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: 8. September 2020

Betreuer: Prof. Dr. Oliver Dürr

Zusammenfassung

2-3 Sätze über die Netzte. (Das reimt sich auch noch! :)) 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

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

In folgender Abbildung ist die komplette Datenpipeline von dem Server des DWD bis zur Darstellung der Vorhersagen in der App zu sehen.

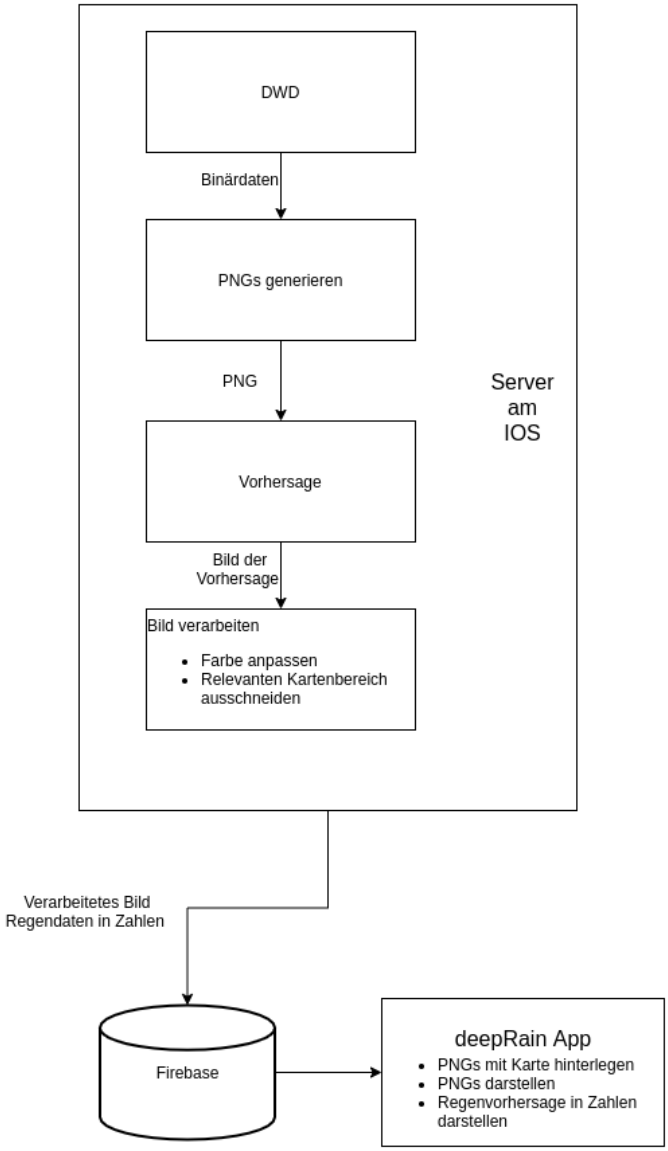


Abbildung 1: Das Gesamtsystem

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 “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.

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 mit den punktuellen Messungen von über 2000 Bodenniederschlagsstationen (https://www.dwd.de/DE/leistungen/radolan/radolan.html). Eine dieser Stationen befindet sich in Konstanz.

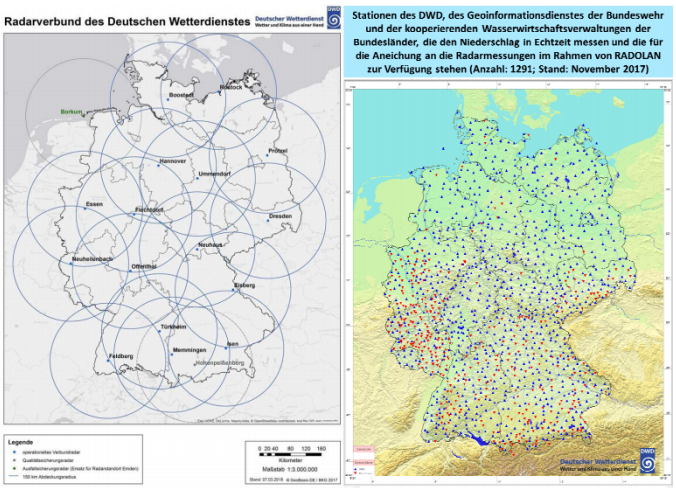


Abbildung 2: Übersicht über alle Boden und Radarstationen

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 Abbildung 3 ist der Aufbau der Binärdaten zu sehen.

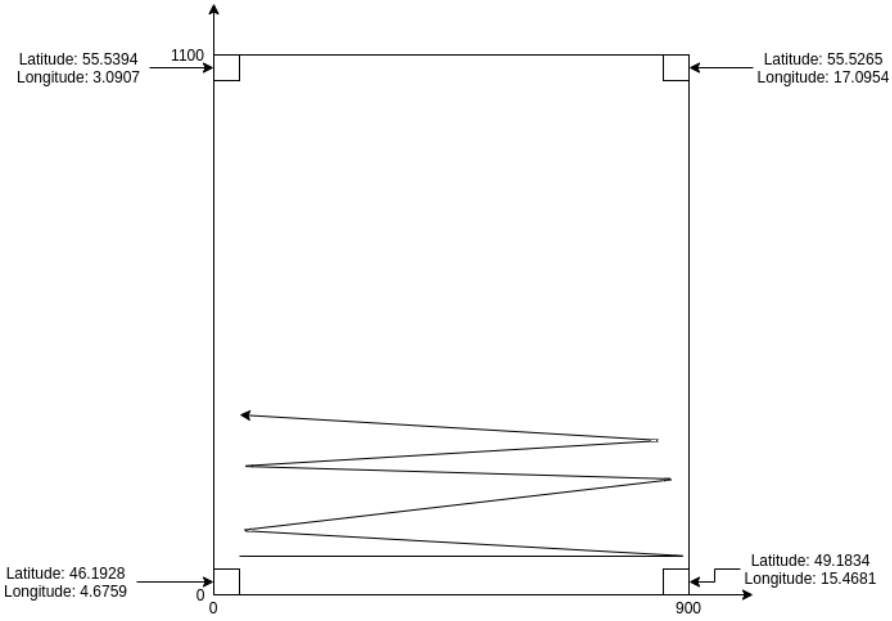


Abbildung 3: Der Aufbau des Koordinatensystems der Radar Daten

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.

