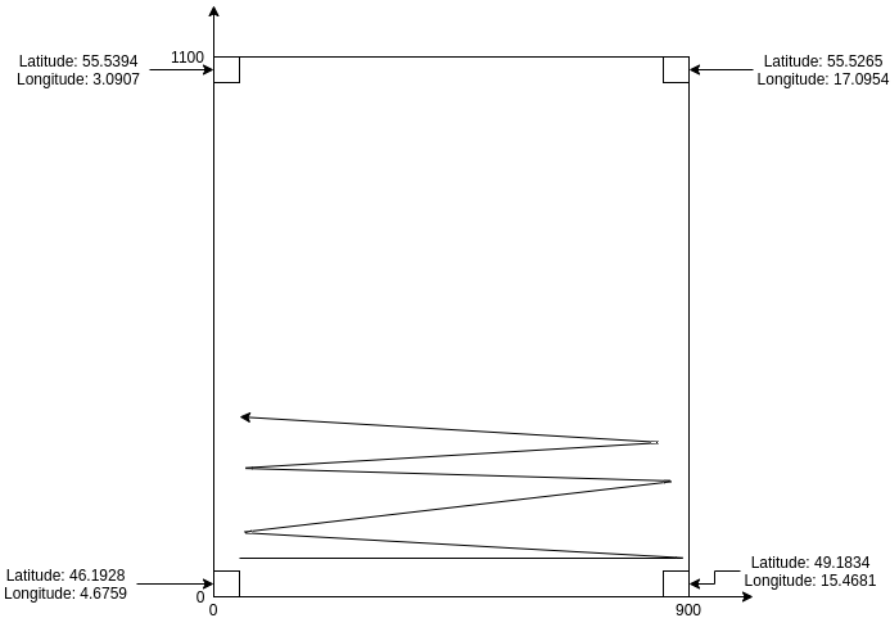


Abbildung 3: Der Aufbau des Koordinatensystems der Radar Daten

Wie zu sehen ist, befinden sich der Pixel (0|0) in der Ecke unten links. Bei der Datenvorverarbeitung wird das Array mit den Pixeln jedoch von oben Links beginnend gefüllt, was bei dem entstehenden PNG zu einer Spiegelung von 180 grad um die vertikale Achse führt (Siehe Kapitel Datenaufbereitung). Des Weiteren ist zu beachten, dass die Breitengrade in den Ecken nicht übereinstimmen. So hat die Ecke unten Links einen größeren Breitengrad als die Ecke oben Links. Das Gleiche ist auch bei den Höhengraden zu beobachten. Die Ursache hierfür ist die Transformation der Daten von einer 3D Kugel auf eine 2D Karte. In Abbildung 4 ist das daraus resultierende Ergebnis abgebildet.

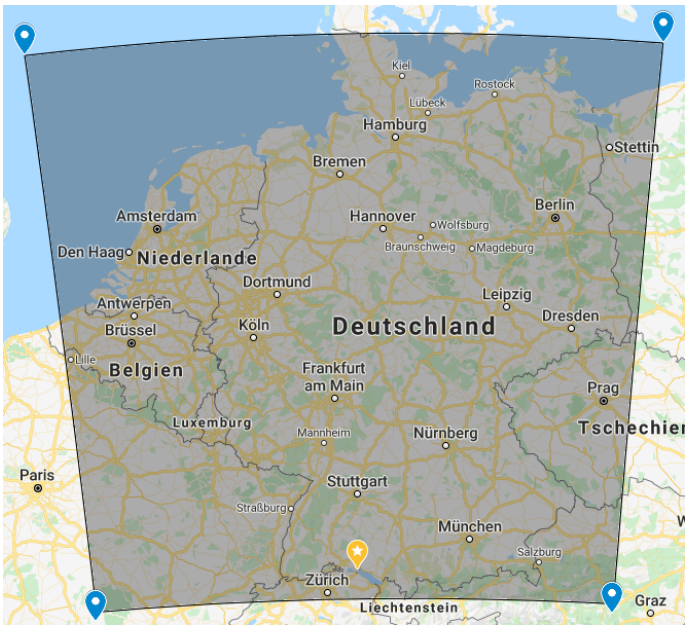


Abbildung 4: Die von den Daten abgedeckte Fläche auf einer 2D Karte

3.1 Die Qualität der Daten

Um die Qualität der aus den RADAR Daten gewonnenen PNGs zu überprüfen, wurden die PNGs mit den Niederschlagsdaten von der Bodenstation in Konstanz verglichen. Die Daten der Station stehen in ein und zehnminütiger Auflösung zur Verfügung. Da bei der einminütigen Auflösung Daten Fehlen, wurden die Daten der zehnminütigen Auflösung verwendet. Dafür musste aus den RADAR Daten jeder zweite Datensatz entfernt werden. Im ersten Schritt sollte die bereits berechnete Position von Konstanz, in dem Vorhersage PNG, bestätigt werden. Dazu wurde die Korrelation zwischen der Regenstation in Konstanz und allen Pixel berechnet. Diese Korrelationen sind in Abbildung 5 grafisch dargestellt. Dabei stellen hellere Farben eine höhere Korrelation dar.

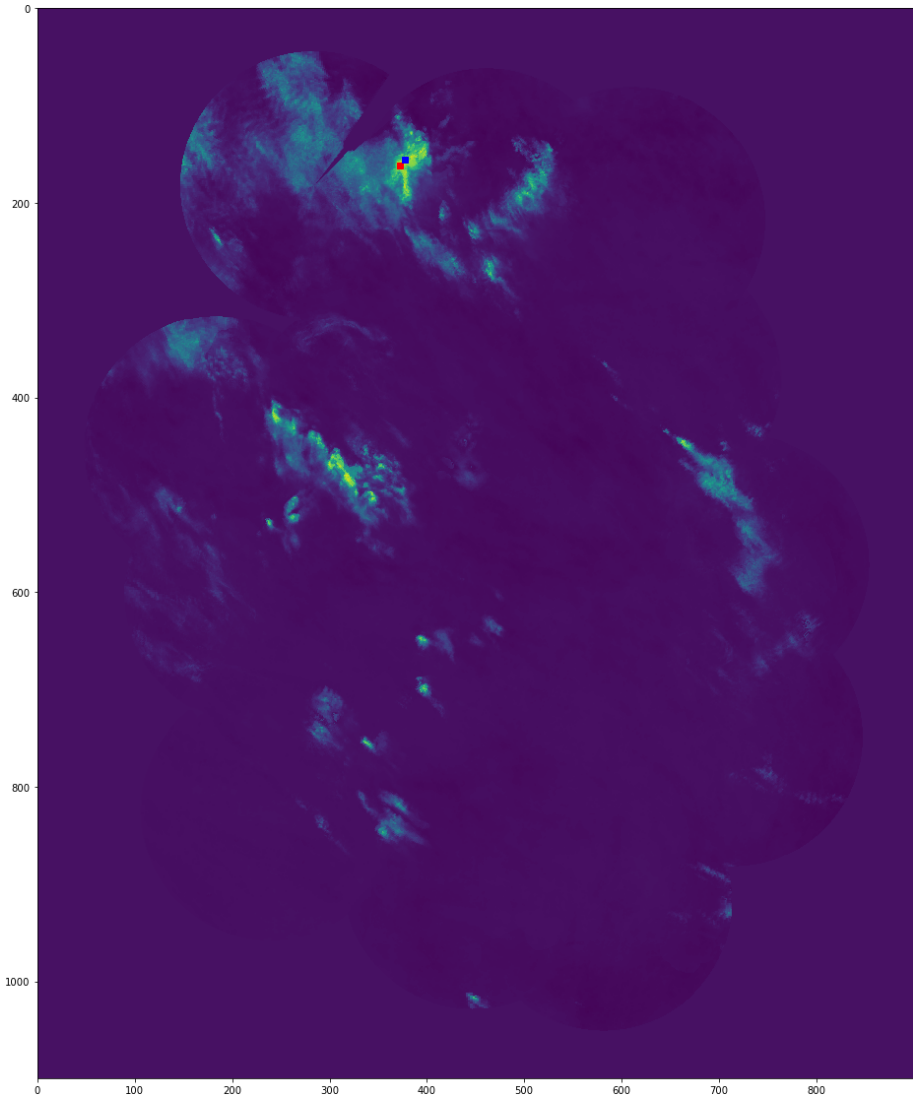


Abbildung 5: Korrelation der Regendaten in Konstanz mit jedem Pixel

Das rote Quadrat stellt den Pixel mit der größten Korrelation dar, das blaue Quadrat den bisher berechneten Pixel von Konstanz. Die Abweichung dieser beiden Pixel voneinander lässt sich durch die kleine, für die Korrelation verwendete Datenmenge erklären. Wie man an der Position von Konstanz sieht, ist das Vorhersage PNG um 180 Grad an der vertikalen Achse gespiegelt.
Um sicherzugehen, dass die für die Vorhersage verwendeten Daten mit dem tatsächlichen Niederschlag in Konstanz übereinstimmen, wurden sie grafisch dargestellt. In Abbildung 6 ist der Niederschlag aus den RADAR Daten und der Bodenstation zu sehen. Es ist zu erkennen, dass die Regenstärke tendenziell leicht abweicht, während der Zeitpunkt des Regens sehr gut übereinstimmt. Die Abweichung der Intensität ist systematisch und sollte somit kein Problem darstellen.

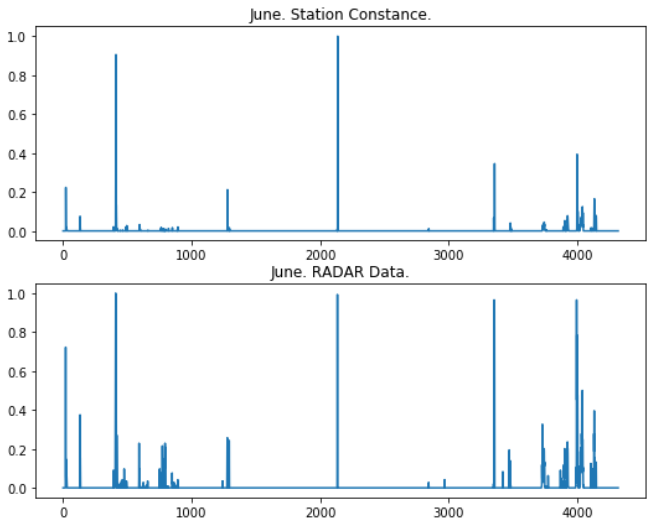


Abbildung 6: Die Radar und Stationsdaten für Juni im Vergleich

In folgender Abbildung ist zu erkennen, dass die Regenstation in Konstanz im Juni 214 Zeitschritte mit Regen aufgezeichnet hat, es laut Radardaten aber 258 Zeitschritte mit Regen gab. In 68% der Regenzeitpunkte stimmen die Radar und Stationsdaten überein. Der fehlende Regen in den Stationsdaten könnte sich durch Messungenauigkeiten oder eine etwas abweichende geografische Lage von Konstanz erklären lassen.

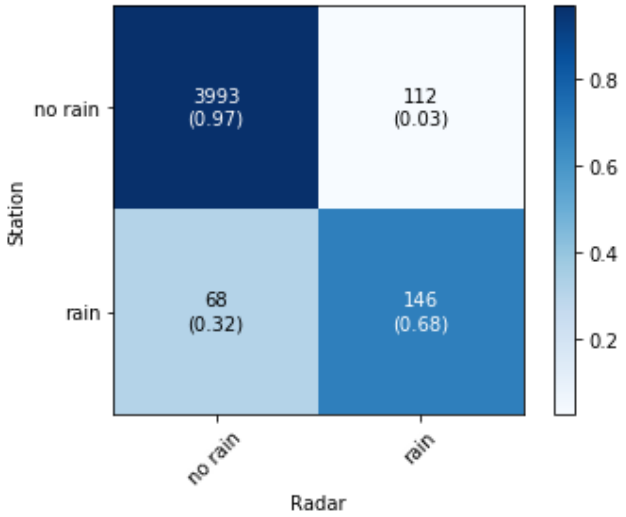


Abbildung 7: Confusionmatrix für die Radar und Stationsdaten im Juni

4 Die Datenaufbereitung

Die vom Deutschen Wetterdienst (im Folgenden DWD abgekürzt) in fünf minütiger Auflösung bereitgestellten Radardaten, müssen für das Training der Netze und deren Vorhersage in ein Bildformat umgewandelt werden. Bei den bereits umgewandelten

Bildern viel während der Entwicklung der Baseline auf, dass die Bilder weit weniger Regentage abbilden als es tatsächlich regnet. Daraufhin wurde das bisher vorgenommene Preprocessing evaluiert. Um die Radardaten in ein Bild umzuwandeln, muss jeder Radardatenpunkt in ein Pixelfarbwert transformiert werden. Bisher wurde dafür ein Skalierungsfaktor berechnet mit dem jeder Radardatenpunkt multipliziert wurde. Der Faktor ergab sich aus dem zur Verfügung stehenden Wertebereich (0 bis 255), welcher durch den Maximalwert der Radardaten geteilt wurde. So erhält man transformierte Radarwerte in einem Bereich von 0 bis 255. Anschließend folgte eine Inspizierung der Daten. Hierfür wurde exemplarisch die Radardaten von Juni 2016 herangezogen.

