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: 6. September 2020

Betreuer: Prof. Dr. Oliver Dürr

Zusammenfassung

Hier steht der Text, welcher den Inhalte der Arbeit zusammenfasst...

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Here goes the English text which summarizes the content of the thesis...

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

Abbildungsverzeichnis

KNN Künstliches neuronales Netz

Literatur

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

[Musterfrau, 2005] Musterfrau, M. (2005). Ein weiteres beispielbuch.

[Mustermann, 2009] Mustermann, M. (2009). Ein beispielbuch.

1 Gesamtsystem

Die komplette Datenpipeline von dem Server des DWD bis zur Darstellung in der App sieht wie folgend aus.

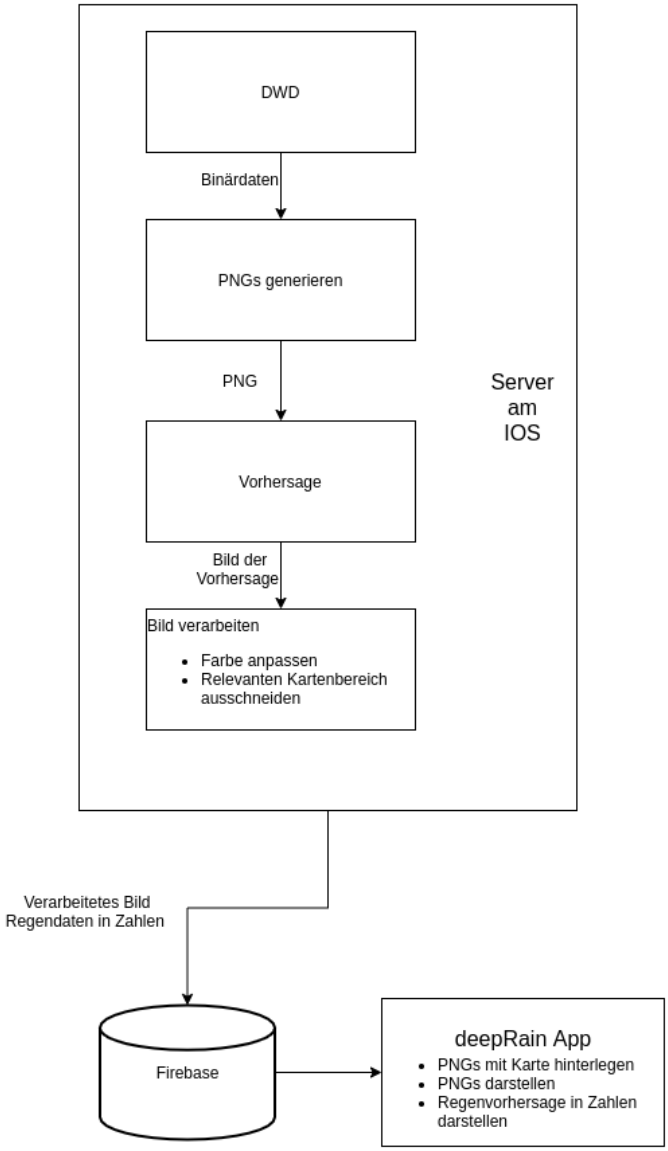


Abbildung 1: Das Gesamtsystem

Der DWD stellt alle fünf Minuten die neusten Regendaten für Deutschland im

Binär Format zur Verfügung. Diese werden heruntergeladen und in der Datenaufbereitung verwendet um PNGs zu generieren (Siehe Kapitel “Datenaufbereitung”). Dieses PNGs werden im Anschluss verwendet um eine Regenvorhersage zu machen (Siehe Kapitel “Regenvorhersage / Netze”). 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 “App und Datenbank”). 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 Binär - Daten 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 mit der Firebase, und um die App welche die Daten darstellt. (Hier sollten wir gegen Ende des Projektes eventuell noch ein bisschen genauer beschreiben wie das alles genau funktioniert. Das wird unseren Nachfolgern sehr stark helfen um sich schnell einzuarbeiten)

2 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 und den punktuellen Messungen von über 2000 Bodenniederschlagsstationen (https://www.dwd.de/DE/leistungen/radolan/radolan.h Eine dieser Bodenniederschlagsstationen befindet sich in Konstanz.

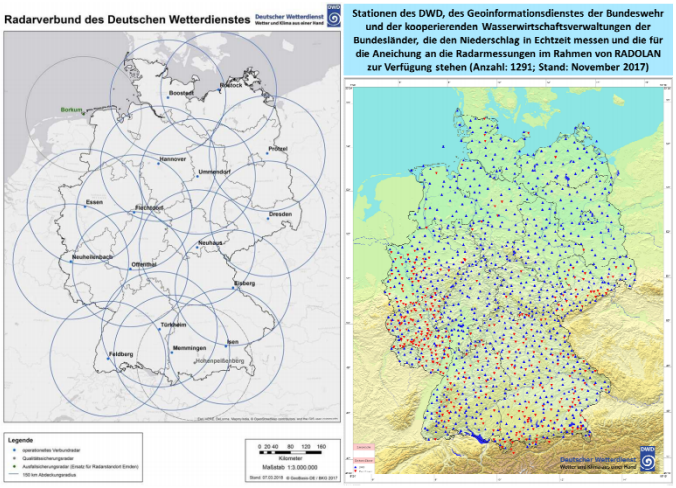


Abbildung 2: Übersicht über alle Boden und Radarstationen

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 zehn minütiger Auflösung zur

Verfügung. Da sich herausstellte, dass bei der ein minütigen Auflösung Daten Fehlen, wurden die Daten der 10 minütigen Auflösung verwendet um die PNGs zu validieren. 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 und Breitengrad Koordinaten. In folgender Abbildung ist der Aufbau der Binärdaten zu sehen.