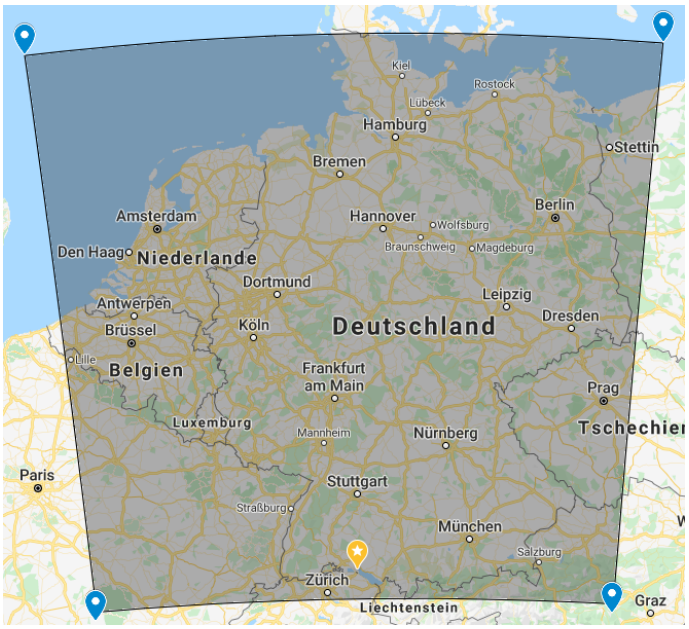


Abbildung 4: Die von den Daten abgedeckte Fläche auf einer 2D Karte

2.1 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.

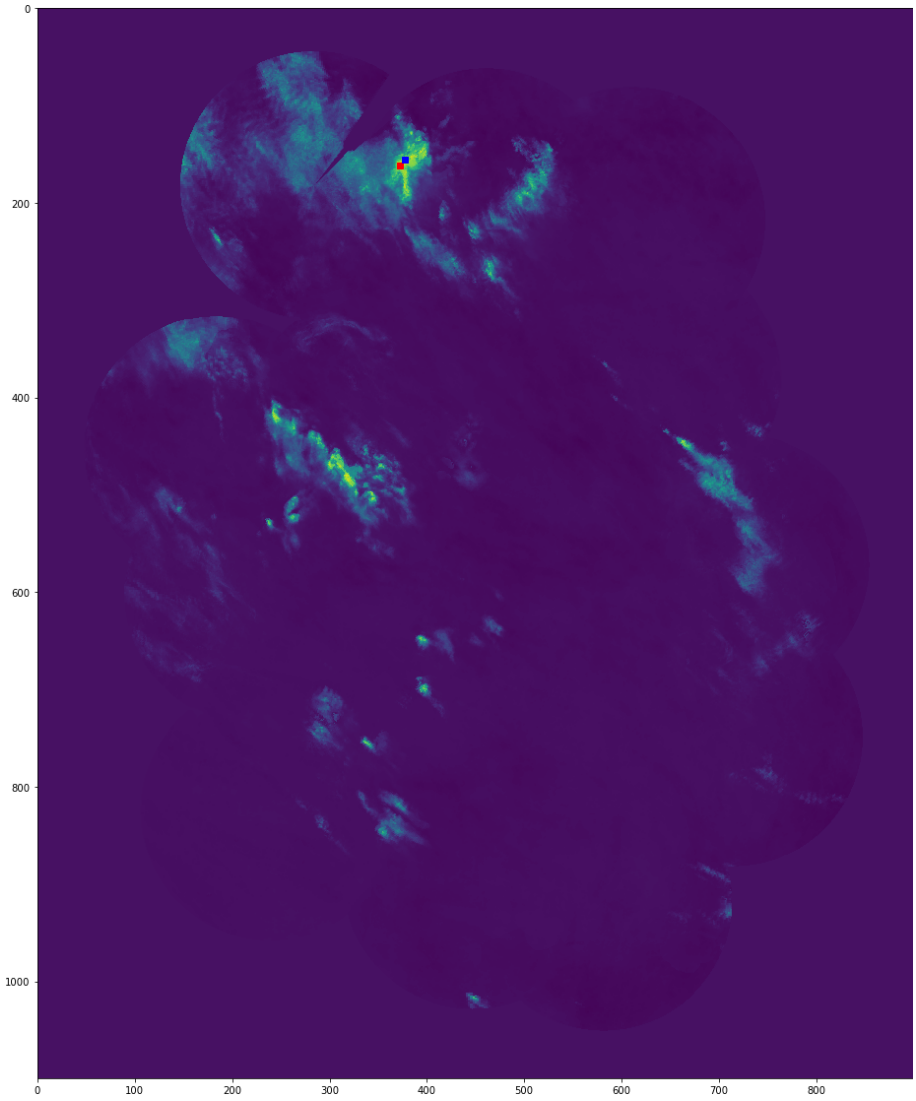


Abbildung 5: Korrelation der Regendaten in Konstanz mit jedem Pixel

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° an der vertikalen Achse gespiegelt.
Um sicherzugehen, dass die für die Vorhersage verwendeten Daten mit dem tatsächlichen Niederschlag in Konstanz übereinstimmen, wurden sie grafisch veranschaulicht. 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.