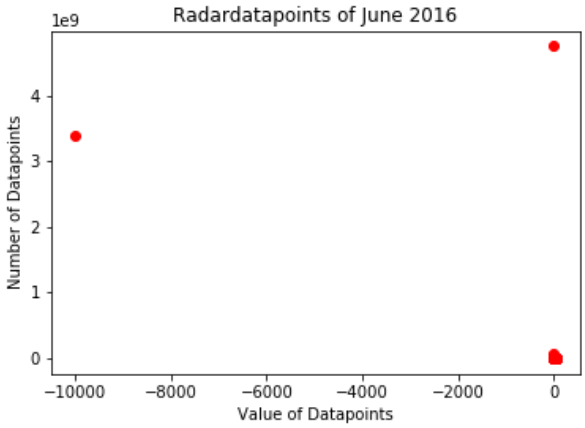


Abbildung 8: Häufigkeitsverteilung der Regendaten

Geplottet werden alle auftretenden Werte sowie dessen Häufigkeit. Hierbei stechen in Abbildung 8 zwei Ausreißer hervor, welche sehr viel häufiger vorkommen als die restlichen Werte. Der Wert –9999 steht dabei dafür, dass keine Daten verfügbar sind und der zweite Ausreißer ist bei 0, was für “kein Regen” steht. In Abbildung 9 werden die beiden Ausreißer gefiltert, da die relevanten Informationen in den restlichen Datenpunkten stecken.

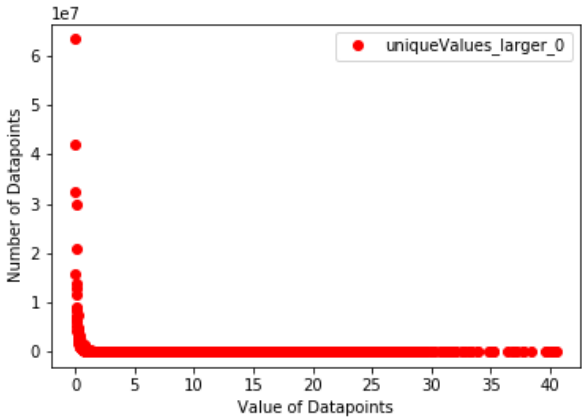


Abbildung 9: Häufigkeitsverteilung der Regendaten größer null

In Abbildung 9 dargestellt sind die Radardaten bei denen es regnet. Es wird deutlich, dass ein Großteil der Datenpunkte klein ist. Der Mittelwert liegt bei 0,1744 und zeigt das Problem der bestehenden Preprocessing Methode: Radardatenpunkte deren Wert auch nach der Multiplikation mit dem berechneten Skalierungsfaktor kleiner eins sind werden zu null. Aufgrund der vorliegenden Datenverteilung führt das zu einem erheblichen Fehler weshalb eine andere Methode entwickelt werden muss. Abbildung 10 und 11 zeigen verschiedene Perzentile und machen so die Häufigkeitsverteilung deutlich.

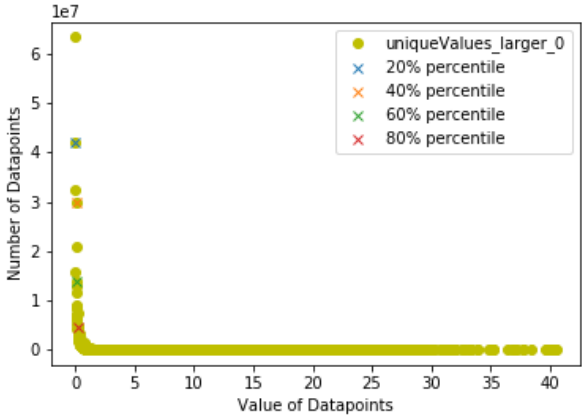


Abbildung 10: Häufigkeitsverteilung der Regendaten mit Perzentilen

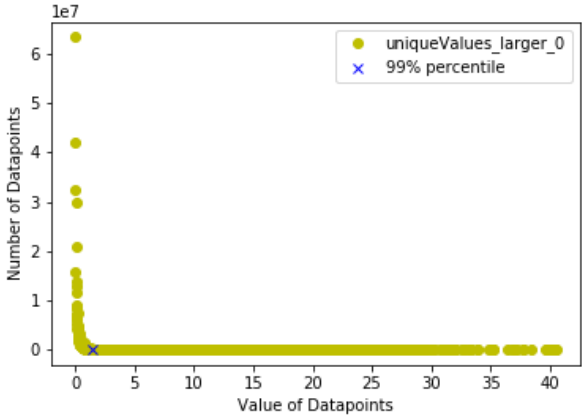


Abbildung 11: Häufigkeitsverteilung der Regendaten mit 99% Perzentil

Das 99 Prozent-Perzentil, dargestellt in Abbildung 11, beinhaltet 138 verschiedene Werte. Wenn jedem dieser Werte ein Farbwert zugeordnet wird, werden 99 Prozent der daten bereits abgebildet und es verbleit ein Wertebereich von 117 mit welchem das letzte Prozent der Daten dargestellt werden kann. Abbildung 12 zeigt die Verteilung der noch verbleibenden Daten.

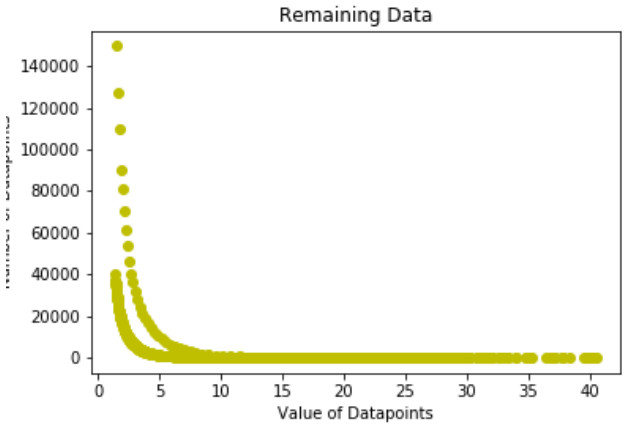


Abbildung 12: Regendaten außerhalb des 99% Perzentils

Noch immer befindet sich der größte Teil der Datenpunkte im kleinem Wertebereich. Da für die verbleibenden Daten eine lineare Transformation ähnlich dem bereits bestehenden Preprocessing Vorgang eingesetzt wird, entstehen Rundungsfehler. Für die Berechnung des Skalierungsfaktors wird nicht der tatsächliche Maximalwert genutzt, weshalb Werte die nach der Transformation über 255 sind auf 255 gesetzt werden. Diese Fehler sind verkraftbar, da es zum einen wichtiger ist alle Datenpunkte abzubilden und zum anderen die Auflösung der verschiedenen Regenstärken immer noch höher ist, als die der menschlichen Wahrnehmung. In Abbildung 9 werden die Radardaten dargestellt welche im folgenden Transformiert werden. Die transformierten Daten befinden sich in einem Wertebereich von 0 bis 255, dargestellt in Abbildung 13.

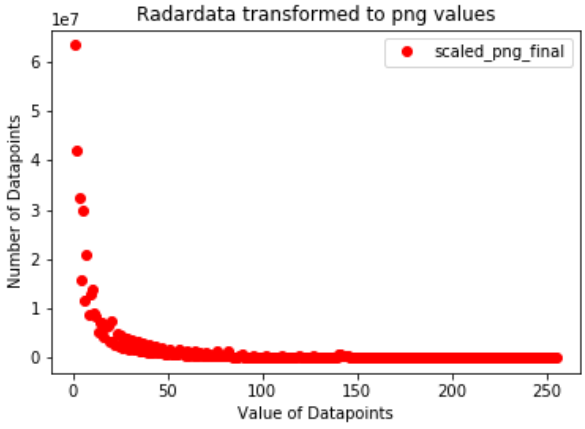


Abbildung 13: Häufigkeitsverteilung der Regendaten größer null nach der Transformation

Rekonstruiert man aus den PNG Daten wieder die Radardaten ergibt sich der in Abbildung 14 dargestellte Plot. Die quantitativ wiederhergestellte Anzahl der Datenpunkte beträgt 100%. Von den beschriebenen Fehlern macht sich im Plot vor allem letzterer im Plot bemerkbar. Werte größer als der für diesen Plot angenommenem

Maximalwert von 21,39 wurden bei der Transformation auf 255 gesetzt und lassen sich daher nicht mehr rekonstruieren und erhalten deshalb den Wert 21,39.

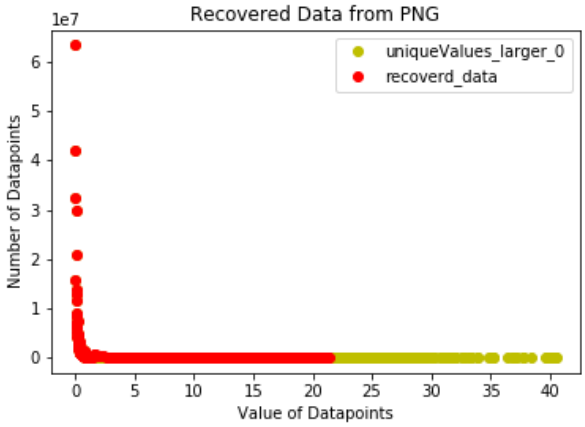


Abbildung 14: Vergleich zwischen Ausgangsverteilung und Verteilung aus wiederhergestellten Daten

5 Die neuronalen Netze

Die Regenvorhersage mit Hilfe von künstlichen neuronalen Netzen (KNN) möchten wir mit einem Zitat einführen. Der Nobelpreisträge Richard P. Feynman ist in einem Gespräch mit einem nicht namentlich erwähnten Laien wobei es um die Existenz fliegender Untertassen geht. Feynman trifft die Aussage, dass es sehr unwahrscheinlich ist, dass es fliegende Untertassen gibt. Der Laie antwortet darauf, dass das sehr unwissenschaftlich sei, worauf Feynman erwidert:

> "...But that is the way that is scientific. It is scientific only to say what is more likely and what less likely, and not to be proving all the time the possible and impossible."
>
> (Richard P. Feynman)

Treffend wird in dem Gespräch von Feynman erläutert, dass der wissenschaftliche Weg eine Wahrscheinlichkeit beinhaltet, die Auskunft über das Eintreten eines Ereignisses gibt. Wir werden für die Regenvorhersage also eine probabilistische Aussage treffen. Dies bedeutet insbesondere, dass eine Verteilung für die Regenvorhersage geschätzt wird, durch die eine Aussage über die Regenwahrscheinlichkeit und auch die Wahrscheinlichkeit der Intensität getroffen werden kann.

Bei der Regenvorhersage via KNN bekommt das KNN als Input ein Set von Bildern, woraus ein Feld von Parametern geschätzt wird, wodurch in Kombination mit der zugehörigen Verteilung eine 30-minütige Vorhersage generiert werden kann. Dieser Vorgang ist in der nachfolgenden Abbildung skizziert.

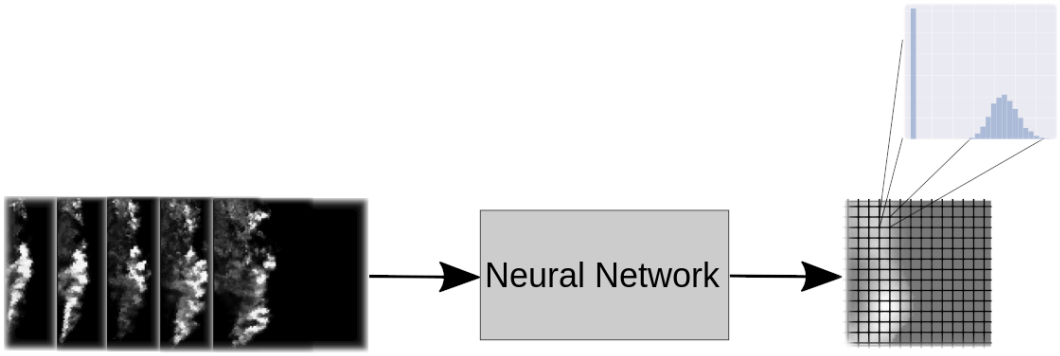


Abbildung 15: Skizzierung der Vorhersage

Wie aus Abbildung 15 hervorgeht, besteht der Ausgang des KNN aus einem Feld von Verteilungen. Diese Verteilungen nehmen wir als unabhängig an. Die zu schätzende Verteilung ist im vornherein nicht klar, weshalb wir eingangs drei Verteilungen begutachten. Wir schauen uns die Multinomiale, und die negativ Binomial Verteilung an. Die negativ Binomialverteilung wird allerdings mit der Multinomialen Verteilung

gemischt, sodass wir die ”Zero Inflated negativ Binomial” Verteilung (ZINB) erhalten. Dies kann so aufgefasst werden, dass wir eine Bernoulli-Verteilung für die binäre Entscheidung über Regen oder kein Regen erhalten. Für den Fall, dass Regen vorhergesagt wird, greifen wir auf die Negativ Binomialverteilung zurück.

5.1 Netzwerk Topologie

Wir schauen uns zwei verschiedene Netzwerarchitekturen an. Hierbei passen wir uns den Beschränkungen der uns zur Verfügung stehenden Hardware an. Uns steht eine Geforce 2060 Super mit 8GB VRAM zur Verfügung. Die Netzwerktopologie ist auf die Größe des VRAMS beschränkt, wir werden unser Netzwerk also auf diese Größe beschränken. Die Größe der Eingangsbilder setzen wir auf 96x96 Pixel fest und die Größe der Ausgangsbilder auf 64x64. Die Patches werden um den Pixel, der Konstanz repräsentiert ausgeschnitten. Alle Netzwerke die wir trainieren haben die Klassifikationsschicht gemeinsam, unterscheiden sich jedoch für die verschiedenen Verteilungen.