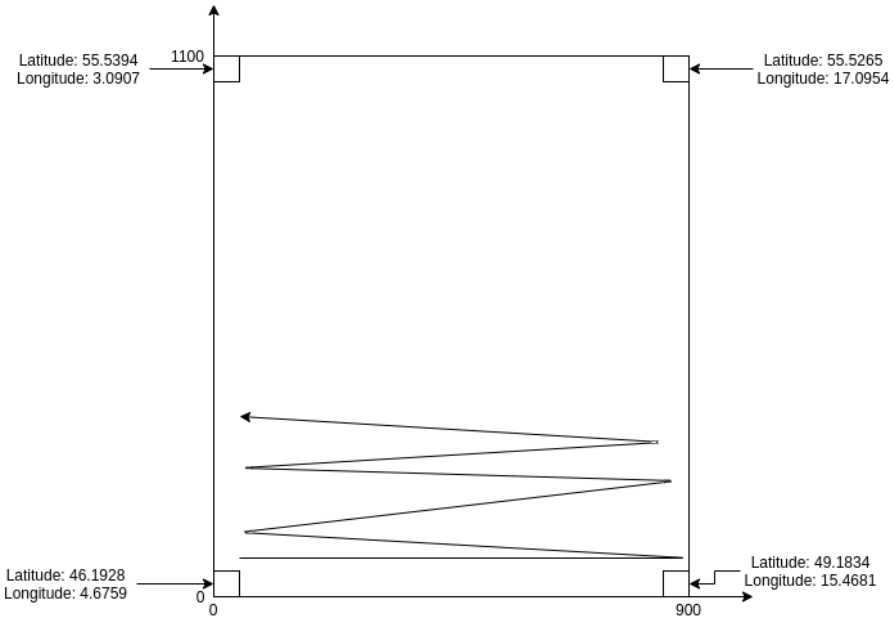


Abbildung 3: Der Aufbau des Koordinatensystems der Radar Daten

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 90 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.

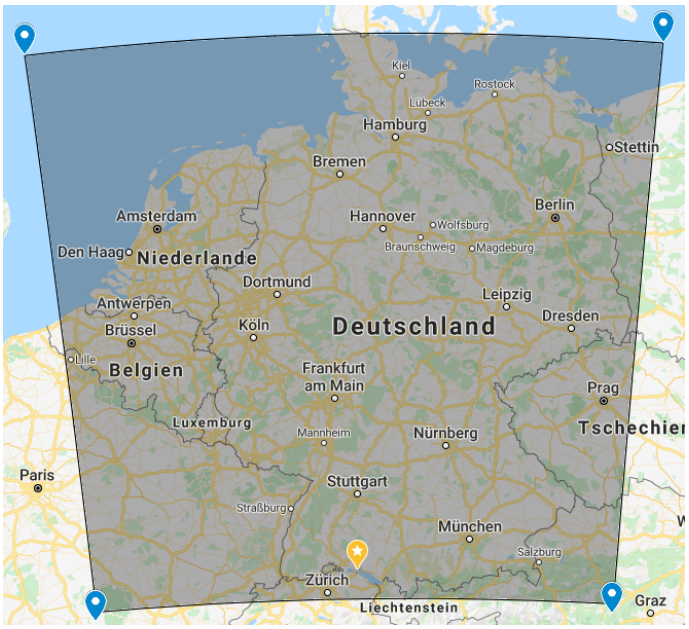


Abbildung 4: Die von den Daten abgedeckte Fläche auf einer 2D Karte

3 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 Sakalierungsfaktor 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.

Geplottet werden alle auftretenden Werte sowie dessen Häufigkeit. Hierbei stechen 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 dem folgenden Plot werden die beiden Ausreißer gefiltert, da die relevanten Informationen in den restlichen Datenpunkten stecken.

In diesem Plot 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. Die beiden folgenden Plots zeigen verschiedene Perzentile und machen so die Datenverteilung deutlich.

Das 99 Prozent-Perzentil 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. Der folgende Plot zeigt die Verteilung der noch verbleibenden Daten.

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. Da für die Berechnung des Skalierungsfaktors auch nicht der tatsächliche Maximalwert genutzt wird werden Werte die nach der Transformation über 255 sind auf 255 gesetzt. 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. Radardaten welche transformiert werden müssen:

Die transformierten Daten befinden sich in einem Wertebereich von 0 bis 255.

Rekonstruiert man aus dem PNG Daten wieder die Radardaten ergibt sich der folgende Plot. Die quantitativ wiederhergestellte Anzahl der Datenpunkte beträgt 100

4 Die neuronalen Netze

Die Regenvorhersage mit Hilfe von künstlichen neuronalen Netzen 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 erwiedert:

> "...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.

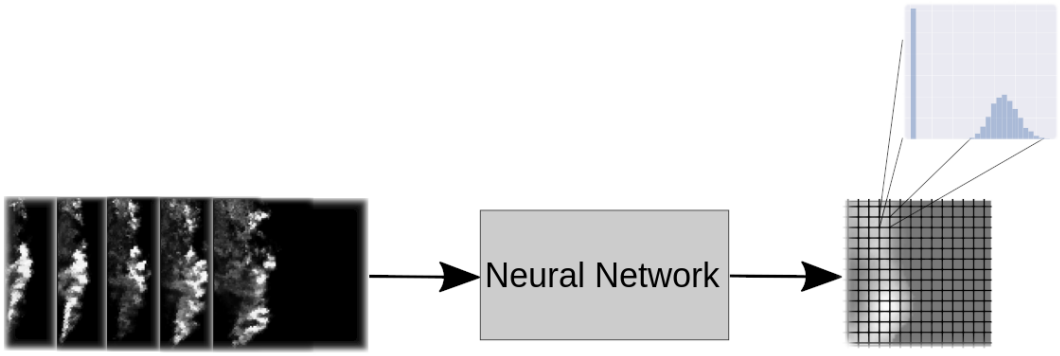


Abbildung 5: Skizzierung der Vorhersage

Wie aus Abbildung 5 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, die Poisson und die negativ Binomial Verteilung an. Die Poisson und die negativ Binomialverteilung wird allerdings

mit der Multinomialen Verteilung gemischt, sodass wir die ”Zero-Inflated negativ Binomial” bzw. ”Zero-Inflated Poisson” Verteilung 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 Binomial bzw. Poisson Verteilung zurück.

4.1 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 Netzwertopologie 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.