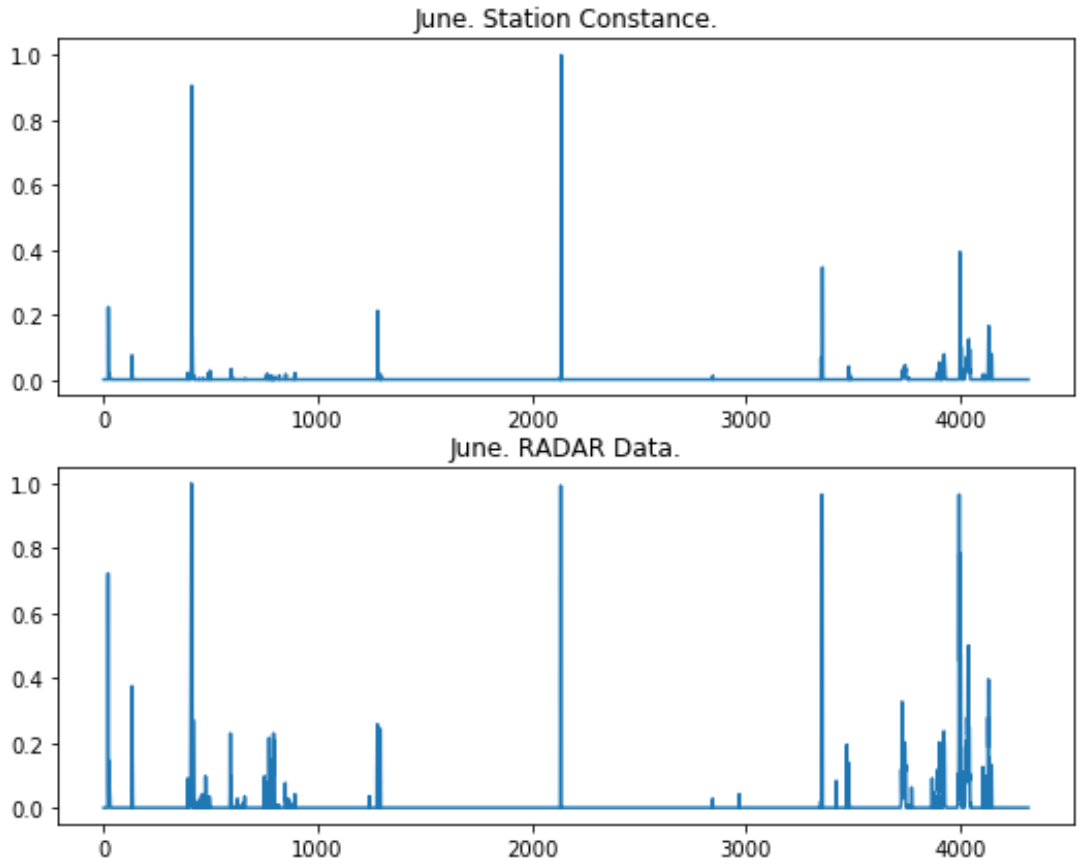


Abbildung 6: Die Radar und Stationsdaten für Juni im Vergleich

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 (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 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.

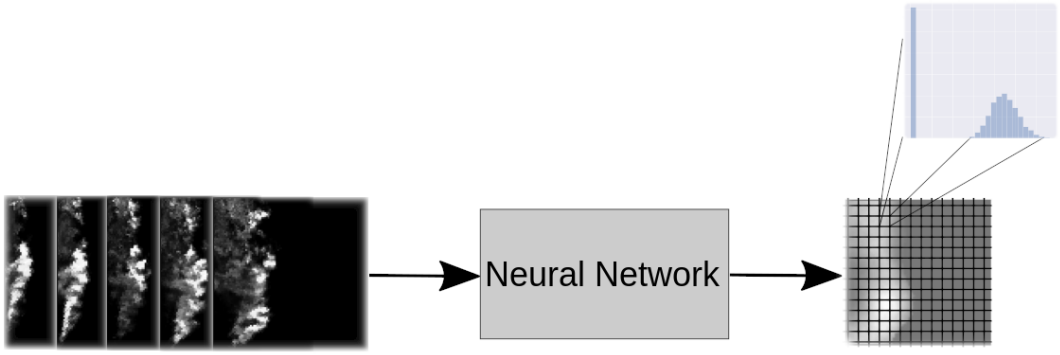


Abbildung 7: Skizzierung der Vorhersage

Wie aus Abbildung 7 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.

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.

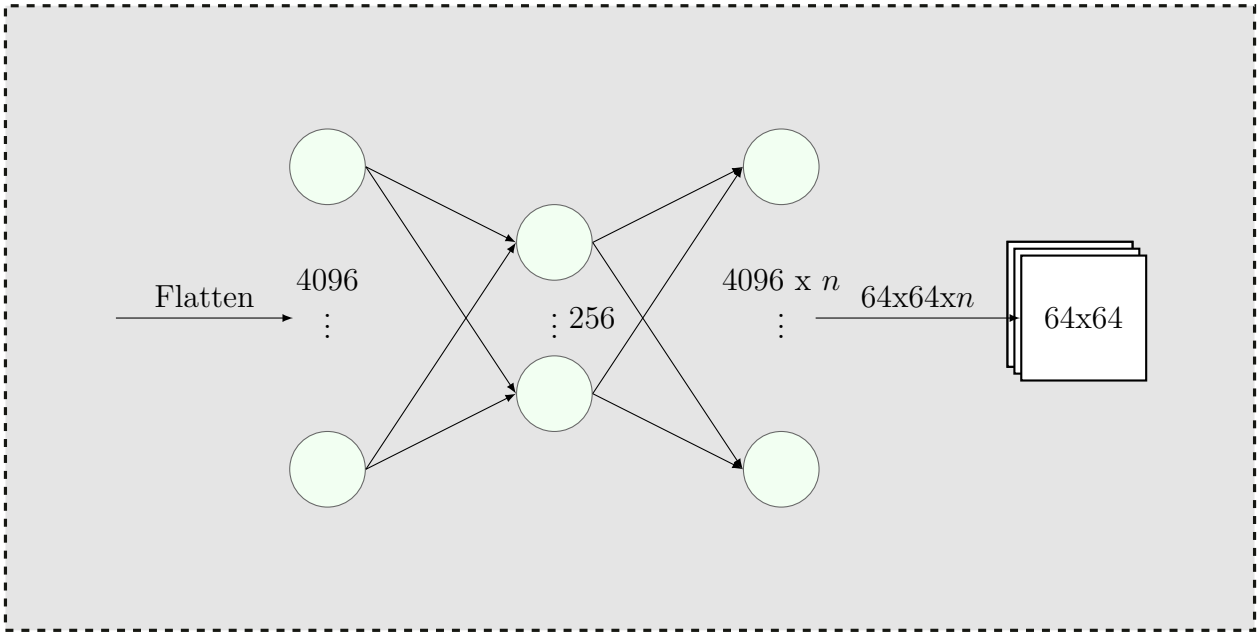


Abbildung 8: Klassifikationsschicht, n variiert hierbei je nach Verteilung. Für die Zero-Inflated Poisson Verteilung ist n gleich 2, für die Zero-Inflated Negativ Binomial Verteilung ist n gleich 3, und für die Multinomiale Verteilung ist n gleich 7