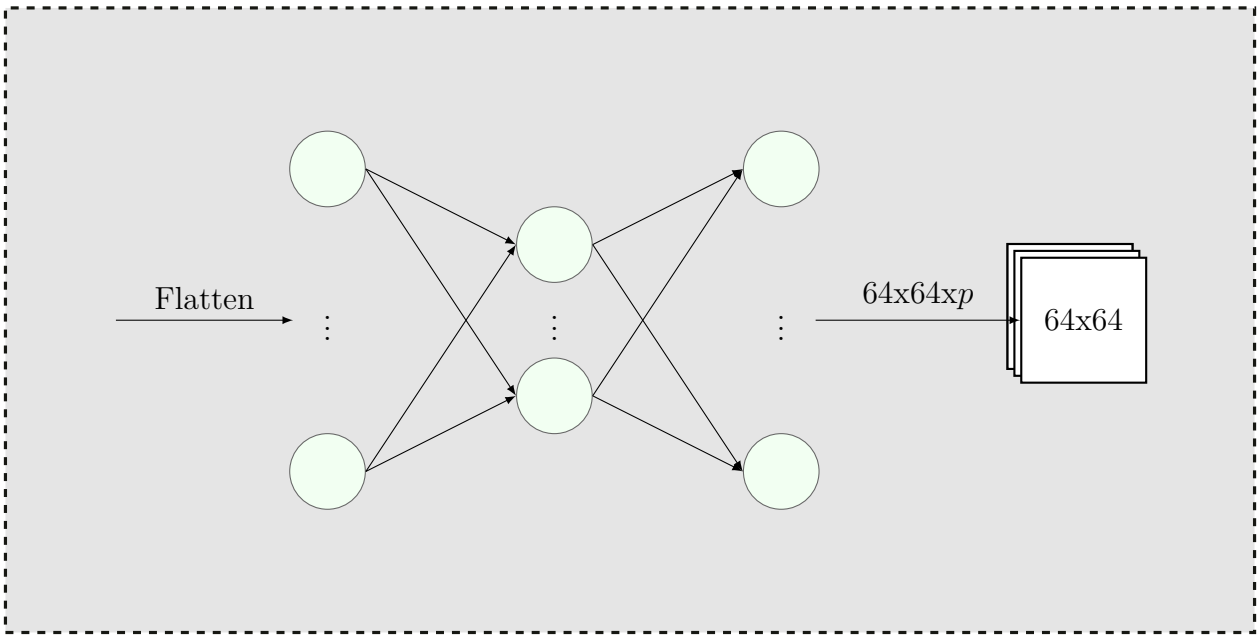


Abbildung 16: Klassifikationsschicht, p variiert hierbei je nach Verteilung. Für die Zero-Inflated Negativ Binomial Verteilung ist p gleich 3 und für die Multinomiale Verteilung ist p gleich 7

Wir verwenden zum einen die U-netarchitektur, wie sie von unseren Vorgängern verwendet wird. Diese Architektur ist in der folgenden Abbildung zu sehen.

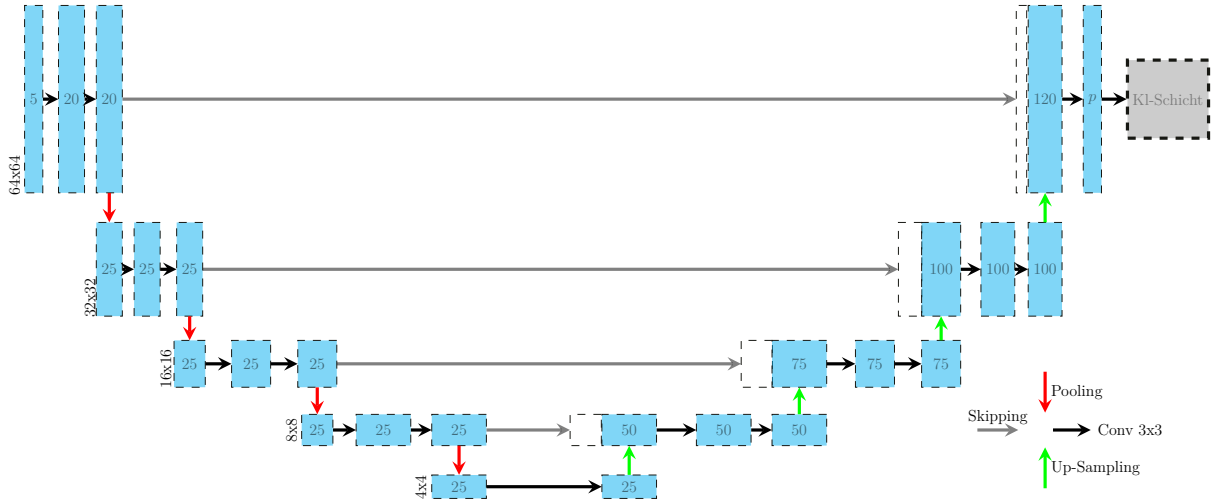


Abbildung 17: Abgespeckte Version des U-nets

Die abgespekte Version des U-nets hat im Gegensatz zur originalen Architektur [Ronneberger et al., 2015] wesentlich weniger Gewichte. Hier sind es ca 10 000 trainierbare Gewichte. Zusätzliche Gewichte liegen in der Klassifikationsschicht (Kl-Schicht in Bild 16).

Eine weitere Architektur die wir begutachten ist ein Mix aus Inception-Layer und CNN-LSTM-Layer. Die Regenvorhersage in unserem Setup wird für gewöhnlich auch als Next-Frame prediction bezeichnet. Aus einem Set aufeinanderfolgender Eingangsbilder wird eine plausible Vorhersage für das nächste Bild erzeugt.

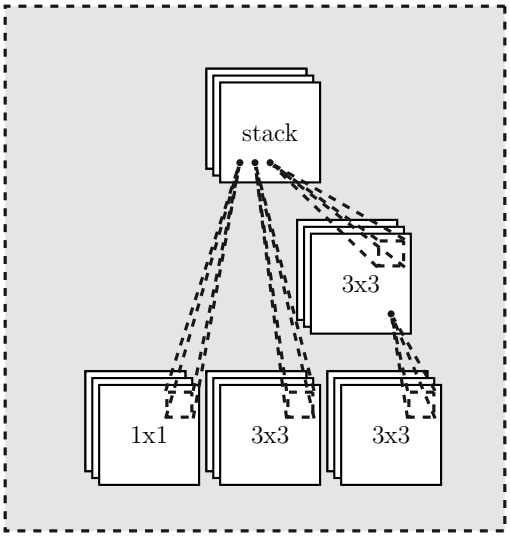


Abbildung 18: Aufbau der von uns verwendeten Inception Layer, angelehnt an die inception Layer im Paper

In der nachfolgenden Abbildung ist die von uns verwendete Architektur zu sehen. Diese Architektur ist angelehnt an die Inception-LSTM Layer des Papers "Inception-inspired LSTM for Next-frame Video Prediction" von [Hosseini et al., 2019]. Wie Eingangs erwähnt beschränkt die Hardware (und auch die Größe der Bilder) unsere Architektur und aus diesem Grund werden nicht mehr LSTM-Layer gestackt, wie im Paper beschrieben.

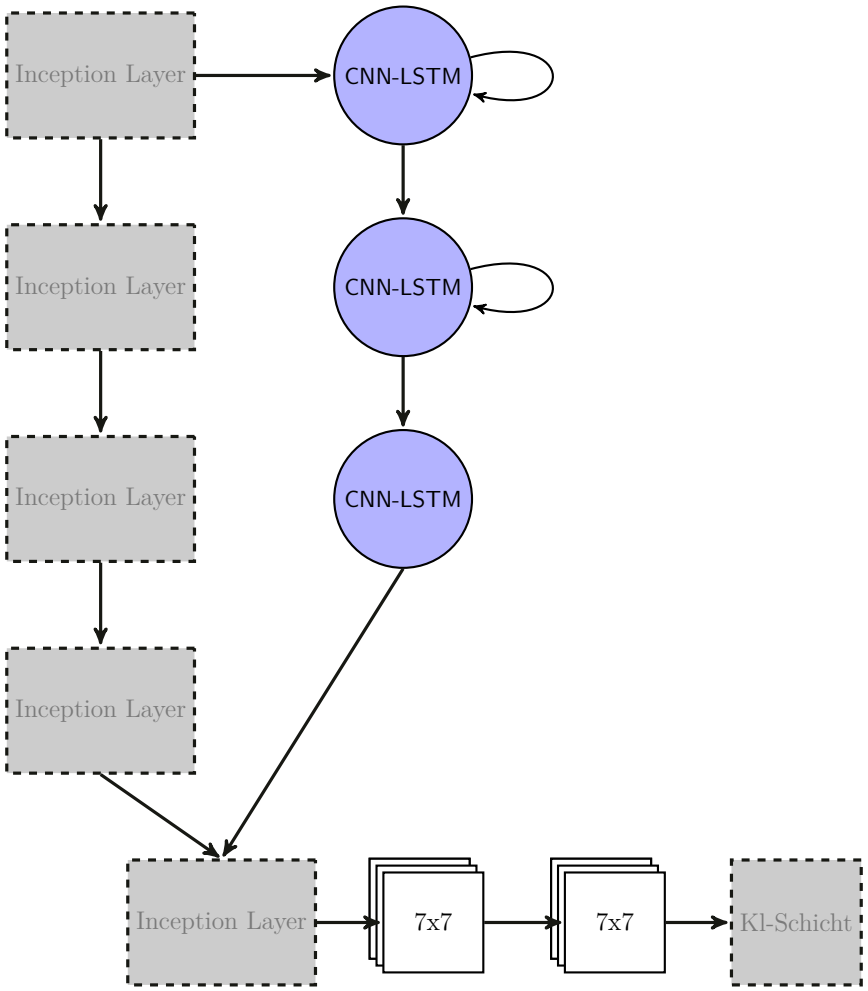


Abbildung 19: LSTM Architektur

Unsere Architektur unterscheidet sich jedoch von der im Paper vorgestellten Architektur insofern, als dass die Inception-Layer nicht in das LSTM-Modul eingebaut sind. Wir verwenden hierbei herkömmliche Convolution-LSTM Layer. Die Inception-Layer sind hierbei lediglich vor oder nach den Convolution-LSTM Layern zu finden. Der Vorteil von Inception-Layer ist, dass diese in die Breite und nicht so sehr in die Tiefe gehen. Das hat zum Vorteil, dass beim aktualisieren der Gewichte der Gradient nicht so weit in das Netzwerk durchgereicht werden muss. Dadurch soll das Auftreten des vanishing Gradients vermindert werden und das aktualisieren der Gewichte bzw. das Lernen verbessert werden.

5.2 Training

Beide Architekturen werden mindestens 35 Epochen trainiert, wobei die Architektur mit den LSTM-Layern 1:30 Stunden benötigt, das U-net hingegen benötigt ca. 10 Minuten pro Epoche. Für das Training nutzen wir alle Daten der Jahre 2008 bis einschließlich 2017. Das entspricht insgesamt ca. 1 051 200 Zeitschritten. Hierbei werden 75% der Daten für das Training und 25% der Daten für das Testset verwendet. Da der Hauptteil der Regendaten Null ist, also kein Regen vorhanden ist, liegt der Mittelwert der Daten schon nahe Null. Das bedeutet, dass der Mittelwert nicht (wie üblich) auf Null normiert wird. Die Standardabweichung ist ebenfalls nahe Null, weshalb die Standardabweichung der Daten nicht auf Eins normieren (Dies würde zur Folge haben, dass Werte wesentlich größer als 1 sein können). Die Eingangsdaten werden allerdings auf den Bereich zwischen Null und Eins normiert. Als Regularisierungsmaßnahme verschieben wir die Trainingsbilder nach jeder Epoche um ein paar Pixel in x- und/oder y-Richtung.

Für den Ausgang der Netzwerke beschränkten wir uns auf einen 64x64 großen Pixelbereich, bei dem Konstanz in der Mitte liegt. Die Eingangsbilder bestehen aus einem 96x96 großen Pixelbereich um Konstanz. Die Ausschnitte die wir verwenden sind lediglich um Konstanz herum lokalisiert. Die anderen Teile der Regenkarte nutzen wir nicht, da wir davon ausgehen dass die Eigenschaften der Konstanzer Landschaft einen direkten Einfluss auf das Wetter haben. Regenfreie Bilder werden aussortiert, da diese Daten keinerlei Informationen beinhalten und das vorhandene Klassenungleichgewicht verstärken. Als Regularisierungsmaßnahme werden die Trainingsdaten pro Epoche um wenige Pixel verschoben, was zur Folge hat, dass das Netzwerk in jeder Epoche auf etwas unterschiedliche Daten trainiert. Als Kostenfunktion verwenden wir die Negative Loglikelihood.

In der Nachfolgenden Abbildung sind die Trainingskurve für die beiden Architekturen zu sehen. Die hierfür Verwendete Verteilung ist die ZINB.
Zu sehen ist, dass die U-netarchitektur schlechtere Performance als die LSTM-Architektur liefert. In dieser Abbildung scheint der Overfitting Bereich noch nicht erreicht worden zu sein. Tatsächlich verbessern sich beide Architekturen noch marginal nach weiteren Epochen, aus Darstellungsgründen wurde die X-Achse auf 35 Epochen beschnitten.