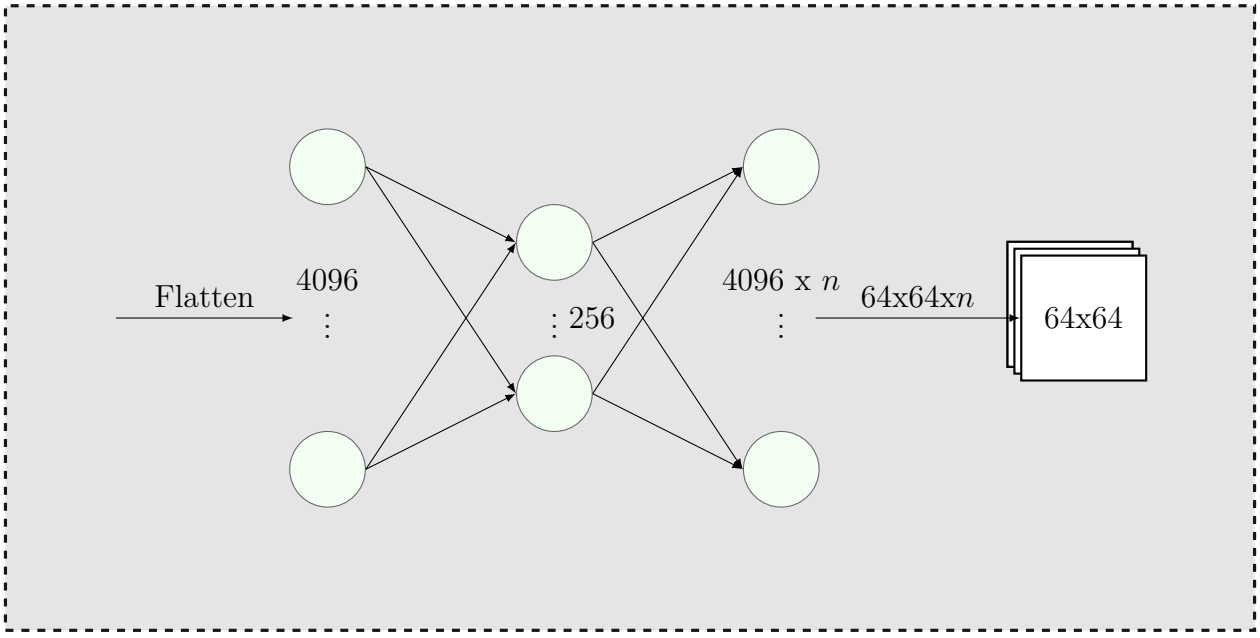


Abbildung 6: Klassifikationsschicht, n variiert hierbei je nach Verteilung. Für die Zero-Inflated Poisson Verteilung ist n gleich 2, für die Zero-Inflated Negativ Binomial Verteilung ist n gleich 3, und für die Multinomiale Verteilung ist n gleich 7

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

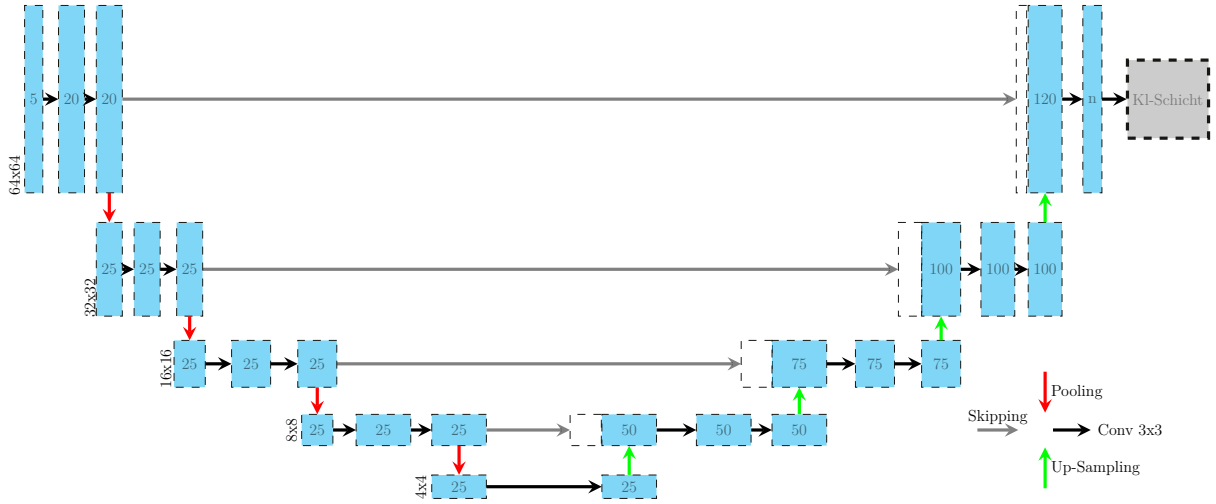


Abbildung 7: Abgespekte Version des Unets

Die abgespekte Version des Unets hat im Gegensatz zur originalen Architektur wesentlich weniger Gewichte. Hier sind es ca 10 000 trainierbare Gewichte. Hierbei liegt der Hauptteil der Gewichte in der Klassifikationsschicht (Kl-Schicht in Bild 7).

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 aufeinander folgender eingehender Bilder wird eine plausible vorhersage für das nächste Bild erzeugt.

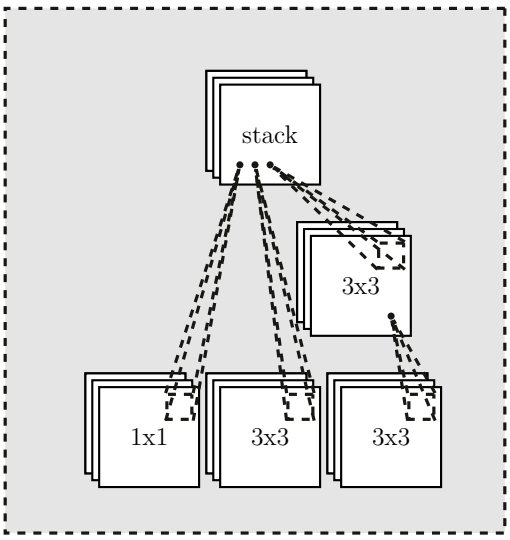


Abbildung 8: Aufbau der von uns verwendeten Inception Layer, angelehnt an die inception Layer im Paper

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.