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

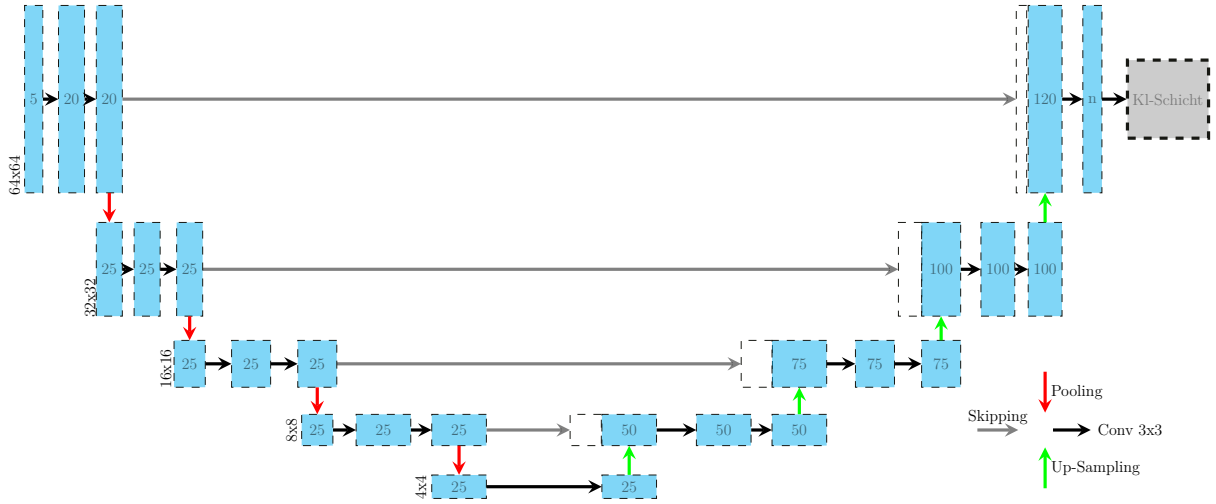


Abbildung 9: Abgespekte Version des Unets

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 9).

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.

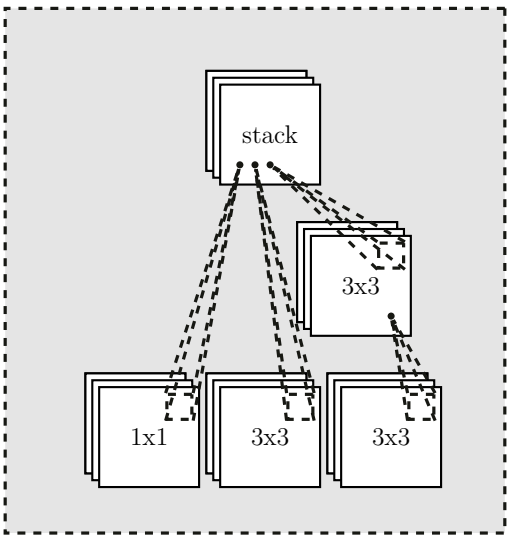


Abbildung 10: Aufbau der von uns verwendeten Inception Layer, angelehnt an die inception Layer im Paper

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.

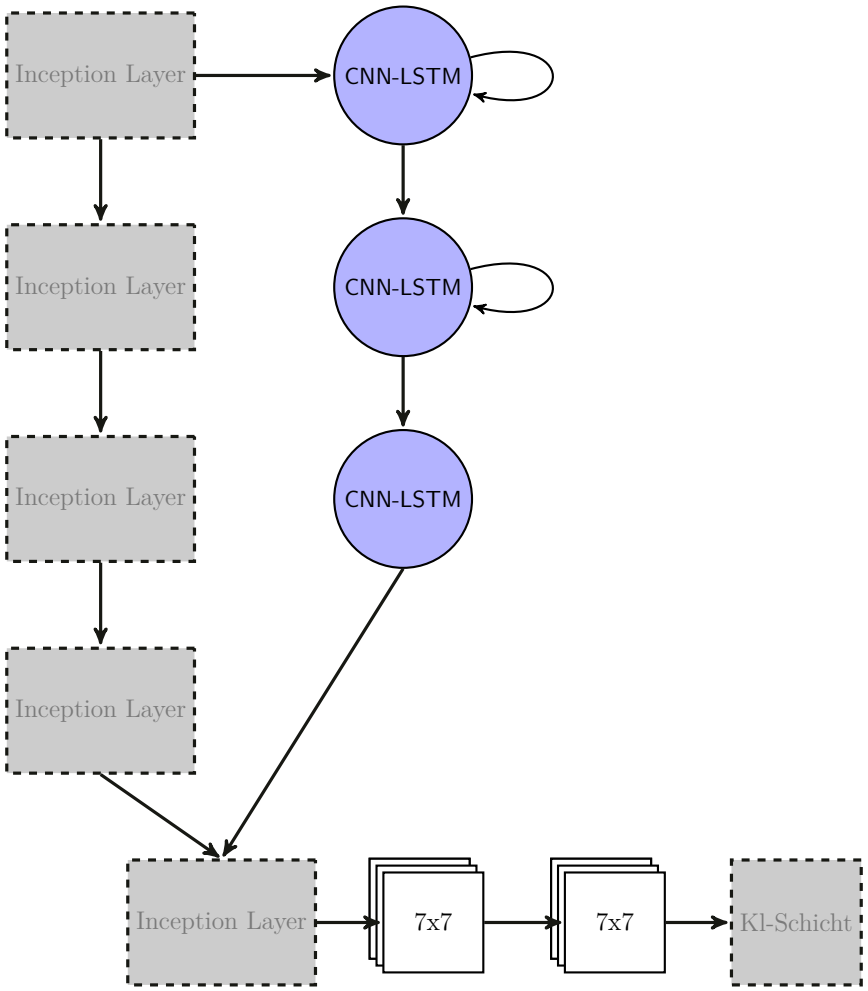


Abbildung 11: LSTM Architektur

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. 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.