Zusätzlich zur ZINB wurden beide Architekturen mit einer weiteren Verteilung trainiert. Hierfür verwenden wir die Multinomiale Verteilung, wobei wir die Daten in Sieben Klassen einteilen. Die geschieht durch logarithmisches skalieren der Daten. Dadurch soll zusätzlich dem Klassenungleichgewicht entgegengesteuert werden (Regenwerte

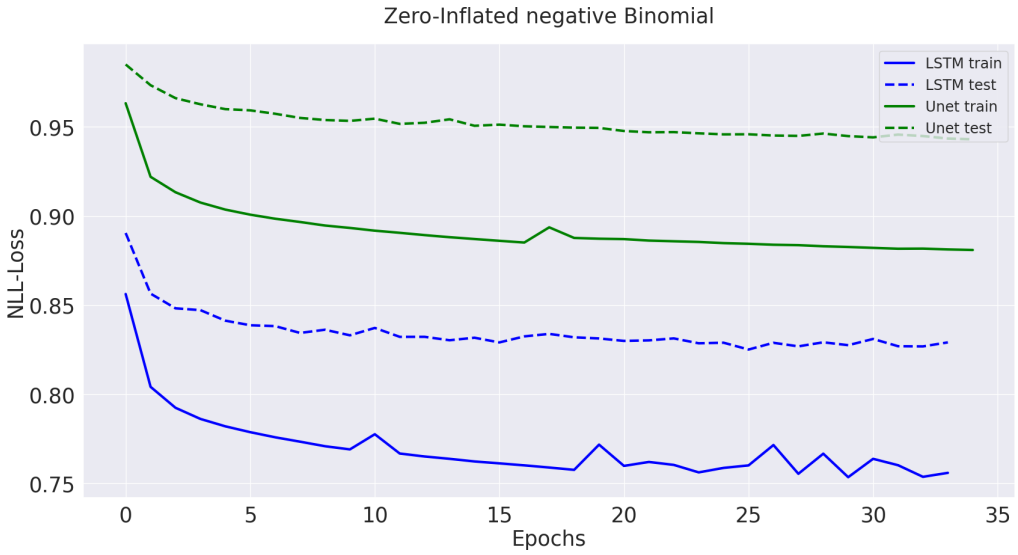


Abbildung 20: Trainingskurven für ZINB

im höheren Bereich kommen seltener vor).

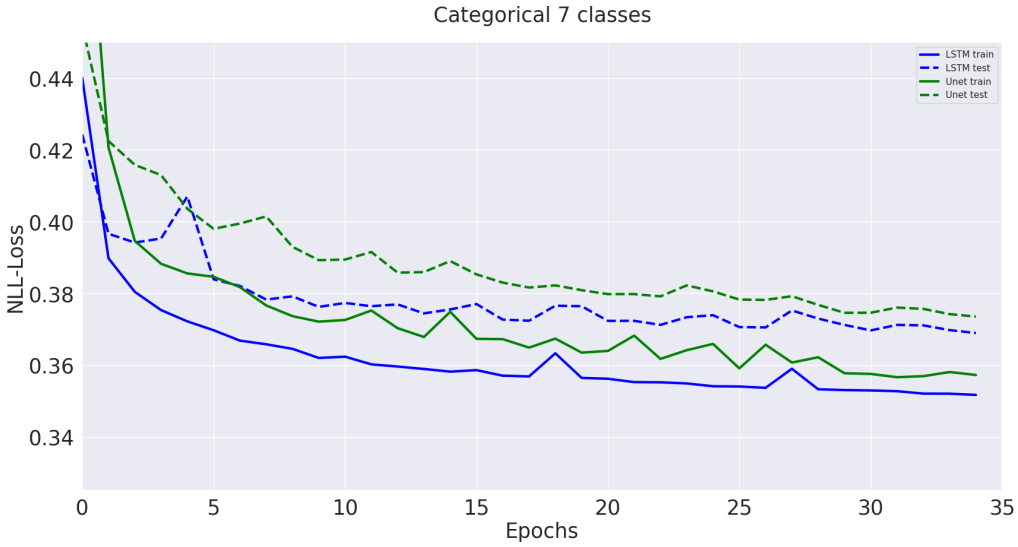


Abbildung 21: Multinomiale Verteilung

In der obigen Abbildung sind die Trainingskurven der beiden Architekturen zu sehen. Auch hier ist zu sehen, dass die LSTM-Architektur der U-netarchitektur überlegen ist und dies eine bessere Performance liefert. Vergleicht man die Trainingskurven für beide Verteilungen sieht man, dass der NLL für die Multinomiale Verteilung etwa die Hälfte der ZINB entspricht.

5.3 Auswertung

Wir beschränken uns zuerst auf die Auswertung der binären Klassifikation von Regen und kein Regen. Später werden wir für die verschiedenen Architekturen und Verteilungen die Klassifikation der Intensität begutachten.

5.3.1 Auswertung der binären Regenvorhersage

Die folgende Abbildung zeigt einen zufälligen Ausschnitt der Regenvorhersage für sieben aufeinanderfolgende Zeitschritte.

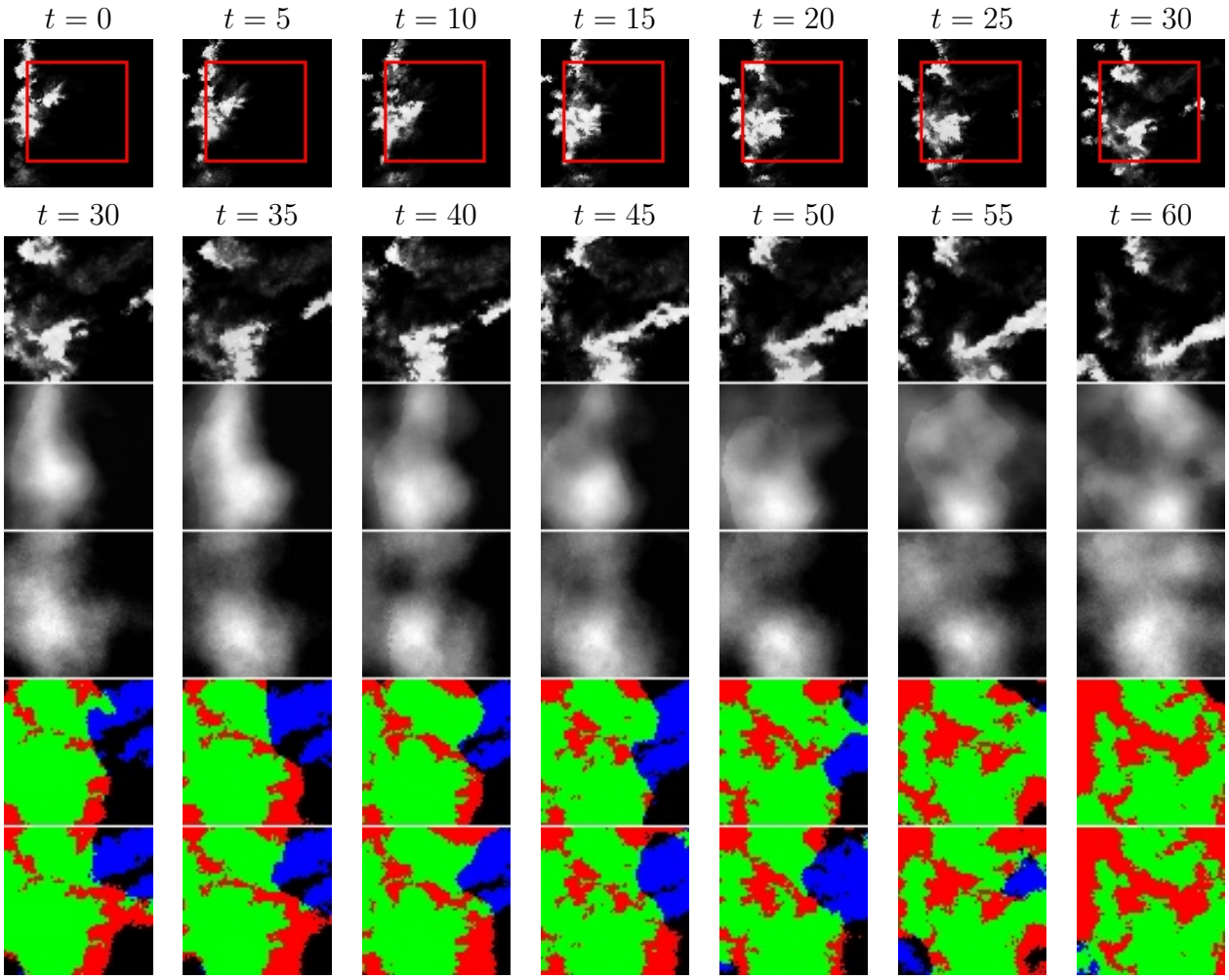


Abbildung 22: Vorhersage in Abständen von fünf Minuten für die ZINB. Die erste Zeile entspricht dem momentanen Regen. Der Rotes Quadrate ist der Bereich der Vorhersage. Zeile zwei entspricht dem tatsächlichen Regen in 30 Minuten. Zeile drei ist die 30 Minuten Vorhersage des U-nets. Zeile vier ist die 30 Minuten Vorhersage der LSTM-Architektur. Für die Vorhersage wurde der Mittelwert der Verteilung berechnet. Die Bilder sind kontrastmaximiert dargestellt. Die letzten beiden Zeilen stellen die korrekte bzw. inkorrekte Regenvorhersage für die U-net- und die LSTM-Architektur (letzte Zeile) dar. Hierbei entspricht true positiv der Farbe Grün, false positiv Rot, false negativ Blau und true negativ Schwarz .

Anhand der Abbildung 22 ist zu sehen, dass für beide Architekturen tendenziell zu viel Regen vorhergesagt wird. Anhand der Bilder ist zudem zu sehen, dass Form der Regenfront nicht korrekt vorhergesagt wird. Des weiteren ist zu erkennen, dass false negativ (Blau) einen geringeren Teil der Vorhersage ausmachen. Durch die

Kontrastmaximierte Darstellung der Vorhersage, sehen die Bilder so aus, als würden die maximalen Werte der Vorhersage in etwa der maximalen Werte des tatsächlichen Wetters entsprechen. Tatsächlich wird im allgemeinen die Regenmenge unterschätzt. Dies wird in einem späteren Teil der Auswertung dargelegt.

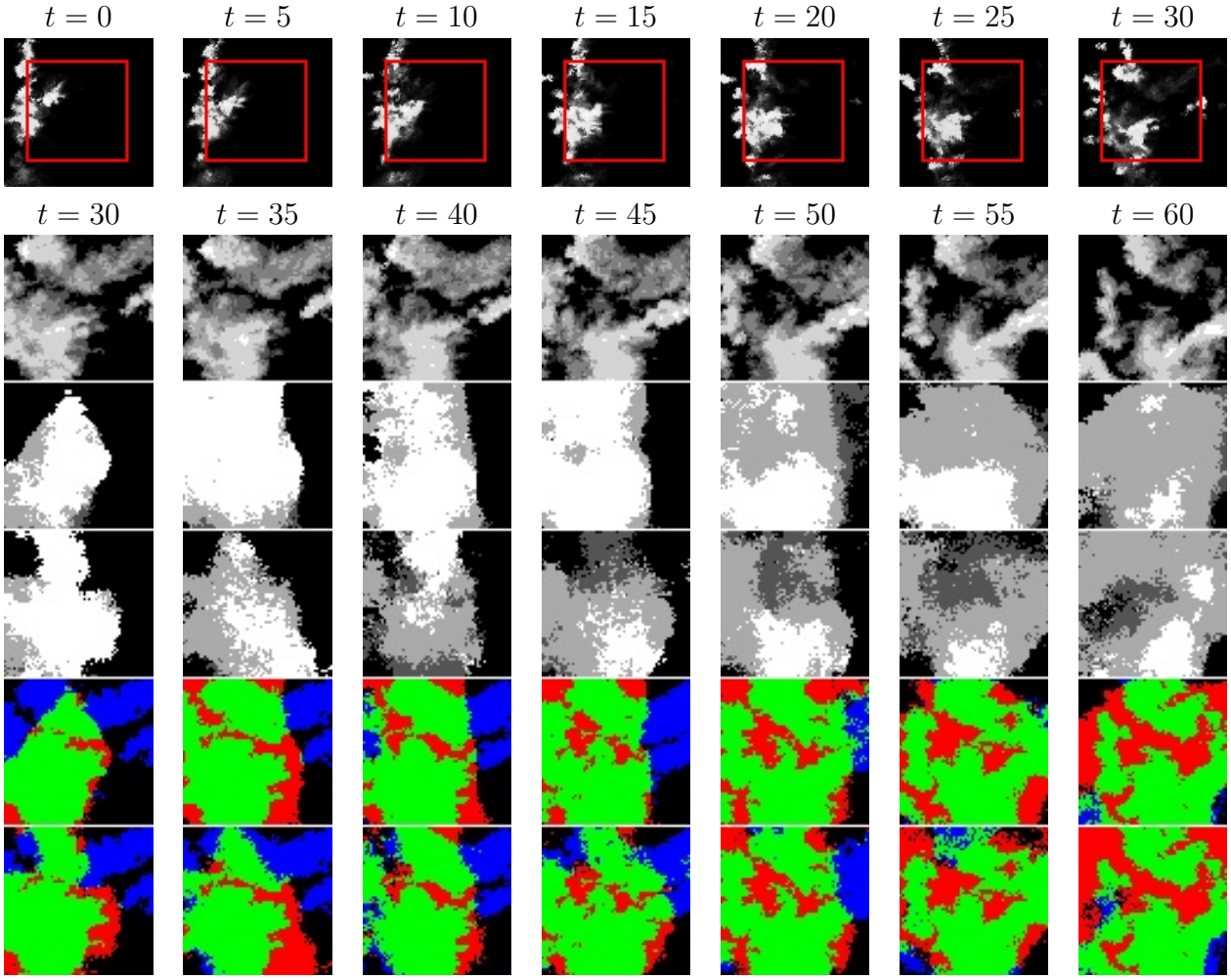


Abbildung 23: Vorhersage in Abständen von fünf Minuten für die Multinomialverteilung. Hier sind die Zeilen wie in 22 dargestellt. Alle Bilder bis auf die Bilder der ersten Zeile sind in sieben Klassen dargestellt.