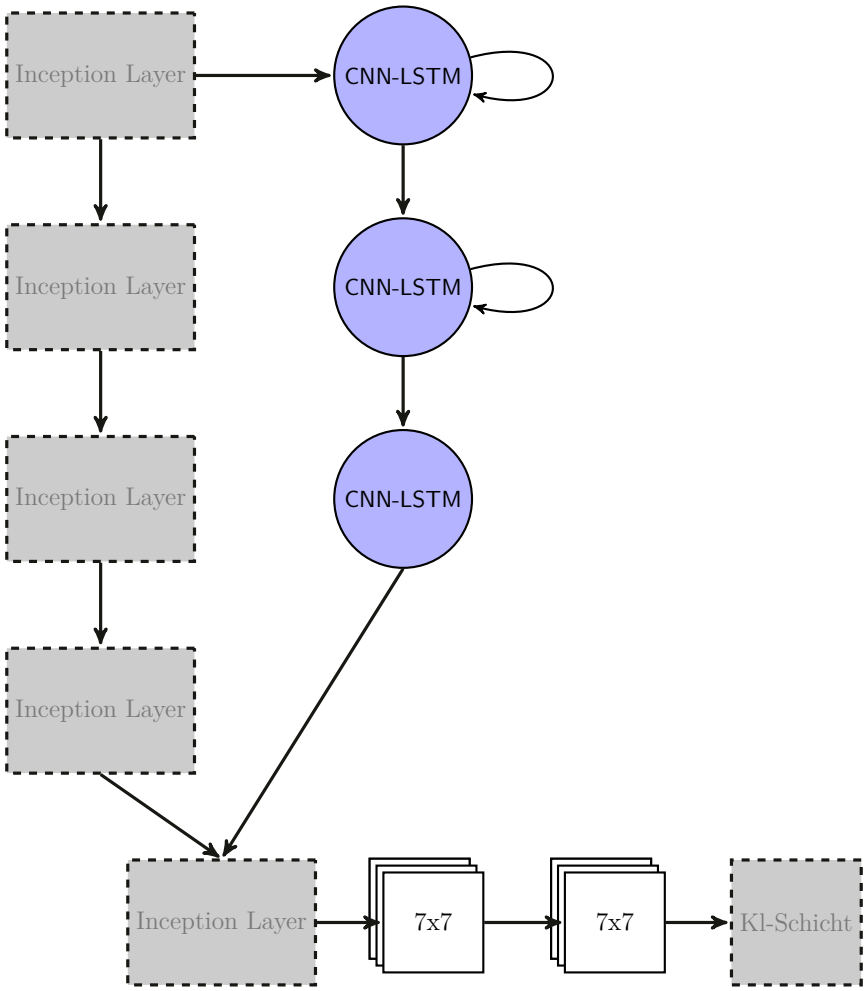


Abbildung 9: LSTM Architektur

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.

4.2 Training

Beide Architekturen werden mindestens 35 Epochen trainiert, wobei die Architektur mit den LSTM-Layern 1:30 Stunden benötigt, das Unet hingegen benötigt ca. 10 Minuten pro Epoche. Für das Training nutzen wir alle Daten der Jahre 2008 bis einschließlich 2017. 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.

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. 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 Zero-Inflated negative Binomial Verteilung.

Abbildung 10: Trainingskurven für Zero-Inflated negativ Binomial verteilung

Zu sehen ist, dass die Unet-Architektur 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 Zero-Inflated negative Binomial Verteilung wurden beide Architekturen mit einer weiteren Verteilung trainiert. Hierfür verwenden wir die Multinomiale Verteilung, woebei wir die Daten in Sieben Klassen einteilen. Die geschieht durch logarithmisches skalieren der Daten. Dadurch soll zusätzlich dem Klassenungleichgewicht entgegengesteuert werden (Regenwerte im höheren Bereich kommen seltener vor).

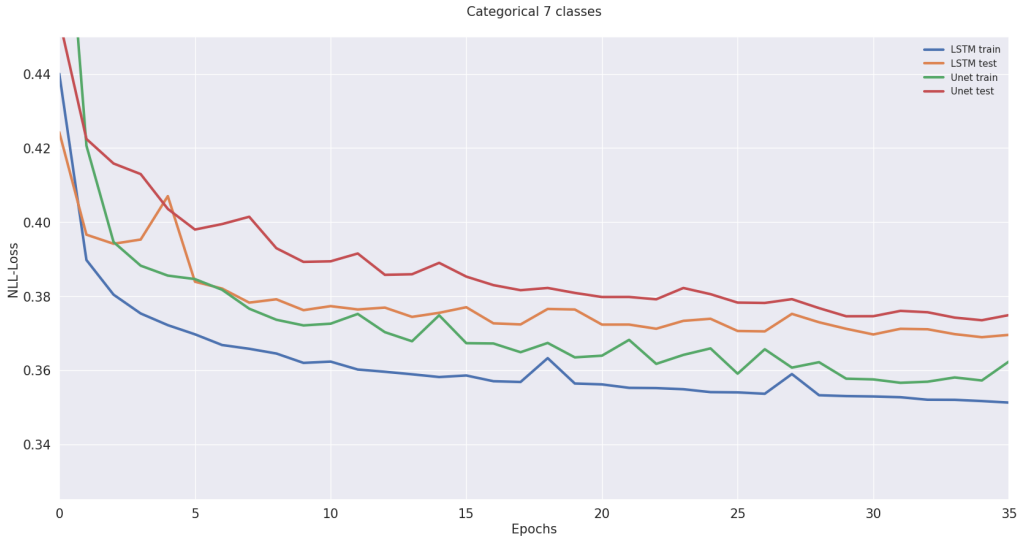


Abbildung 11: Multinomiale Verteilung

In der obigen Abbildung sind die Trainingskurven der beiden Architekturen zu sehen. Auch hier ist zu sehen, dass die LSTM-Architektur der Unet-Architektur 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 Zero-Inflated negative Binomialverteilung entspricht.