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 ZINB.

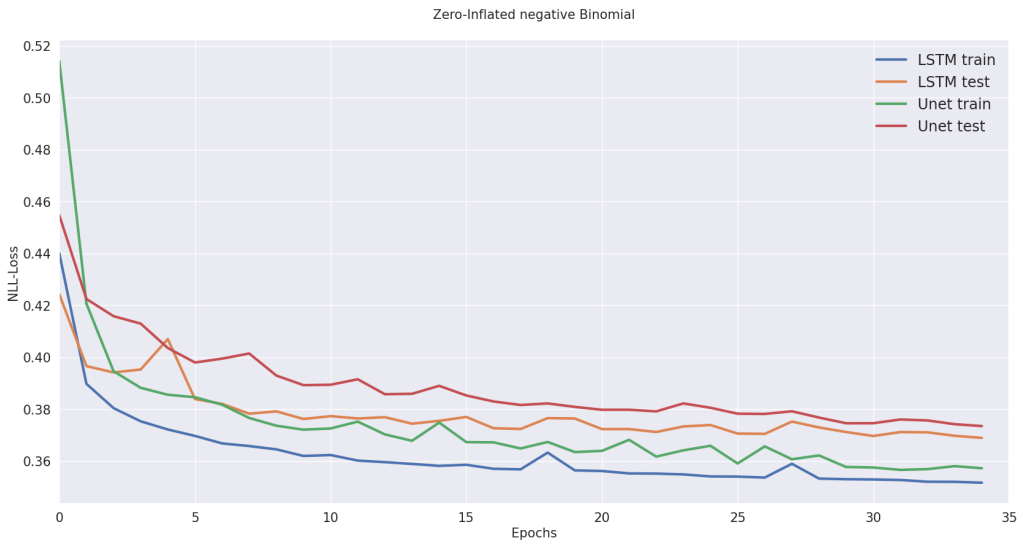


Abbildung 12: Trainingskurven für ZINB

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 ZINB 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).

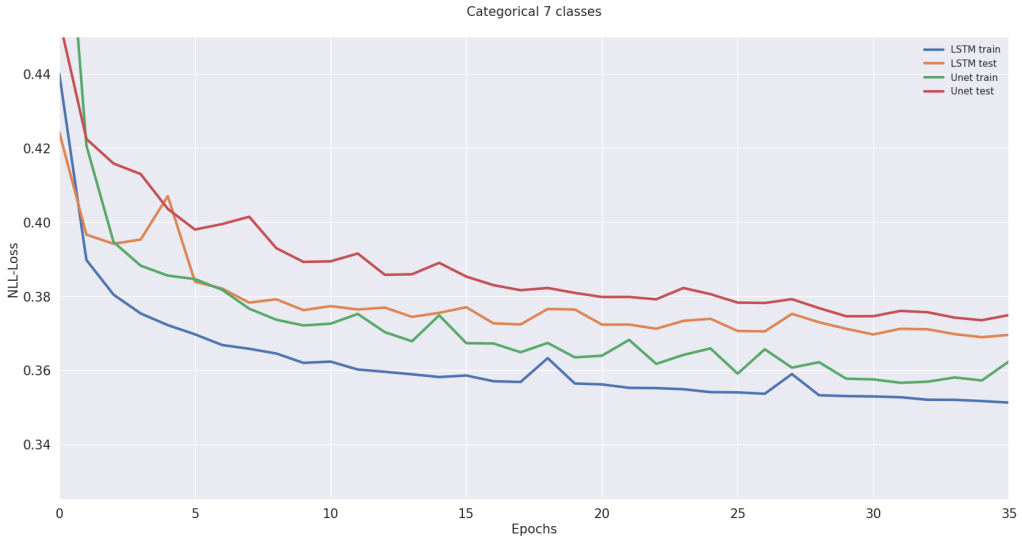


Abbildung 13: Multinomiale Verteilung

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 ZINB entspricht.

4.3 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 für die Regenvorhersage begutachten.