Auch bei der Multinomialenverteilung wird tendenziell zu viel Regen vorhergesagt. Für die Vorhersage wird die Klasse mit der höchsten Wahrscheinlichkeit herangezogen. Es ist erkennbar, dass die Form der Regenfront nicht korrekt vorhergesagt. Zudem scheint die Anzahl der false negative (Blau) Vorhersagen höher als bei der ZINB. Zusammenfassend ist für die Ausschnitte der Vorhersagen zu sagen, dass die true positive vorhersagen die false negative und false positiv überwiegen. Um eine generelle Aussage treffen zu können wird im weiteren Verlauf die ”receiver operating characteristic” auch **ROC** herangezogen.

In der folgenden Abbildung ist die ROC für beide Architekturen und Verteilungen zu sehen. Die Kurven entstehen indem wir für verschiedene Schwellwerte die binäre Regenvorhersage treffen. Das bedeutet konkret, dass die Wahrscheinlichkeit für kein Regen berechnet wird. Ist diese Wahrscheinlichkeit größer als der Schwellwert so wird auch kein Regen vorhergesagt. Die Schwellwerte liegen gleichmäßig verteilt zwischen Null und Eins.

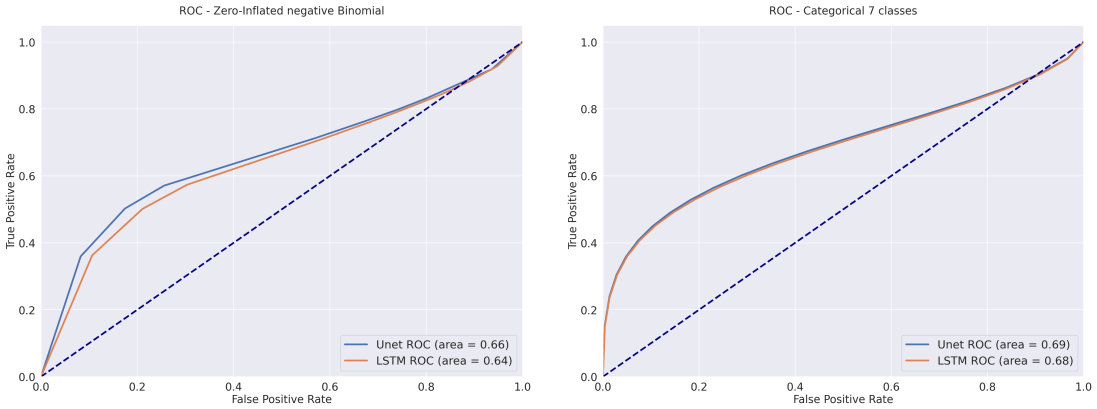


Abbildung 24: ROC/AUR für beide Architekturen und beide Verteilungen. Links für die ZINB und rechts für die Multinomialverteilung. Die Auserwertung erfolgt für 20 verschiedene Schwellwerte.

Wie Anhand der Kurven zu erkennen ist, ist die ROC für die Multinomialverteilung gleichmäßiger als die für die ZINB. Auch die Fläche unter der Kurve (”area under curve” **AUC**) ist für die Multinomialverteilung marginal größer. Interessanterweise ist die Performance des U-nets bei beiden Verteilungen besser (auch hier nur marginal) als für die LSTM-Architektur.

Für den Schwellwert 0.5 erhalten wir in beiden Fällen ein relativ gutes Verhältnis von true positiv zu false positiv. In den nachfolgenden Abbildungen ist die Confusionmatrix für diesen Schwellwert zu sehen.

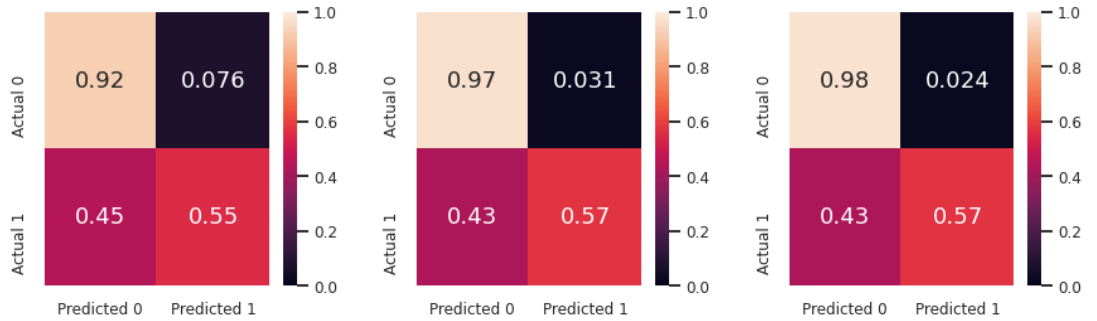


Abbildung 25: Confusionmatrix für die ZINB und dem Schwellwert 0.5. Links die einfache Baseline, mittig für die LSTM-Architektur und rechts für die U-netarchitektur.

Anhand der Confusionmatrix ist zu sehen, dass die Netzwerke in beiden Fällen eine bessere Vorhersage liefern als die einfache Baseline.

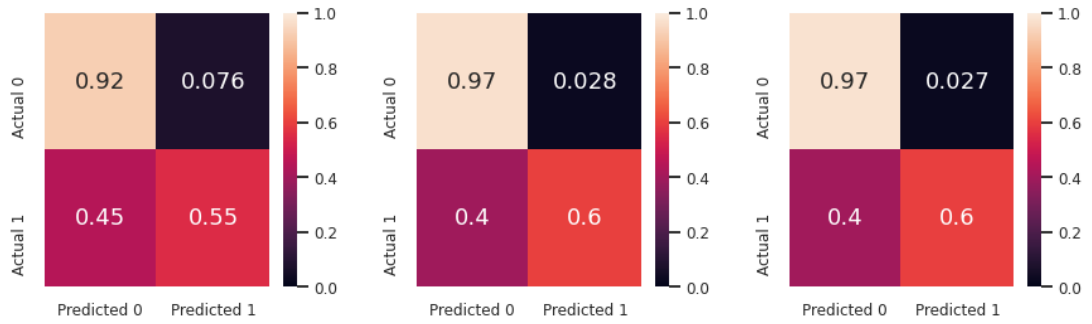


Abbildung 26: Confusionmatrix für die Multinomialverteilung und dem Schwellwert 0.5. Links die einfache Baseline, mittig für die LSTM-Architektur und rechts für die U-netarchitektur.

Auch für die Multinomialverteilung erhalten wir eine bessere Vorhersage als mit der einfachen Baseline. Vergleichen wir die Confusionmatrix beider Verteilungen, so lässt sich feststellen, dass die Verteilungen in verschiedenen Bereichen andere Vorteile aufweisen. So ist die Regenvorhersage für die Multinomialverteilung etwas besser als für die ZINB. Letztere (insbesondere die U-netvariante) produziert mehr true negative und weniger false positiv Vorhersagen als die Multinomialverteilung.

5.3.2 Auswertung der Regenintensität

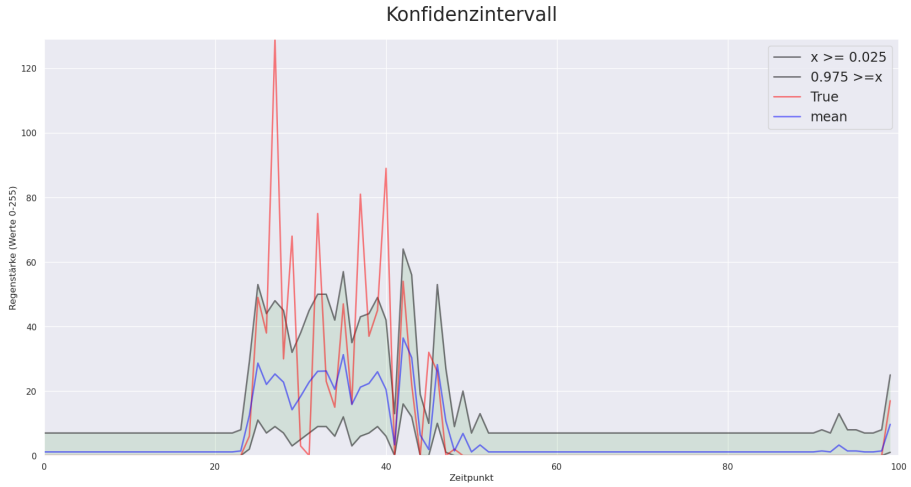
Schauen wir und nun die Klassifikation der Regenintensität für die Multinomialverteilung und die ZINB an. Der Wertebereich der ZINB sind diskrete Werte zwischen 0-255. Die Vorhersage wird hierfür in sieben Klassen umgewandelt, so dass ein direkter Vergleich mit der Multinomialverteilung möglich ist.

	LSTM		U-net	
Klasse	ZNIB	Multi	ZNIB	Multi
0	97 %	97 %	**98** %	97 %
1	**10** %	0 %	**10** %	0 %
2	**42** %	21 %	41 %	23 %
3	30 %	33 %	**34** %	**34** %
4	5 %	**21** %	7 %	20 %
5	0 %	**8** %	0 %	6 %
6	0 %	0 %	0 %	0 %

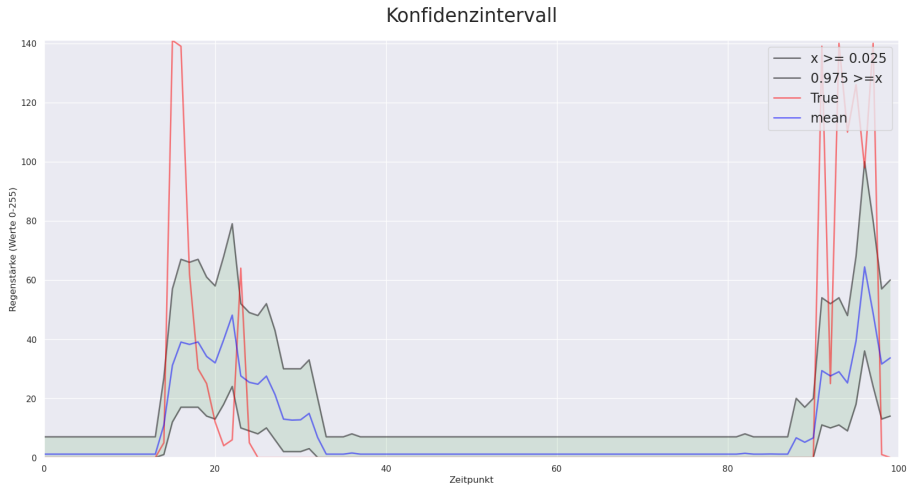
Tabelle 1: Genauigkeit der Klassifikation

Anhand der Tabelle 1 ist zu erkennen, dass die Klasse 6 in allen Fällen falsch vorhergesagt wurde. Eine eindeutig bessere Architektur bzw. Verteilung lässt sich

hier allerdings nicht erkennen. Die ZINB ist allerdings bei den Klassen 0 - 3 genauer als die Multinomialverteilung.
Ein weiterer interessanter Aspekt ist das Konfidenzintervall der ZINB. Insbesondere interessiert uns wie gut die Verteilung die ground truth überdeckt. Hierfür berechnen wir das Konfidenzintervall der Verteilung welches 97,5% der Daten umschließt. Daraufhin zählen wir wie oft der ”wahre” Wert innerhalb des Konfidenzintervalls liegt. In der folgenden Abbildung sind die zwei 97,5 % Konfidenzintervalle für die beiden Architekturen zu sehen.



(a) LSTM-Architektur

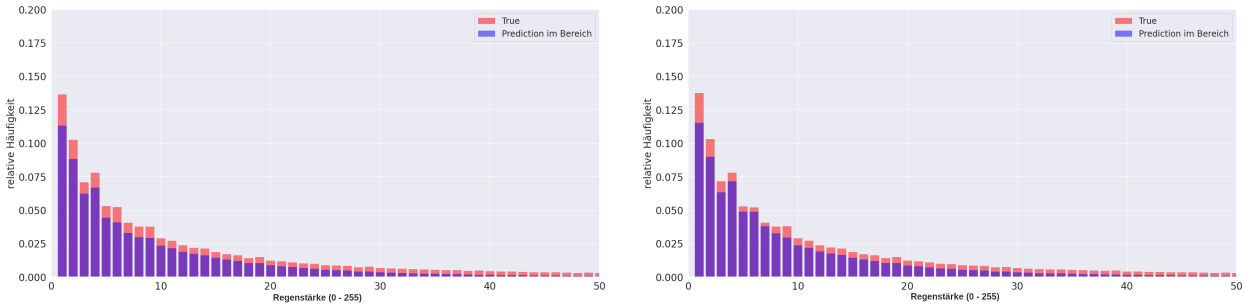


(b) UNET

Abbildung 27: Konfidenzintervall für die ZINB, Oben für die LSTM-Architektur, unten für das UNET.

Die Konfidenzintervalle sind hierbei über eine Sequenz eines zufälligen Ausschnitts eines Pixels berechnet. Sie sind deshalb nicht aussagekräftig für die Vorhersage, sondern sollen lediglich die Erklärung für das Vorgehen vereinfachen. Anhand der Abbildungen ist zu sehen, dass die Konfidenzintervalle in etwa dem ”wahren” Wert folgen. Allerdings ist auch zu sehen, dass Extremwerte meist außerhalb des Intervalls

liegen. Zählen wir nun die Anzahl der Vorhersagen, bei denen der ”wahre” Wert im Konfidenzintervall liegt, so ergibt sich folgendes Histogramm.