4.3 Auswertung

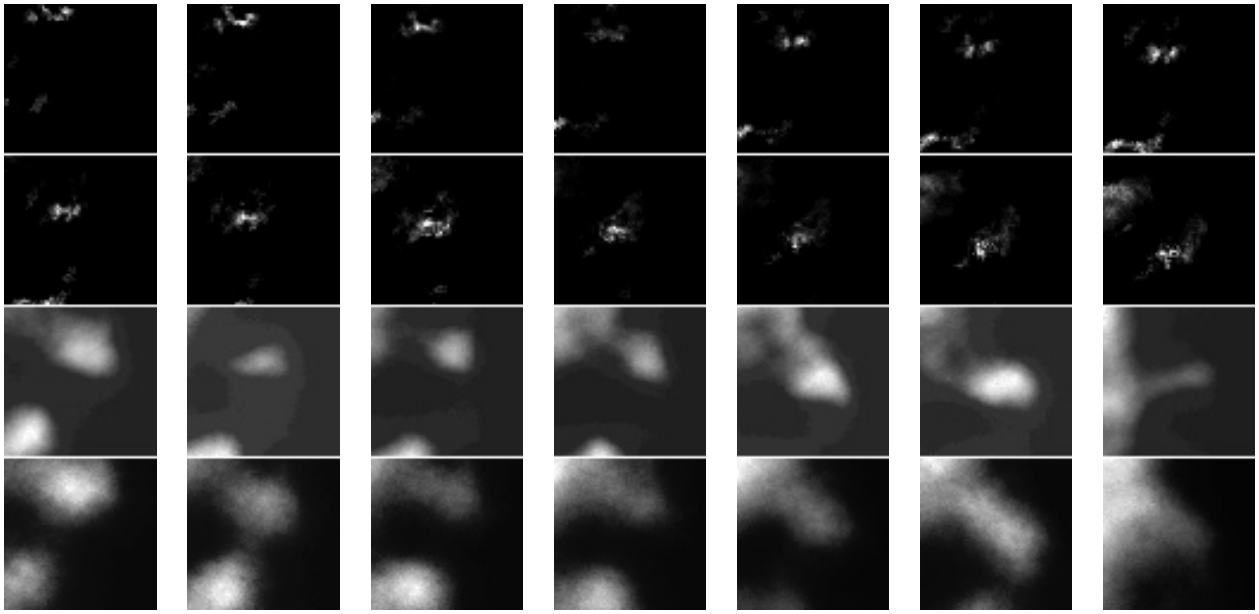


Abbildung 12: Vorhersage für die Zero-Inflated negative Binomialverteilung, erste Zeile entspricht dem momentanen Regen. Zeile zwei entspricht dem Regen in 30 Minuten. Zeile drei ist die 30 Minuten Vorhersage des Unets. Die letzte Zeile ist die 30 Minuten Vorhersage der LSTM-Architektur

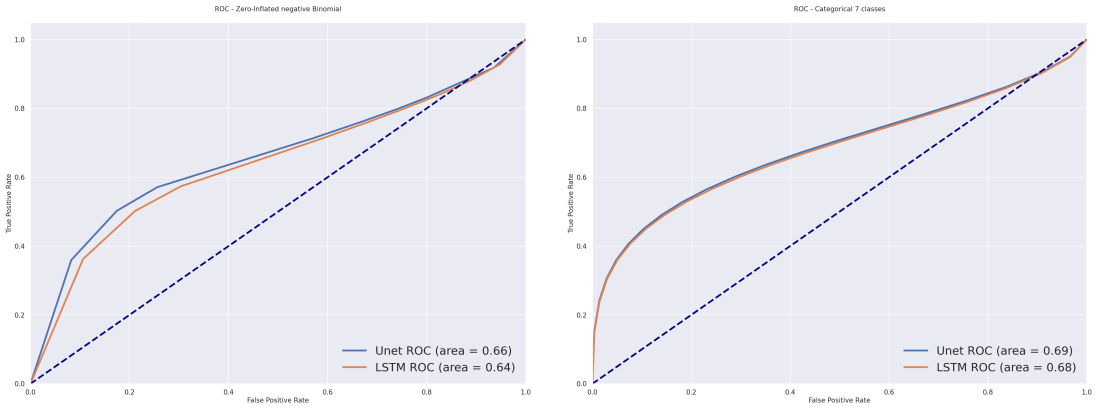


Abbildung 13: ROC/AUR für beide Architekturen und beide Verteilungen. Links für die Zero-Inflated negativ Binomialverteilung und rechts für die Multinomialeverteilung. Die Auserwertung erfolgt für 20 verschiedene Thresholds.

5 Die DeepRainApp und das Datenbankhandling

In den folgenden Abschnitten werden alle Komponenten behandelt welche benötigt werden um die Vorhersage Daten in der DeepRain App anzuzeigen und die volle Funktionalität der App zu gewährleisten.

5.1 Übersicht

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

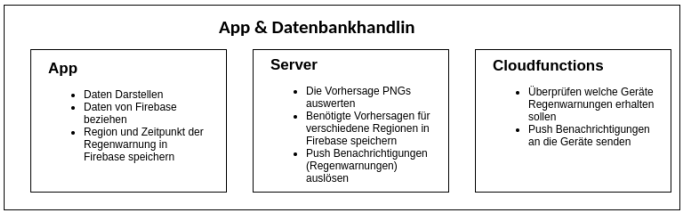


Abbildung 14: Die Komponente App und Datenbankhandling

Die App dient im Wesentlichen 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. Die Cloudfunktion, welche in der Firebase gespeichert ist, ist für das senden der Push Benachrichtigungen an das Gerät verantwortlich. Das senden der Push Benachrichtigungen wird jedoch vom Server getriggert.

5.2 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 eine Datenbank und einen Cloudspeicher zur Verfügung welcher genutzt wird um die Regendaten und Vorhersage PNGs auf den Geräten anzuzeigen und mit dem Server zu synchronisieren. Abgesehen von dem Datenbankservice übernimmt Firebase auch die Authentifizierung der einzelnen Geräte und es wäre möglich das Usermanagement zu handhaben, was in deepRain aber nicht benötigt wird. Des Weiteren werden die In-App Messaging dienste von Firebase verwendet um die Regenwarnungen in Form von Push Benachrichtigungen zu senden. Die genaue Funktionsweise wird in dem Kapitel “Push Nachrichten” 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 Cloudspeicher, von welchem aktuell nur ein Bruchteil für die 20 PNGs verwendet wird. Des Weiteren können am Tag bis zu 20.000 Dokumente geschrieben werden, aktuell werden ca. 400 Dokumente geschrieben. Außerdem können 125.000 Dokumente gelesen werden, was ca. 2500 Appstarts am Tag entspricht und ca. 10.000 Push Benachrichtigungen im Monat versend werden.