4.3.1 Auswertung der binären Regenvorhersage

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

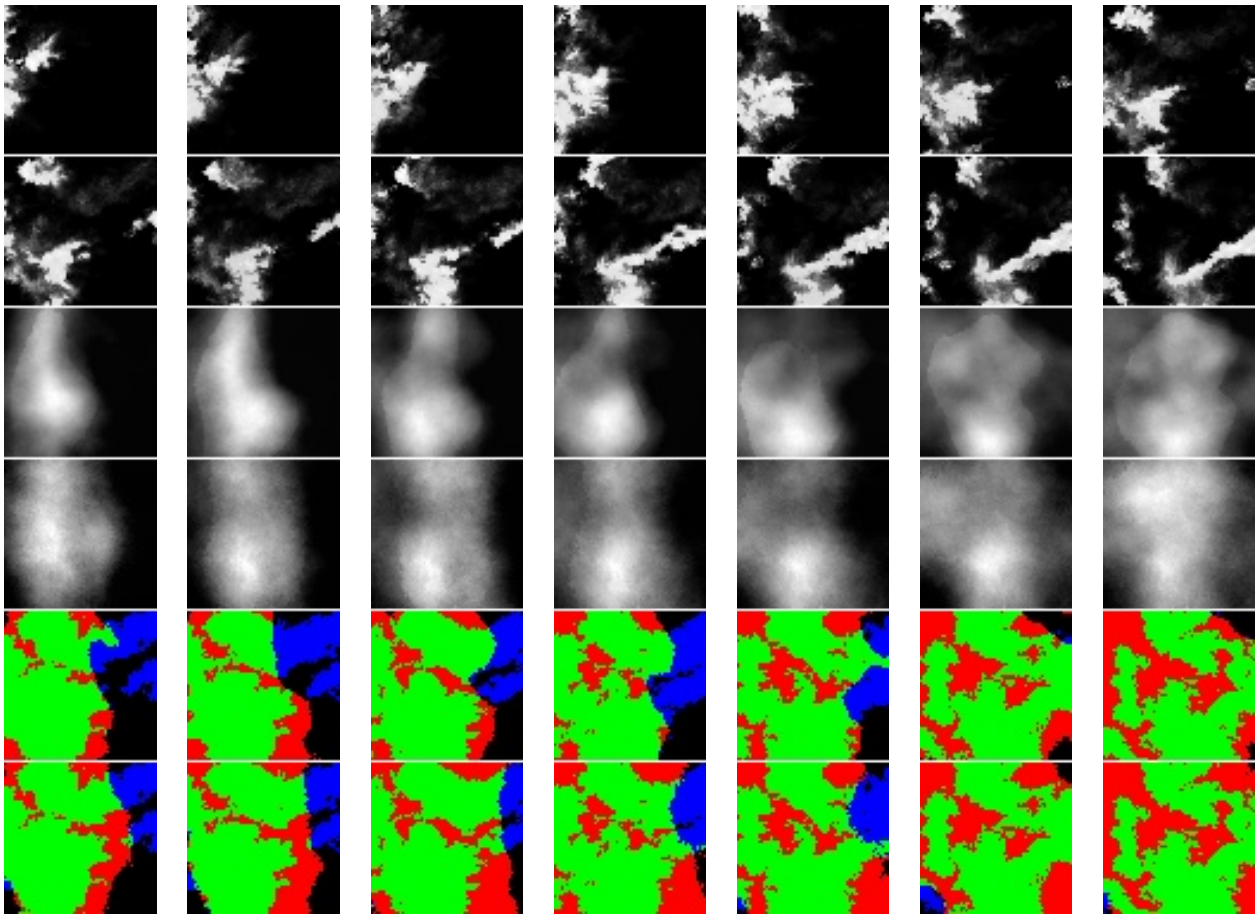


Abbildung 14: Vorhersage in Abständen von fünf Minuten für die ZINB. Die erste Zeile entspricht dem momentanen Regen. Zeile zwei entspricht dem tatsächlichen Regen in 30 Minuten. Zeile drei ist die 30 Minuten Vorhersage des Unets. Zeile vier ist die 30 Minuten Vorhersage der LSTM-Architektur. Für die Vorhersage wurde der Mittelwert der Verteilung berechnet. Die Bilder sind kontrastmaximiert dargestellt. Die letzten beiden Zeilen stellen die korrekte bzw. inkorrekte Regenvorhersage für die Unet- 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 14 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.

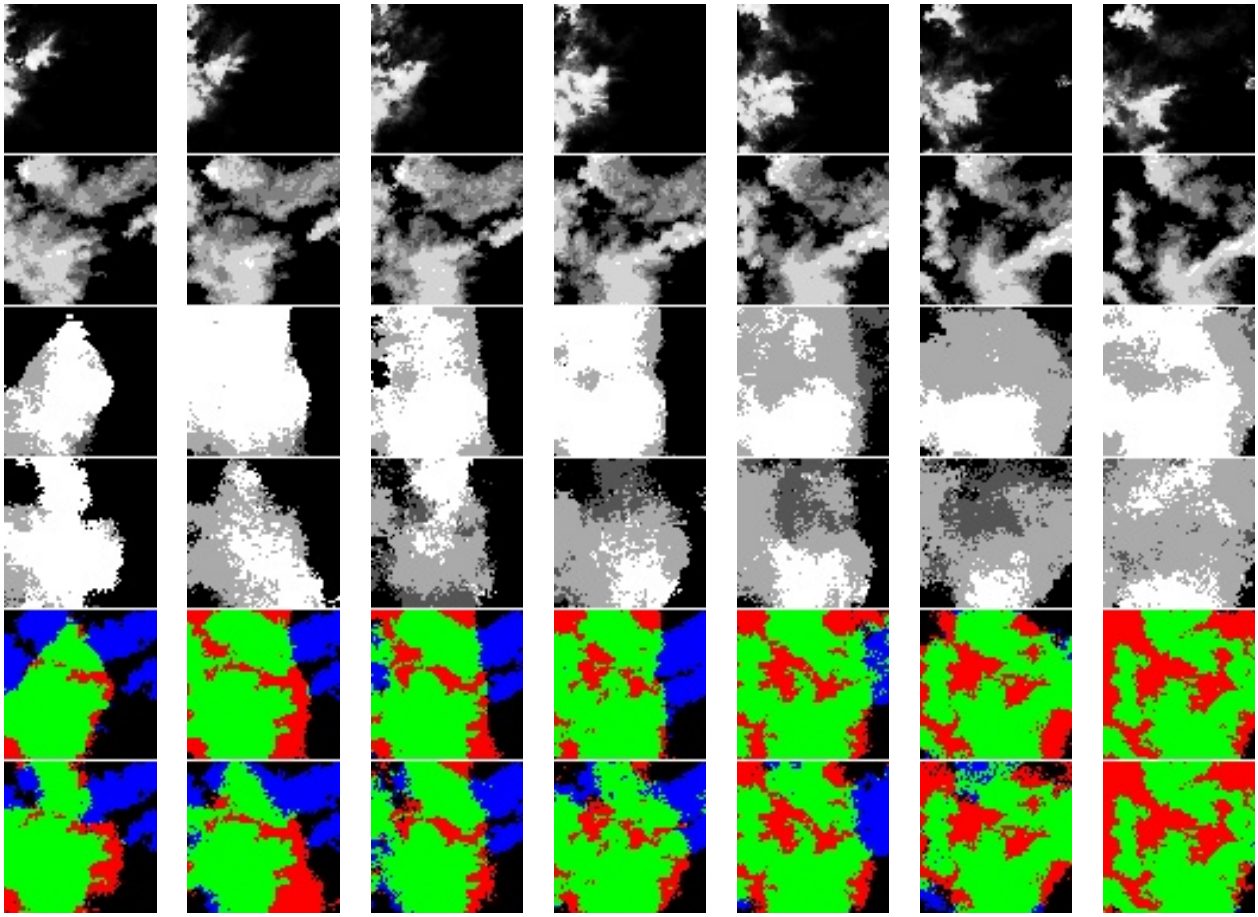


Abbildung 15: Vorhersage in Abständen von fünf Minuten für die Multinomialeverteilung. Hier sind die Zeilen wie in 14 dargestellt. Alle Bilder bis auf die Bilder der ersten Zeile sind in sieben Klassen dargestellt.