(a) LSTM-Architektur (b) UNET

Abbildung 28: Relative Anzahl der ground truth im 97,5% Konfidenzintervall.

Anhand der Diagramme ist zu sehen, dass der ”wahre” Wert in den meisten Fällen von der Verteilung umschlossen wird. Für die Diagramme wurde der Balken der kein Regen repräsentiert weggelassen. Für größere Regenwerte sieht man, dass der ”wahre” Wert selten innerhalb der Konfidenzintervalle liegt. Daraus folgt, dass die Vorhersage mit der Zunahme der Regenstärke schlechter wird. Natürlich liegt der ”wahre” je größer man das Intervall wählt wahrscheinlicher innerhalb des Konfidenzintervalls, weshalb wir das Intervall verkleinern und erneut zählen.

Architektur	inklusive kein Regen	exklusive kein Regen	Konfidenzintervall
UNET	**97,0** %	**73,0** %	97,5 %
	94,0 %	43,0 %	95 %
	90,0 %	35,0 %	90 %
LSTM	96,0 %	71,0 %	97,5 %
	95,0 %	**45,0** %	95 %
	93,0 %	**40,0** %	90 %

Tabelle 2: Relative Anzahl der wahren Werte im Konfidenzintervall für verschiedene Intervalle und Architekturen

In Tabelle 2 ist zu erkennen, dass je kleiner das Konfidenzintervall gewählt wird, desto seltener liegt die ground truth im Konfidenzintervall. Für die Spalte ”inklusive Regen” ist der Unterschied nur marginal. Es ist jedoch zu erkennen, dass für die Auswertung inklusive Regen die ground truth so wie es zu erwarten ist im Konfidenzintervall liegt. Für die Spalte ”exklusive kein Regen” ist ein großer Sprung zwischen dem 97,5 % und dem 95 % Konfidenzintervall zu sehen. Dies kann so interpretiert werden, dass die ground truth für diese Fälle nicht im ”Zentrum” des Konfidenzintervalls liegt (bzw. weiter entfernt vom Zentrum). Der Mittelwert ist in diesen Fällen keine gute Schätzung.

5.3.3 Fazit der Auswertung

Für die binäre Regenvorhersage ist Multinomialverteilung die Verteilung welche die besten Resultate liefert. Laut Abbildung 26 bleibt eine Person zu 60,1 % trocken, wenn sie sich auf die Vorhersage verlässt. Hierbei spielt es keine Rolle ob die UNET oder die LSTM-Architektur verwendet wird. Da der Loss für die LSTM-Architektur besser als für die U-netarchitektur ist(siehe Abbildung 21), bevorzugen wir hier die LSTM-Architektur. Für die Klassifikation der Regenvorhersage liefert die Verteilung der ZINB die besten Resultate (siehe Tabelle 1). Hier ist laut Tabelle 1 die U-netarchitektur marginal besser. Jedoch ist der Loss der LSTM-Architektur besser (siehe Abbildung 20). Zudem liegt laut Tabelle 2 die ground truth für kleinere Konfidenzintervalle öfters in eben diesen Intervallen. Für unsere Vorhersage wäre es optimal, wenn wir für die binäre Regenvorhersage die Multinomialverteilung und für die Regenklassifikation die ZINB in Kombination mit der LSTM-Architektur verwenden. Da dies nicht praktisch ist verwenden wir für die Vorhersage die LSTM-Architektur mit der ZINB.

6 Die DeepRainApp und das Datenbankhandling

Im Folgenden werden die Komponenten des Projektes betrachtet, welche benötigt werden, um die Vorhersage Daten in einer App zu Visualisieren.

6.1 Übersicht

Die Komponente “App und Datenbankhandling” besteht zum wesentlich aus diesen drei Unterkomponenten.

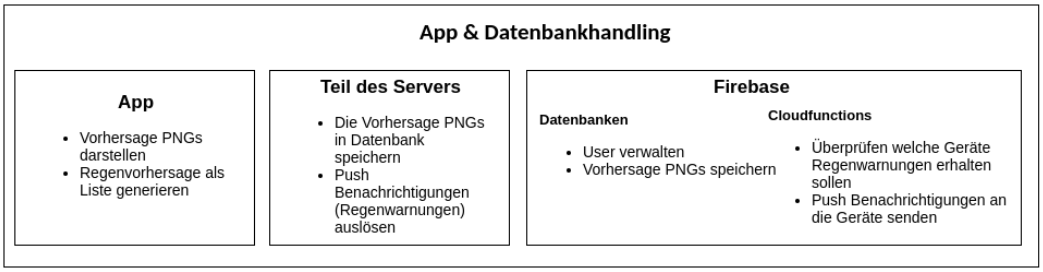


Abbildung 29: Die Komponente App und Datenbankhandling

Die App dient zur Visualisierung der Daten und macht somit die Regenvorhersagen für den Endnutzer brauchbar. Auf dem Server werden die Vorhersagen berechnet und in dem für dieses Kapitel relevanten Teil für die App zur Verfügung gestellt. Außerdem löst der Server die Regenwarnungen (Push Benachrichtigungen) aus, indem er eine Cloud Funktion triggert. Die Firebase verbindet den Server mit der App und erfüllt dabei im Wesentlichen zwei verschiedene Aufgaben. Sie stellt die

Regenvorhersagen als PNG für die App zur Verfügung und übernimmt mit der besagten Cloud Funktion das eigentliche Senden der Regenwarnungen.

6.2 Firebase

Firebase ist eine Entwicklungsplattform für mobile Apps. Diese stellt verschiedene Services zur Verfügung, welche es ermöglichen, effizient Apps für IOS und Android zu entwickeln. Die Firebase ist ein zentraler Baustein der Komponente und wird in jeder Unterkomponente verwendet, weshalb hier einführend ein Überblick gegeben werden soll. Firebase stellt eine Datenbank und einen Cloud-Speicher zur Verfügung, welche genutzt werden, um die User zu verwalten und die Vorhersage PNGs auf den Geräten anzuzeigen. Des Weiteren werden die In-App Messaging Dienste von Firebase verwendet, um die Regenwarnungen in Form von Push Benachrichtigungen zu senden. Die genaue Funktionsweise der Push Benachrichtigungen wird in Kapitel 6.4.1 beschrieben. Firebase ist bis zu einem gewissen Punkt der Verwendung komplett kostenlos. Wird dieser Punkt überschritten, sind die Kosten von der tatsächlichen Nutzung abhängig. In der kostenlosen Version enthalten sind 1 GB Cloud-Speicher, von welchem aktuell nur ein Bruchteil für die 20 PNGs verwendet wird. Des Weiteren können am Tag bis zu 20.000 Dokumente geschrieben und 125.000 Dokumente gelesen werden. Durch die komplett überarbeitete Datenbank und Softwarearchitektur wurden die Datenbanknutzung so weit verringert, dass diese Limitierungen bei weitem nicht erreicht werden sollten.

6.2.1 Datenbank und Cloudspeicher

Die verwendete Datenbank ist ein Firestore von Firebase. Firestore ist ein Cloud NOSQL Datenbanksystem. Die Daten werden in sogenannte Kollektionen und Dokumente eingeteilt. Dabei gehören zu jeder Kollektion Dokumente, in welchen die eigentlichen Daten gespeichert sind. In Abbildung 30 ist der Datenbankaufbau zu sehen.

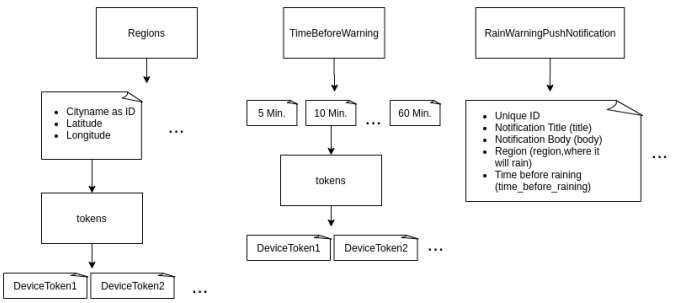


Abbildung 30: Der Aufbau der Kollektionen und Dokumente in der Firebase

Jedes Gerät besitzt einen einmaligen Devicetoken welcher in zwei Kollektionen gespeichert wird. Die eine Kollektion steht für den Zeitpunkt, in dem die Regenwarnung gesendet werden soll, die andere Kollektion steht für die Region, in

der die App verwendet wird. Diese beiden Kollektionen werden für das Senden der Push Benachrichtigungen benötigt, auf welches in dem Kapitel 6.4.1 eingegangen wird. In den Einstellungen der App kann eingestellt wann die Regenwarnung als Push-Benachrichtigung gesendet werden soll. Je nachdem, was der User einstellt, wird sein Devicetoken in eine andere Kollektion gespeichert. Dabei gibt es für jede einstellbare Zeit ein eigenes Dokument in der Kollektion TimeBeforeWarning. Wenn die Netze Regen vorhersagen, wird vom Server ein Dokument in RainWarningPushNotification gepusht. Dieses Dokument wird von einer Cloud-Function (Siehe Kapitel 6.4.1) verwendet, um die Pushbenachrichtigungen an die richtigen Geräte zu senden. Der Cloud Storage von Firebase wird verwendet, um die PNGs mit der jeweiligen Vorhersage zu speichern. Dabei lädt der Server alle 5 Minuten die neuen Vorhersagen hoch. Diese PNGs werden in der App angezeigt.

6.3 Server

Mit der Serverkomponente ist im folgenden der Teil des Servers gemeint, der für die Kommunikation mit der Firebase verantwortlich ist. Dazu gehört das Bereitstellen der aktuellsten Regendaten im Bildformat sowie das Triggern von Push Benachrichtigungen. Da zu Beginn des Projektes noch keine Vorhersagen von den Netzen zur Verfügung stand, wurde ein Programm entwickelt, dass den Server simuliert und zufällig generierte Regendaten in der Firebase speichert. Somit konnte die App unabhängig und parallel zu den Netzen entwickelt werden. Bei der Entwicklung der Pipeline (siehe Kapitel 7) konnten die meisten Komponenten dieses Programms übernommen werden. Wenn die Netze eine neue Regenvorhersage berechnet haben, werden die daraus resultierenden Vorhersagebilder in den Firestore hochgeladen. Um die Push Benachrichtigungen auszulösen, muss für jede Region, in der sich ein Nutzer befindet, der Pixel der Region berechnet werden. Dafür werden für alle vorhandenen Regionen, in denen Device Tokens gespeichert sind, es also Nutzer in der Region gibt über den jeweiligen Breiten und Höhengrad der dazugehörige Pixel im Vorhersagebild für die Region berechnet. Der Wert der jeweiligen Pixel wird ausgelesen. Liegt dieser Farbwert über einem bestimmten Grenzwert, wird ein Dokument in der Kollektion RainWarningPushNotification gespeichert. Hierdurch wird eine Cloudfunktion getriggert, welche sich um das senden der Push Benachrichtigungen kümmert.

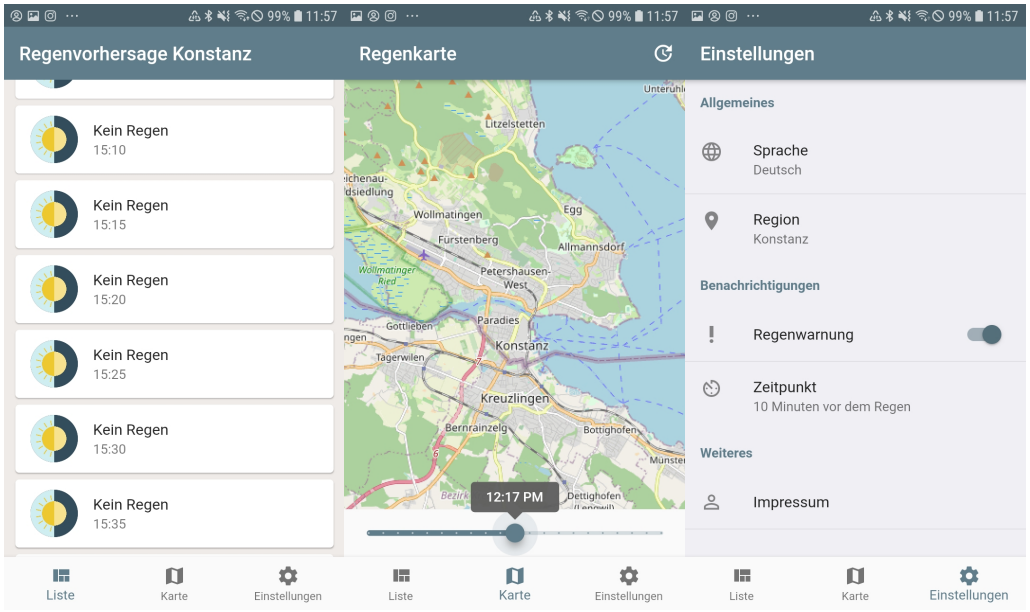
6.3.1 Funktionen der App

Die App soll die von den Netzen berechnete Vorhersagen visualisieren und dem Benutzer zur Verfügung stellen. Die Daten werden dabei sowohl in Tabellarischer als auch in Form einer Karte dargestellt. Dabei wird gewährleistet, dass immer die aktuellsten Daten zur Verfügung stehen. Außerdem wird der Benutzer benachrichtigt,

sobald es eine Regenwarnung gibt. Der Zeitpunkt der Regenwarnung kann eingestellt werden. Für Präsentationszwecke wurde der Demo Modus eingeführt. In diesem werden Beispiel Vorhersage Daten angezeigt.