5.3 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 15 ist der Datenbankaufbau zu sehen.

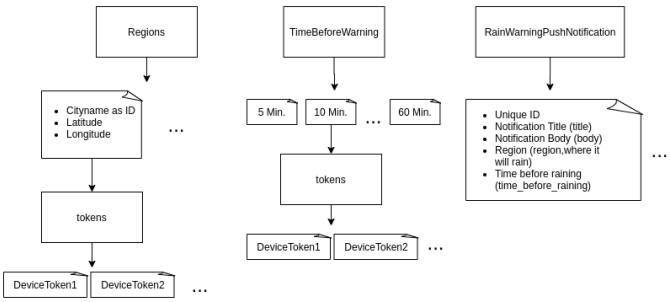


Abbildung 15: Der Aufbau der Kollektionen und Dokumente in der Firebase

Jedes Gerät besitzt einen einmaligen Device Token welcher in zwei Kollektionen gespeichert wird. Die Eine Kollektion steht für den Zeitpunkt in dem die Regenwar-

nung 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, in dem Kapitel 5.6.1 wird darauf genauer eingegangen. In den Einstellungen der App kann eingestellt werden 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 5.6.1) verwendet um die Push Benachrichtigungen an die richtigen Geräte zu senden. Der Cloud Storage von Firebase wird verwendet um die PNGs mit der jeweiligen Vorhersage zu speichern. Dabei pushed der Server alle 5 Minuten die neuen Vorhersagen. Diese PNGs werden in der App angezeigt.

5.4 Server

Mit der Serverkomponente ist 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. Wenn die Netze eine neue Regenvorhersage getroffen haben, werden die daraus resultierenden Vorhersage Bilder in den Firestore hochgeladen. Um die Pushbenachrichtigungen 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 für die Region berechnet. Der Wert der jeweiligen Pixel wird ausgelesen. Liegt dieser Farb, oder auch Regenwert, über einem bestimmten Grenzwert, wird ein Dokument in der Kollektion RainWarningPushNotification gespeichert. Hierdurch wird eine Cloudfunktion getriggert, welche sich um das senden der Pushbenachrichtigungen kümmert.

5.5 Die App

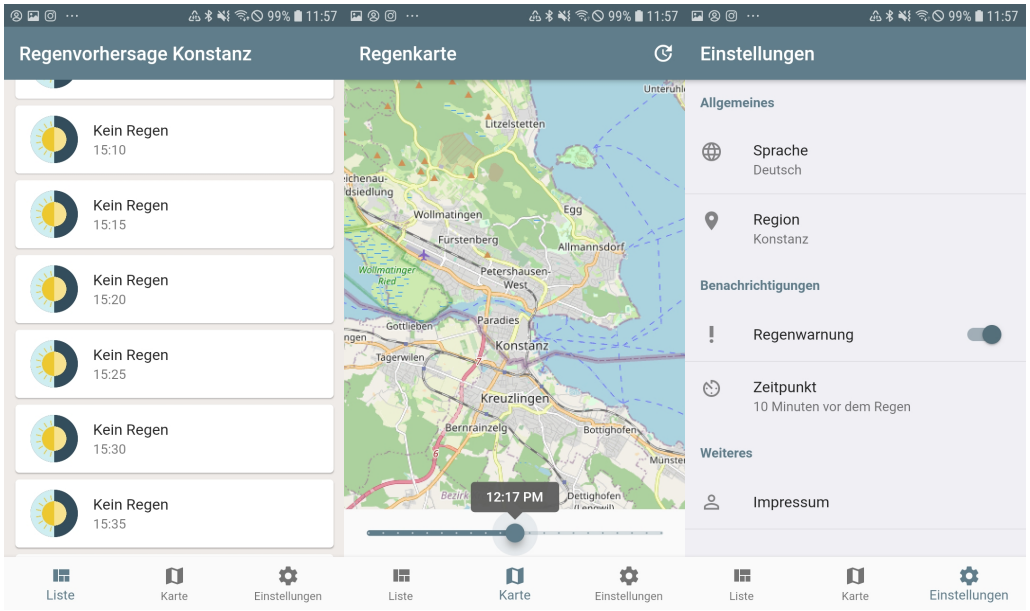
5.5.1 Funktionen der App

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.

5.5.2 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.



(a) (b) (c)

Abbildung 16: 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.

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. 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 Benachrichtigung's Kategorie können die Regenwarnungen Aktiviert werden sowie der Zeitpunkt der Regenwarnung eingestellt werden. Jede Aktion in dieser Kategorie löst eine Datenbankaktion aus. Wenn die Regenwarnung aktiviert wird, wird der Device Token von dem Gerät in die Datenbank hochgeladen, beim Deaktivieren wird der Device Token gelöscht. Wenn der Zeitpunkt der Regenwarnung verändert wird, wird der Device Token in der Datenbank von einer Kollektion in eine andere Kollektion verschoben. Alle gemachten Einstellungen werden in sogenannten Shared Preferences gespeichert, damit sie auch nach App Start immer noch vorhanden sind. Die Shared Preferences sind eine Key-Value Datenbank die auf dem Gerät gespeichert wird. Bei App Start werden die Einstellungen aus den Shared Preferences gelesen und auf globale Variablen gespeichert. Diese Variablen sind dafür entscheidend, welche Einstellungen wiederhergestellt werden. In der folgenden Abbildung kann der Datenfluss nachdem der Zeitpunkt der Regenwarnung verändert wurde nachvollzogen werden.