Auch bei der Multinomialenverteilung wird tendenziell zu viel Regen vorhergesagt. Für die Vorhersage wird das Label mit der größten 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 wir 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.

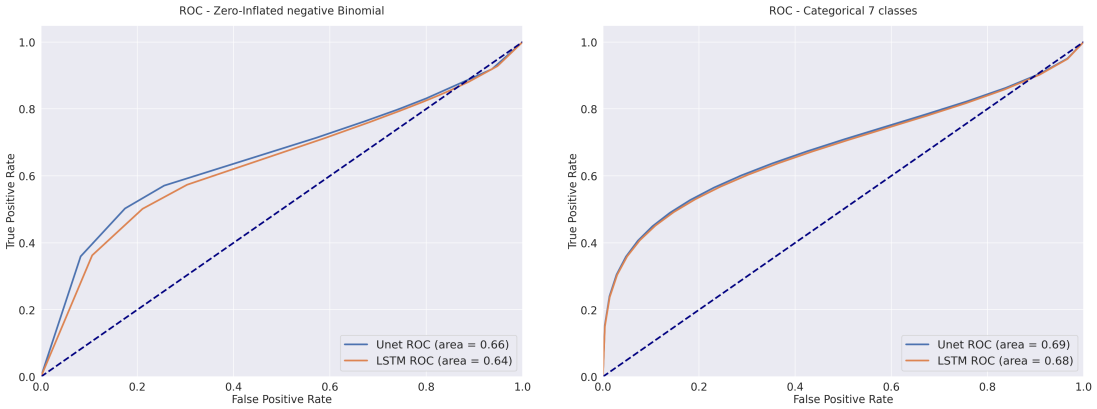


Abbildung 16: ROC/AUR für beide Architekturen und beide Verteilungen. Links für die ZINB und rechts für die Multinomialeverteilung. Die Auserwertung erfolgt für 20 verschiedene Schwellwerte.

Wie Anhand der Kurven zu erkennen ist, ist die ROC für die Multinomialeverteilung gleichmäßiger als die für die ZINB. Auch die Fläche unter der Kurve (”area under curve” **AUC**) ist für die Multinomialeverteilung marginal größer. Interessanterweise ist die Performance des Unets 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.

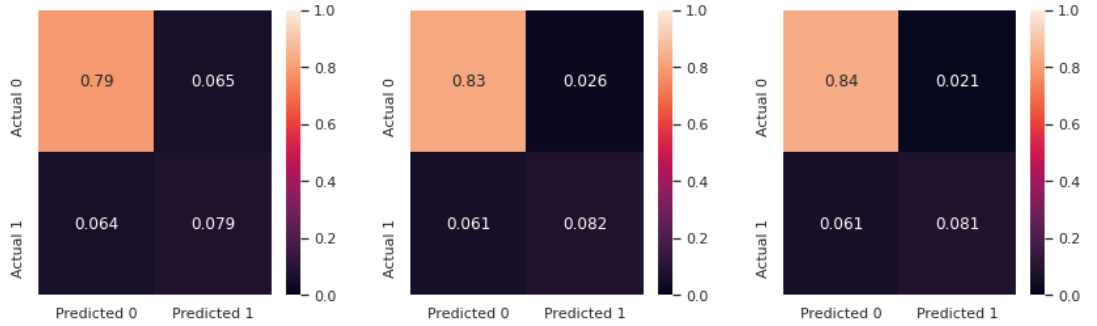


Abbildung 17: Confusionmatrix für die ZINB und dem Schwellwert 0.5. Links die einfache Baseline, mittig für die LSTM-Architektur und rechts für die Unet-Architektur.

Anhand der Confusionmatrix ist zu sehen, dass die Netzwerke in beiden Fällen eine

bessere Vorhersage liefern als die einfache Baseline. Anhand der Confusionmatrix wird das ungleichgewicht zwischen Regen und kein Regen deutlich.

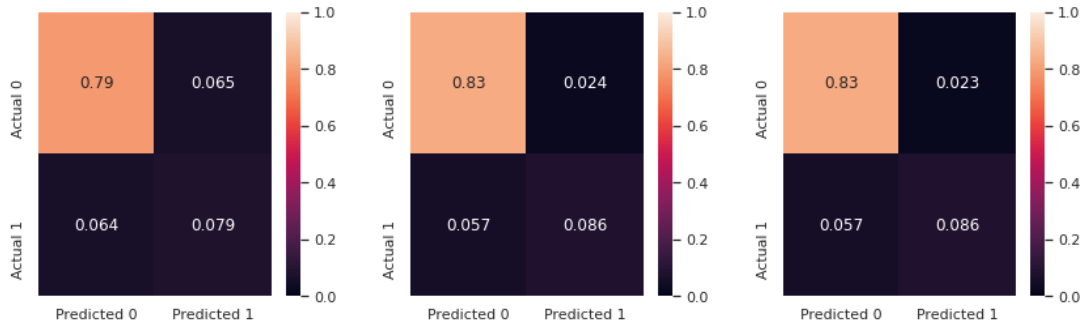


Abbildung 18: Confusionmatrix für die Multinomialeverteilung und dem Schwellwert 0.5. Links die einfache Baseline, mittig für die LSTM-Architektur und rechts für die Unet-Architektur.

Auch für die Multinomialeverteilung 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 Multinomialeverteilung etwas besser als für die ZINB. Letztere (insbesondere die Unet-Variante) produziert mehr true negative und weniger false positiv Vorhersagen als die Multinomialeverteilung.

Für die tatsächliche Regenvorhersage ist jedoch interessant, in wie vielen Fällen eine Person Nass wird obwohl kein Regen vorhergesagt wird. Dazu wird die untere Zeile der Confusionmatrix mit einem Faktor multipliziert, sodass deren Summe 100 ergibt.