6.3.2 Screens der App

Im Wesentlichen besteht die App aus drei Screens. Einem Screen zum Anzeigen der Daten in Listenform, einem zum Anzeigen der Daten auf einer Karte und den Einstellungen. Die jeweiligen Screens können über die Bottomnavigation erreicht werden, somit ist es möglich, intuitiv zwischen den einzelnen Screens zu wechseln. Im Folgenden wird auf die einzelnen Screens und deren technische Funktionsweise genauer eingegangen.



(a) (b) (c)

Abbildung 31: Die drei Hauptscreens der App

Regenvorhersage als Liste

Auf diesem Screen werden die von den Netzen berechneten Regenvorhersagen angezeigt. Dabei wird in die drei Kategorien “Kein Regen”, “Leichter Regen” und “Starker Regen” unterschieden. Je höher die berechnete Regenintensität ist, je dunkler wird der Regenschirm, welcher zu Beginn jedes einzelnen Listeneintrages zu sehen ist. Die anzuzeigenden Daten werden während dem Appstart in der Klasse ProvideForecastData gespeichert und können währende der Laufzeit von dort gelesen werden.

Regenvorhersage als Karte

Auf diesem Screen werden die von den Netzen erzeugten PNGs visualisiert. Dafür werden die PNGs mit einer Karte hinterlegt, auf welcher der User frei navigieren

kann, um die aktuelle Regensituation an jedem beliebigen Ort zu prüfen. Dabei wird standardmäßig der Kartenausschnitt von der Region angezeigt, die in den Einstellungen eingestellt wurde. Mit dem Slider kann der Zeitpunkt eingestellt werden, in dem die Regenvorhersage angezeigt werden soll. Mit dem Aktualisieren Button in der Actionbar können die neusten Bilder vom Server heruntergeladen werden. Aufgrund der verhältnismäßig großen Datenmenge werden die neuen Vorhersagen nicht automatisch vom Server heruntergeladen. Im Normalfall werden die Bilder einmalig bei dem App Start heruntergeladen.

Einstellungen

Auf diesem Screen können alle relevanten Einstellungen gemacht werden. Dazu gehört z.B. die Sprache der Benutzeroberfläche. Außerdem kann die Region eingestellt werden Die hier ausgewählte Region wird standardmäßig auf der Karte angezeigt und nur für diese Region werden Regenwarnungen gesendet. Unter der Benachrichtigungskategorie können die Regenwarnungen aktiviert und der Zeitpunkt der Regenwarnung eingestellt werden. Jede Aktion in dieser Kategorie löst verschiedene Datenbankaufrufe aus. Wenn die Regenwarnung aktiviert wird, wird der Devicetoken von dem Gerät in die Datenbank hochgeladen, beim Deaktivieren wird der Devicetoken gelöscht. Wenn der Zeitpunkt der Regenwarnung verändert wird, wird der Devicetoken in der Datenbank von einer Kollektion in eine andere Kollektion verschoben. Alle gemachten Einstellungen werden in sogenannten Shared Preferences gespeichert, damit sie auch nach Appstart noch vorhanden sind. In Abbildung 32 ist der Datenfluss beim Anpassen des Zeitpunktes für die Regenwarnung dargestellt.

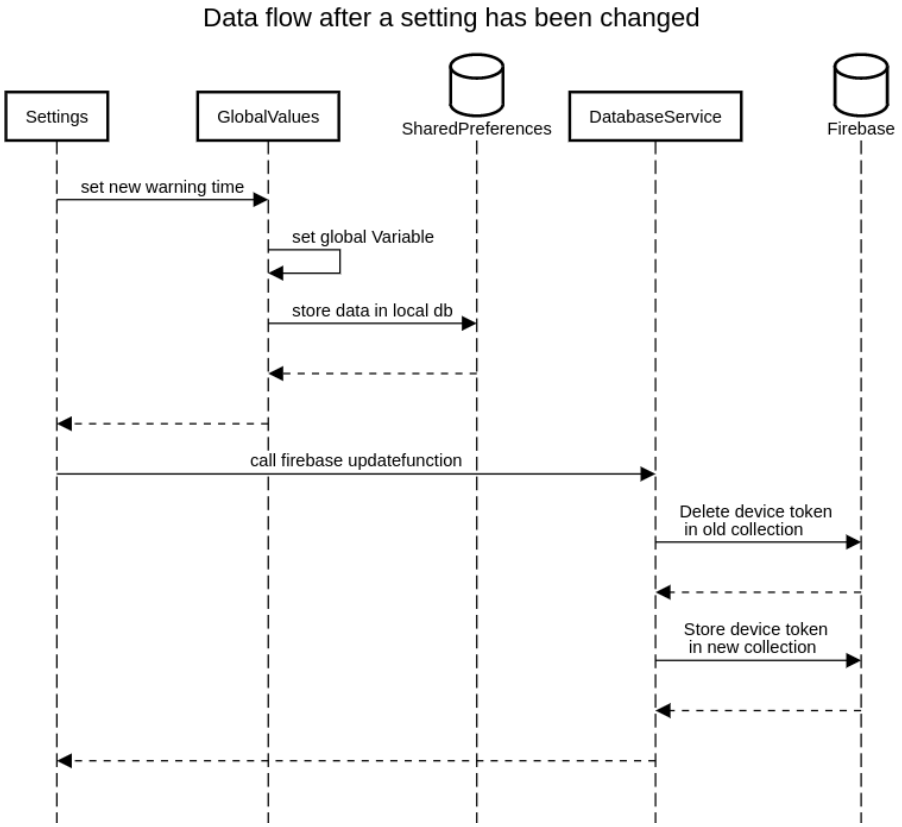


Abbildung 32: Der Datenfluss beim ändern des Zeitpunktes der Regenwarnung

6.3.3 Der Appstart

Während dem Appstart wird die App für die Verwendung vorbereitet, Einstellungen werden wiederhergestellt und die aktuellen Regenvorhersagen werden aus der Datenbank heruntergeladen. Die Einstellungen können dabei aus den Shared Preferences ausgelesen werden. Die Shared Preferences sind eine lokale Key-Value Datenbank, in der alle benutzerspezifischen Daten gespeichert werden. Bei dem App Start werden die Einstellungen aus den Shared Preferences gelesen und auf globale Variablen gespeichert, somit sind sie während der Laufzeit der App dynamisch verfügbar. Außerdem wird bei dem ersten Appstart die Position in dem Vorhersagebild berechnet. Diese wird benötigt, um aus den Vorhersagebildern den richtigen Pixel auszulesen und in der Vorhersageliste anzuzeigen. Der Appstart ist in folgender Abbildung schematisch dargestellt.

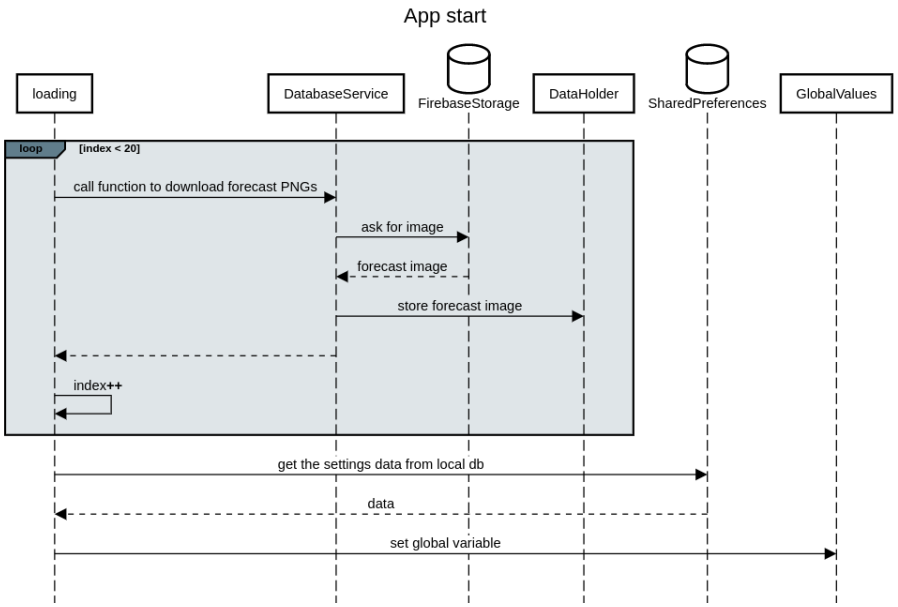


Abbildung 33: Datenfluss beim starten der App inklusive der Datenbankabfragen (Server und Lokale Datenbank)

6.3.4 Die Berechnung der Regenintensität

Um die aktuelle Regensituation auf einer Liste anzuzeigen, muss der Farbwert aus dem richtigen Pixel in der Vorhersage ausgelesen werden. Zu Beginn wurde angenommen, dass die App nur in Konstanz verwendet werden soll. Im Laufe des Projektes stellte sich jedoch heraus, dass die entwickelten Netze in Zukunft auch in der Lage seien könnten, Vorhersagen für ganz Deutschland zu machen. Da die Softwarearchitektur nicht für eine solche Anwendung ausgelegt war, mussten einige Änderungen vorgenommen werden. Bis zu diesem Zeitpunkt wurden die Vorhersage Daten für jeden Pixel auf dem Server berechnet und im Anschluss in der Firebase gespeichert. Bei verschiedenen Nutzern in verschiedenen Regionen kommt diese Architektur allerdings schnell an seine Grenzen. Hat die App bspw. 1000 Nutzer in verschiedenen Regionen, müssen für jeden der 1000 Nutzer alle fünf Minuten 20 Vorhersage Daten hochgeladen werden. Daher musste der Datenfluss so umstrukturiert werden, dass der neue Regenwert direkt in der App berechnet wird. Dabei muss das Handy den entsprechenden, eigenen Pixel auf der Karte berechnen. Die Berechnung hierfür ist verhältnismäßig aufwendig, da mit großen Listen (810.000 Einträge) gearbeitet werden muss. Auf diese Berechnung wird in Abschnitt 6.3.5 eingegangen.

Wenn die Bilder beim Appstart oder bei einem Vorhersageupdate heruntergeladen werden, wird von jedem Bild der Regenwert in dem entsprechenden Pixel berechnet. Es wird eine Liste mit ForecastListItem Objekten erstellt. Diese wird global gespeichert und in der Vorhersage Liste angezeigt.

6.3.5 Berechnung des Pixels

Um die Regensituation an dem jeweiligen Ort des Users auszuwerten, muss der Pixel in dem Vorhersagebild berechnet werden. Hierfür dienen zwei Listen welche die Latitude und Longitude Werte für jeden Pixel enthalten. Die Koordinaten in den einzelnen Indizes kombiniert geben die Position der einzelnen Pixel im Weltkoordinatensystem an. Diese Listen wurden in Python mit der Wradlib erstellt, und anschließend im JSON Format in die App übertragen. Somit steht jeder Index für eine Position im Weltkoordinatensystem, ausgedrückt durch Höhen und Breitengrad Informationen. Da das Bild eine Auflösung von 900x900 Pixeln hat, sind diese Listen 810.000 Elemente groß.

listLatitude	listLongitude	listCoordinates
46.95351109003129	3.6010757050026125	[0, 1]
46.95444001727318	3.6132220366153205	[0, 2]
46.955367192755986	3.625368944704319	[0, 3]

Abbildung 34: Der Exemplarische Aufbau der Listen zur Berechnung des eigenen Pixels

Jeder User hat eine eigene Position in Deutschland, welche in Form von Höhen und Breitengrad bekannt ist. Diese wird durch die in den Einstellungen festgelegte Region geändert und festgelegt. Es wird nun ein Algorithmus gesucht der mithilfe von einer Koordinate, bestehend aus Latitude und Longitude werten, den dazugehörigen Pixel in dem Vorhersage PNG findet. Nun wäre es natürlich möglich, durch alle Indizes zu Iterieren und somit den richtigen Pixel zu finden. Ein solcher Bruteforce Ansatz ist bei einem Regenbild von n auf n Pixeln aufgrund der Laufzeit von $O(n^2)$ nicht zielführend.
Der erste Ansatz basierte auf dem Teile und Herrsche Prinzip. Dieser auf einem Quadtree basierende Algorithmus, teilt das Regenbild in vier Bereiche auf und berechnet anschließend in welcher der vier Mittelpunkte der aktuellen Position am nächsten ist. Anschließed wird der Bereich erneut in vier Teilbereiche unterteilt und die Berechnung wiederholt sich. Dieser Prozess wird so lange wiederholt, bis man den Pixel indem sich die aktuelle Position befindet ausgemacht hat. Wie bereits in 3 beschrieben, ist der abgedeckte Kartenbereich nicht rechteckig. Die daraus resultierenden Nichtlinearität verursachte bei diesem Verfahren in bestimmen Fällen fehler. Daher wurde ein neuer, stabilerer, Algorithmus entwickelt.
Das neue Verfahren wird von den Problemen des alten Verfahrens nicht beeinflusst, ist dafür allerdings im Durchschnitt ein bisschen langsamer. Dieser Laufzeitunterschied ist für unsere Anwendung vernachlässigbar. Es wird an einem Startpixel gestartet, von dem aus die Distanz zu den Koordinaten des Gerätes berechnet wird. Dann wird die Distanz der Nachbarpixel zu den Koordinaten des Gerätes berechnet. Wenn eine

Verringerung der Distanz durch einen Nachbarpixel erzielt werden kann, wird einer der Nachbarpixel zu dem neuen betrachteten Kästchen. Von diesem aus werden wieder alle Nachbarpixel überprüft. Dieser Vorgang wird so oft wiederholt, bis keine Verbesserung der Distanz mehr erzielt werden kann.