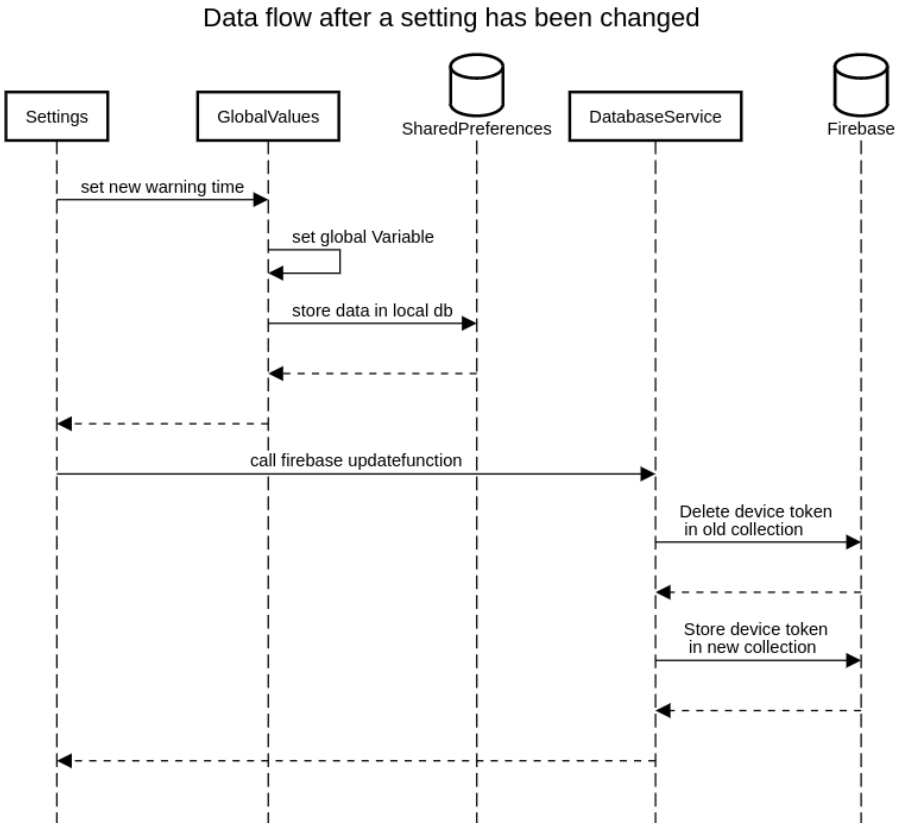


Abbildung 17: Der Datenfluss beim ändern des Zeitpunktes der Regenwarnung

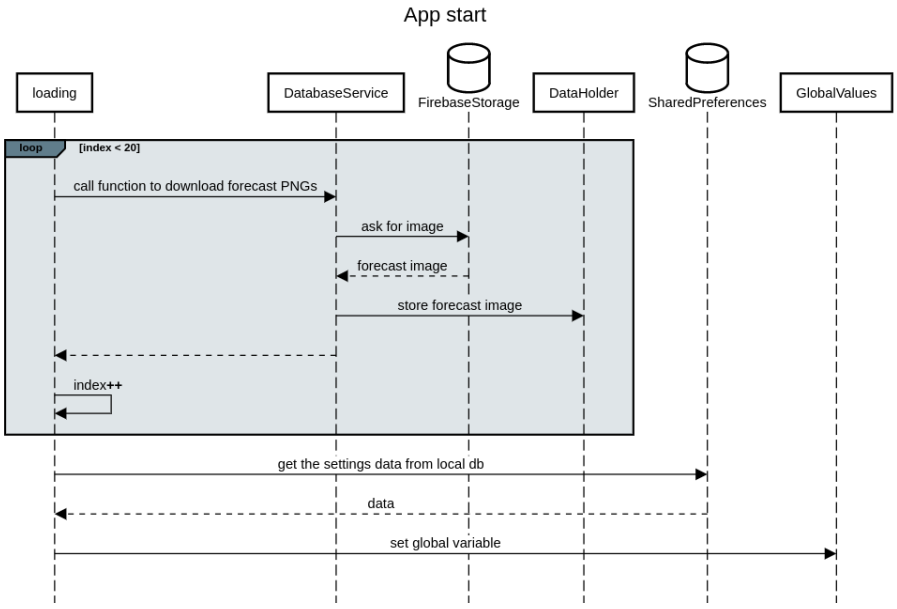


Abbildung 18: Datenfluss beim starten der App inklusive der Datenbankabfragen (Server und Lokale Datenbank)

Die Berechnung der Regenintensität

Zu beginn wurde angenommen, dass die App nur in Konstanz verwendet werden soll. Im Laufe des Projektes stellte sich jedoch heraus, dass die entwickeleten Netze in

der Lage sind, Vorhersagen für ganz Deutschland zu machen. Da die Software Architektur 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 aufwändig, da mit großen Listen (810.000) Einträgen gearbeitet werden muss. Auf diese Berechnung wird in Abschnitt 5.5.2 eingegangen.

Wenn die Bilder beim Appstart oder bei einem Vorhersageupdate heruntergeladen werden, wird von jedem Bild der Regenwert in dem entsprechnden Pixel berechnet. Es wird eine Liste mit ForecastListItem Objekten erstellt. Diese wird global gespeichert und in der Vorhersage Liste angezeigt. Aktuell können nur beim App Start und durch manuelles auslösen eines Updates die Vorhersage Daten aktualisiert werden.

Berechnung des Pixels

Um die Regensituation an dem jeweiligen Ort des Users auszuwerten, muss der Pixel in dem Vorhersagebild berechnet werden. Hierfür dienen drei verschiedene Listen. Diese Listen wurden in Python mit der Wradlib erstellt, und anschließend im JSON Format in die App übertragen. Zwei der Listen enthalten alle Latitude bzw. Longitude Werte. Die Koordinaten in den einzelnen Indizes Kombiniert geben die Position der einzelnen Pixel im Weltkoordinatensystem an. In der dritten Liste steht zu jedem Index die jeweilig zugehörige Pixel Koordinate im Bild. Somit steht jeder Index für eine Position im Weltkoordinatensystem, ausgedrückt durch Höhen und Breitengrad Informationen, und der Abbildung dieser Position auf den Vorhersagebild. 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 19: 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 angaben bekannt ist. Diese wird durch die in den Einstellungen festgelegte Region bestimmt. Es wird nun also ein Algorithmus gesucht, der mithilfe der Höhen und Breitengrad Informationen den Pixel im Bild findet, der am besten zu diesen Koordinaten passt. Nun wäre es natürlich möglich, durch alle Indizes zu Iterieren und somit den richtigen Pixel zu finden. Dieses Verfahren ist aber sehr Zeit und Rechenintensiv und daher nicht geeignet um es auf einem Smartphone auszuführen.

Wir brauchen noch eine Lösung!!

5.5.3 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 nach dem auf welchen Betriebssystemen die App verwendet werden soll, muss eine komplett andere Framework wahl getroffen werden. So würde man, wenn man entweder nur für IOS oder Android entwickeln möchte, zu einer der nativen Lösungen greifen. Je nach Anforderungen kann auch, wenn beide Betriebssysteme bedient werden sollen, zu der nativen Lösung gegriffen werden. In dem Fall muss der komplette Code natürlich doppelt geschrieben werden. Daher wird normalerweise auf Frameworks zurück gegriffen welche beide Betriebssysteme bedienen. Einige bekannte Frameworks sind zum Beispiel 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 Framework Wahl kann von ca. 80-100 Prozent von dem kompletten Code für beide Betriebssysteme verwendet werden. Dafür sind Cross-Plattform Frameworks oft nicht so performant wie die nativen. Dies fällt besonders bei rechenaufwändigen Apps und Spielen ins Gewicht. Bei so einer leichten App wie DeepRain fällt dieser Nachteil nicht ins Gewicht. Außerdem hätten wir auch nicht genug Kapazitäten um zwei native und voneinander unabhängige Apps zu entwickeln. 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. Das in Kombination mit eigenem Interesse an dem Framework, ist der Grund dafür, dass die DeepRain App mit Flutter entwickelt wurde.

Abbildung 20: Entwicklung der Suchanfragen für den Begriff 'Flutter'

5.5.4 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 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 einzelene 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.

Die Grundlage von Flutter sind Widgets.
Wie funktioniert flutter?
Was ist der Unterschied zu anderen hybriden Frameworks?
Nur eine Codebasis
Keine weiteren Frameworks nötig
Alles dabei, UI, Widgets, Animationen

5.5.5 Projektstruktur

Welche Files gibt es?
Welche Klassen gibt es?
Wie arbeitet was zusammen?

5.6 Cloudfunktionen

Was ist eine Cloudfunktion?
Was macht die Cloudfunktion?
Wie funktioniert eine Cloudfunktion?

5.6.1 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 30 Minuten vor dem bevorstehenden Regen warnen lassen. Um eine Push Benachrichtigung zu versenden speichert der Server ein Dokument in der Firebase welches alle für die Push Benachrichtigung relevanten Informationen enthält. Dazu gehört zum Beispiel Der Titel, der Text und wie lange es noch bis zu dem Regen braucht. Es wurde eine Cloud Funktion in der Firebase hinterlegt, welche darauf reagiert, dass ein neues Dokument vom Server gespeichert wurde. Daraufhin liest die Cloud Funktion aus gesonderten Kollektionen die Device Tokens aus, für welche die Push Benachrichtigung gedacht ist. Dafür wurden verschiedene Kollektionen angelegt in welchen die Device Tokens von jedem Gerät gespeichert wurden. Je nach dem zu welchem Zeitpunkt ein User die Regenwarnung erhalten möchte, wird sein Device Token in eine andere Kollektion gespeichert. Die Cloud Funktion liest also aus dem Dokument vom Server, welches alle Informationen zu der Push Benachrichtigung enthält, aus, wie langes es noch bis zu Regen dauert. Daraufhin liest es aus der Kollektion für den jeweiligen Zeitpunkt alle Device Tokens aus die nun benachrichtigt werden müssen und schickt die Push Benachrichtigung raus. Wenn der Benutzer den Zeitpunkt zu dem er benachrichtigt werden möchte verändert, wird sein Device Token aus der einen Kollektion gelöscht, und zu einer anderen hinzugefügt.

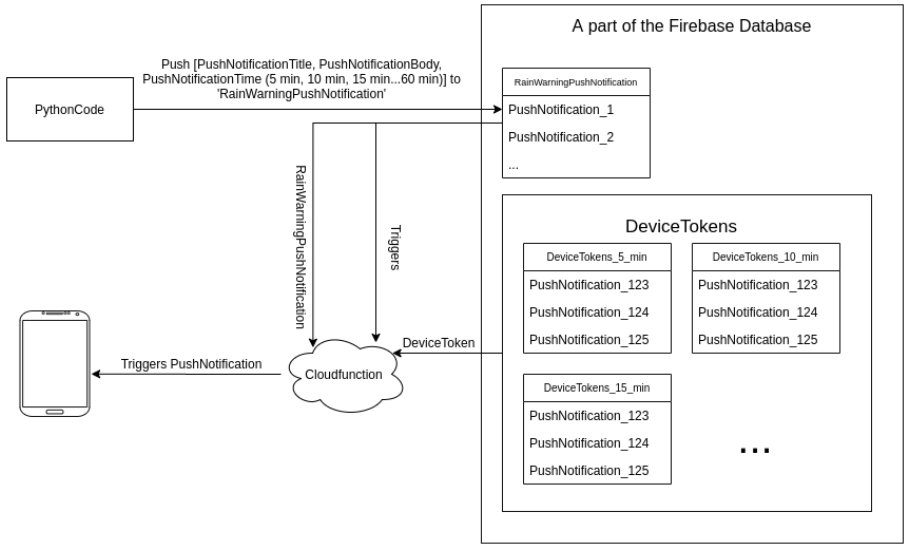


Abbildung 21: Funktionsweise des Prozesses zum senden von Push - Benachrichtigungen.

5.6.2 Vorgehen bei Entwicklung

Da für die Entwicklung einer IOS App MacOS benötigt werden, wurde die komplette App unter Linux entwickelt und während dem Entwicklungsprozess nur auf

Android Geräten getestet. Als die Grundfunktionalitäten verfügbar waren wurde die App mithilfe einer VirtualBox auf der MacOS installiert wurde auf einem IPhone installiert.
Softwaretests
Welche Tests und warum?