6.3.6 Framework Entscheidung

Bei der Entwicklung einer App steht die Frage der zu bedienenden Plattformen an erster Stelle. Soll die App zum Beispiel nur unternehmensintern verwendet werden oder ist das Gerät auf dem sie verwendet wird eine Neuanschaffung kann es ausreichend sein nativ auf einer Plattform zu entwickeln. Soll jedoch, wie bei den meisten Apps, eine breite Zielgruppe angesprochen werden, ist es unerlässlich, die App auf IOS und Android zur Verfügung zu stellen. Je nachdem, auf welchen Betriebssystemen die App verwendet werden soll, muss eine komplett andere Frameworkwahl getroffen werden. Einige bekannte hybride Frameworks sind Xamarin, React Native oder Flutter. Jedes dieser Frameworks ist zukunftsträchtig und wurde von großen Unternehmen auf den Markt gebracht. So steht Microsoft hinter Xamarin, Facebook hinter React Native und Google hinter Flutter. Je nach Frameworkwahl kann ca. 80-100 Prozent von dem kompletten Code für beide Betriebssysteme verwendet werden. Dafür sind Cross-Plattform Frameworks oft nicht so performant wie native. Dies fällt besonders bei rechenaufwendigen Apps und Spielen ins Gewicht. Bei so einer leichten App wie DeepRain ist dieser Leistungsunterschied vernachlässigbar. Die Entscheidung fiel auf Flutter. Ein großer Vorteil von Flutter ist, dass 100 Prozent der Codebasis für Android und IOS übernommen werden können. Außerdem hat die Prominenz von Flutter in den letzten Jahren, seit der Veröffentlichung, stark zugenommen. In der folgenden Abbildung sind die Google Suchanfragen für den Begriff Flutter abgebildet.

Abbildung 35: Entwicklung der Suchanfragen für den Begriff 'Flutter'[Trends, 2020]

6.3.7 Technischer Aufbau von Flutter

Flutter Anwendungen werden in der Programmiersprache Dart geschrieben und können anschließend für IOS, Android, Windows, Linux, MacOs und als WebApp veröffentlicht werden. Dart bringt dabei im Vergleich zu Java Script (React Native) den Vorteil mit, dass es objektorientiert ist, was vor allem in größeren Softwarearchitekturen zum Tragen kommt. Auch alle Bibliotheken um Code asynchron auszuführen, werden

bereits von Dart mitgeliefert. Die wohl größte Rolle für jeden Dart Entwickler spielen die sogenannten Widgets. Widgets sind einzelne Bausteine, welche die UI repräsentieren. Jedes UI Element ist dabei ein eigenes Widget. Dabei werden Widgets oft ineinander geschachtelt, was es ermöglicht, komplexere UI's zu entwerfen. Dabei werden Widgets in Stateless und Statefull Widgets unterschieden. Während ein Stateless Widget keine Daten und somit keinen Zustand speichern kann, ist das mit einem Stateful Widget möglich.

6.4 Cloudfunktionen

Mit den Cloudfunktionen von Firebase kann Backend-Code direkt auf den Servern von Google gespeichert und ausgeführt werden. Dabei ist es möglich, auf bestimmte Datenbankaktionen zu reagieren. Diese Funktionalität wird verwendet, um Push Benachrichtigungen zu senden. Der Code wird in Java Script geschrieben und anschließend als Cloud Funktion hochgeladen.

6.4.1 Push Benachrichtigungen

Zum Warnen der Benutzer, wenn es eine Regenvorhersage gibt, werden Push Nachrichten verwendet. Der Zeitpunkt der Push Benachrichtigungen kann in der App eingestellt werden. So kann man sich zwischen 5 und 60 Minuten vor dem bevorstehenden Regen warnen lassen. Um eine Pushbenachrichtigung zu versenden, speichert der Server ein Dokument in der Firebase welches alle für die Pushbenachrichtigung relevanten Informationen enthält. Dazu gehört zum Beispiel der Push Benachrichtigungstitel und Text, sowie die Zeit bis der Regen eintritt. Sobald das neue Dokument mit den Informationen für die Push Benachrichtigung hochgeladen wurde, wird eine Callback Funktion in Form einer Cloudfunktion aufgerufen. Je nachdem, zu welchem Zeitpunkt ein User die Regenwarnung erhalten möchte, wird sein Device Token in eine andere Kollektion gespeichert. Außerdem wird je nach Region, in der sich der User befindet, sein Token in einer anderen Kollektion gespeichert. Nur die Tokens, die sowohl mit dem Zeitpunkt, als auch mit der Region, aus dem vom Server hochgeladenen Dokument übereinstimmen, sollen eine Pushbenachrichtigung erhalten. Die Funktion findet diese Schnittmenge und weiß somit, welche Geräte eine Pushbenachrichtigung erhalten sollen.

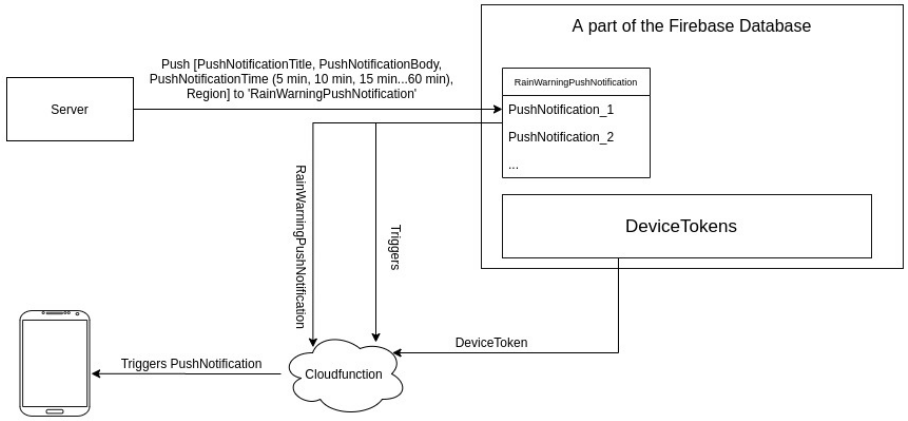


Abbildung 36: Funktionsweise des Prozesses zum Senden von Push - Benachrichtigungen.

6.5 Vorgehen bei Entwicklung

In Flutter entwickelte Apps können sowohl auf Android als auch auf IOS ausgeführt werden. Um eine Flutter App auf einem IPhone auszuführen, wird allerdings MacOS als Betriebssystem benötigt. Da während des Entwicklungsprozesses nur ein Linux Rechner zur Verfügung stand, wurde die App lange Zeit nur unter Android getestet. Erst gegen Ende des Projektes wurde der Code für IOS kompiliert und für das IPhone angepasst. Da die gesamte Pipeline zu Beginn noch nicht funktioniert hat, wurde der für die App relevante Teil der Pipeline simuliert. Dazu wurde ein Pythonprogramm geschrieben, welches reale Daten in die Firebase pushed. Somit war es möglich, gekapselt vom Rest des Projektes zu arbeiten und die App fertigzustellen.

7 Pipeline

Nachdem die einzelnen Komponenten für sich funktionieren gilt es sie in einer Pipeline zu kombinieren und so ein funktionierendes Gesamtkonzept zu erhalten. Der dadurch entstehende Workflow lässt sich in die Bereiche Datenbeschaffung, Regenvorhersage sowie Datenaufbereitung und Bereitstellung aufteilen. Diese werden im Folgenden erläutert.

7.1 Datenbeschaffung

Zum Training sowie zur Validierung der Netze konnten bisher historische Daten verwendet werden. Um eine Praxistaugliche Wettervorhersage zu berechnen, muss diese allerdings in der Zukunft liegen. Die Input Daten der Netze müssen daher möglichst aktuell sein. Hiefür wurden, wie bereits bei den Historischen Daten, Daten des DWDs genutzt. Dieser stellt alle fünf Minuten aktuelle Binärradardaten bereit.

Das Skript prüft alle fünf Sekunden ob neue Daten bereitstehen, falls das der Fall ist, werden diese heruntergeladen und anschließend wie in Abschnitt 4 erläutert in Bilder mit einer Auflösung von 900x900 Pixeln konvertiert. Anschließend wird ein Ausschnitt des Bildes genommen welcher gegebenenfalls skaliert werden kann. So kann der Ort für die Spätere Wettervorhersage leicht angepasst werden.

7.2 Regenvorhersage

Die Vorhersage selbst hängt stark von den eingesetzten Netzen ab. Derzeit werden drei gleiche Netze eingesetzt welche auf 10, 20 und 30 Minuten vorhersagen trainiert wurden. Als Input für jedes Netz werden die 5 aktuellsten Bilder verwendet. Deren Werte werden normiert und anschließend in einen Batch mit der Shape (1, 96, 96, 5) gebracht.

7.3 Datenaufbereitung und Bereitstellung

Die berechnete Vorhersage hat einen Wertebereich von 0-255 und eine Auflösung von 64x64 Pixel. Um die Vorhersage in der App richtig darstellen zu können muss diese noch weiterbearbeitet werden. Um vorhersagen flexibel an verschiedenen Orten oder in verschiedenen Auflösungen bereitstellen zu können ohne die App anzupassen, stellt diese für jede Vorhersage ein PNG mit 900x900 Pixeln dar. Daher muss der Ausschnitt mit der Vorhersage bzw. mit den historischen Daten an der richtigen Position des PNGs eingefügt werden. Die Pixel außerhalb dieses Ausschnitts sind transparent. Die unterschiedlichen Regenstärken werden mit drei verschiedenen Blautönen dargestellt. Da die Regenintensität wie in Abschnitt 4 erläutert nicht gleichverteilt ist, wurden die Schwellwerte zwei und zehn gewählt. Zudem wird jedem historischen Bild sowie jeder Vorhersage einen Zeitpunkt zugewiesen welcher später auf dem Slider in der App visualisiert wird. Nun können die fünf historischen Regenbilder sowie die drei Vorhersagen in der Firebase-Datenbank ersetzt und so der App, also dem Nutzer bereitgestellt werden. Der Vorgang wiederholt sich und es wird alle fünf Sekunden geprüft ob auf dem Server des DWDs neue Daten vorhanden sind.

8 Ausblick

Entzerrung der Vorhersage PNGs

Um die App endgültig praxistauglich zu machen, muss die Projektion der PNGs richtig funktionieren. Hierfür müsste die Krümmung der Erde aus den Vorhersage PNGs rausgerechnet werden, oder die PNGs dementsprechend angepasst werden. Nur dann ist es möglich, die Regenvorhersagen den richtigen Regionen zuzuordnen. Auf diese Thematik wird in Kapitel 3 genauer eingegangen.

Optimierung der Netzwerke

Das Optimieren unserer Netzwerke, insbesondere das Verändern der Architektur ist ein Aspekt bei dem wir sehr großes Verbesserungspotential sehen. Die Architektur unserer Netzwerke berücksichtigt kaum die Tatsache, dass die Regenbilder zusammenhängende Sequenzen sind. Insbesondere das Unet vergisst nach jeder Vorhersage den vorherigen Zustand. Diesen Umstand in die Architektur mit einzubeziehen könnte sich positiv auf die Vorhersage auswirken. Eines der schwerwiegenderen Probleme ist das Ungleichgewicht der Klassen. In Kapitel 5.2 erklären wir, dass wir davon ausgehen, dass die Landschaft um Konstanz einen direkten Einfluss auf das Wetter hat. Hier sehen wir einen Punkt, an dem gearbeitet werden kann. Beispielsweise könnte man die komplette Regenkarte zum trainieren nutzen.

9 Fazit

Mit DeepRain wurde ein Grundbaustein gelegt, auf dessen Basis in Zukunft aufgebaut werden kann. Es wurde ein komplett funktionsfähiges Gesamtsystem entwickelt, dass die Daten vom Server des DWD bis an den User bringt. Es wurde eine App entwickelt die alle Anforderungen erfüllt und stabil funktioniert. Es ist mit minimalem Aufwand möglich, zukünftige Netze in das Gesamtsystem und die App zu integrieren. Um das Gesamtsystem praxistauglich zu machen, müssen die Netze weiterentwickelt werden. Durch das miterleben eines Projektes dieses Umfangs, konnten wir viel lernen. Durch die breit gefächerten Aufgabengebiete war es uns möglich in viele verschiedene Themengebiete einen Einblick zu erhalten. Gleichzeitig konnte sich jedes Teammitglied einen eigenen Schwerpunkt setzten, in dem es besonders tief in die Materie einsteigen konnte. Nur durch die gute Teamarbeit und Koordination war es möglich, ein Projekt dieser Größe umzusetzen.
Wir persönlich sind mit dem Endergebnis mehr als zufrieden und hoffen, dass das Projekt erfolgreich weitergeführt wird.

Literaturverzeichnis

[DWD, 2019] DWD (2019). Radargestützte analysen stündlicher niederschlagshöhen im echtzeitbetrieb für deutschland (radolan) und mitteleuropa (radolan-me). https://www.dwd.de/DE/leistungen/radolan/radarniederschlagsprodukte/radolankurzbeschreibung_pdf.pdf?__blob=publicationFile&v=6.

[Hosseini et al., 2019] Hosseini, M., Maida, A. S., Hosseini, M., and Raju, G. (2019). Inception-inspired lstm for next-frame video prediction.

[Mingels, 2016] Mingels, G. (2016). Früher war alles schlechter - wetterprognosen. *Spiegel*, (38).

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

[Trends, 2020] Trends, G. (2020). Google trends for 'flutter'. https://trends.google.de/trends/explore?date=today 5-y&geo=DE&q=flutter.

[Wetterdienst, 2020] Wetterdienst, D. (2020). Radolan (radar-online-aneichung): Analysen der niederschlagshöhen aus radar- und stationsbasierten messungen im echtzeitbetrieb. DWD. https://www.dwd.de/DE/leistungen/radolan/radolan.html.