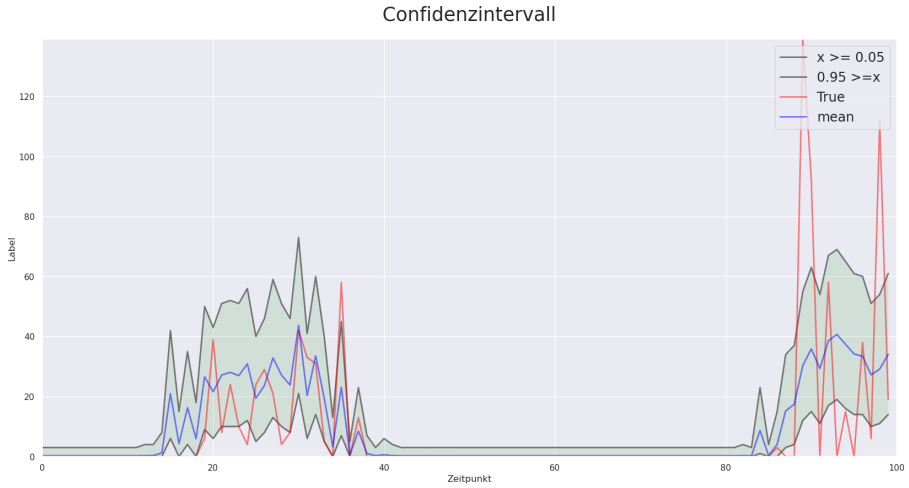
Verteilung	Person wird Nass	Person wird nicht Nass
Multinomialeverteilung	**39,9** %	**60,1** %
ZINB	43,0 %	57,0 %
Einfache Baseline	44,8 %	55,2 %

Tabelle 1: Wahrscheinlichkeit trocken zu bleiben

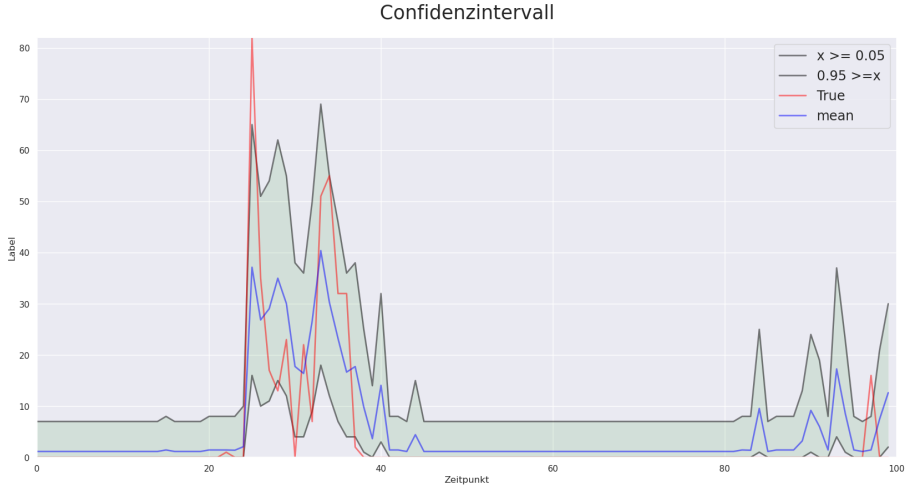
Aus der Tabelle 1 hervorgeht, wird die Person die sich auf die Vorhersage verlässt mit einer Wahrscheinlichkeit von 39,1 (im besten Fall) % Nass. Dies ist zwar nicht überragend, es ist jedoch besser als würde sich die Person darauf verlassen, dass das Wetter so bleibt wie es ist.

4.3.2 Auswertung der Regenvorhersage

Ein weiterer interessanter Aspekt ist die Genauigkeit der Vorhersage für den Regenfall. Hierfür betrachten wir zuerst die ZINB. 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

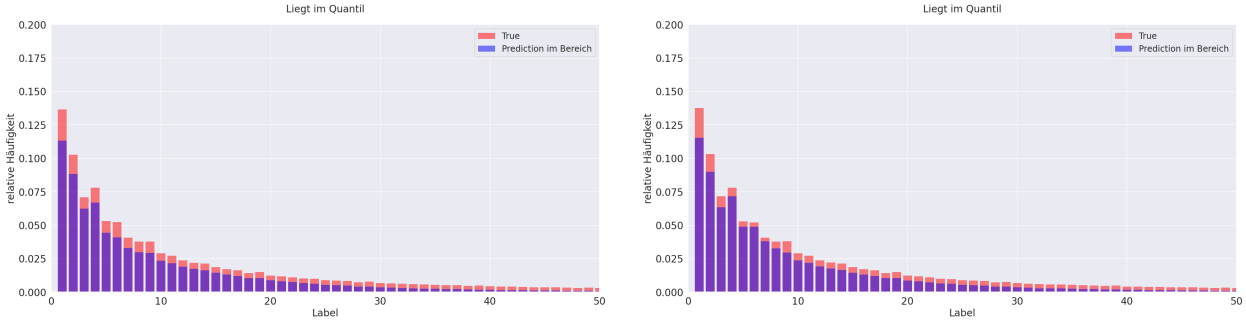


(b) UNET

Abbildung 19: 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 20: Relative Anzahl der ground truth im 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 hingegen, dass der ”wahre” Wert außerhalb der Konfidenzintervalle liegt. Das bedeutet, dass für die extremen Werte (die auch seltener vorkommen) die Vorhersage ungenauer wird.

Architektur	inklusive Regen	exklusive Regen	Konfidenzintervall
UNET	**93,0** %	**73,0** %	97,5 %
	88,0 %	43,0 %	95 %
	86,0 %	35,0 %	90 %
LSTM	92,0 %	71,0 %	97,5 %
	89,0 %	**45,0** %	95 %
	87,0 %	**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 Konvidenzintervall. Für die Spalte ”inklusive Regen” ist der Unterschied nur marginal. Für die Spalte ”exklusive Regen” ist ein großer Sprung zwischen dem 97,5 % und dem 95 % Konvidenzintervall 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 für diese Fälle eine schlechte Schätzung für die ground truth.

Schauen wir uns nun die Multinomialeverteilung an. Wir gehen ähnlich vor wie bei der ZINB vor und stellen die Anzahl der korrekten Vorhersagen gegenüber der Anzahl der ground truth (pro label). Um dies mit der ZINB vergleichen zu können, skalieren wir die Vorhersage der ZINB logarithmisch und stellen dies ebenfalls als Histogramm dar. Die folgende Abbildung zeigt das Ergebnis dieser Vorgehensweise.

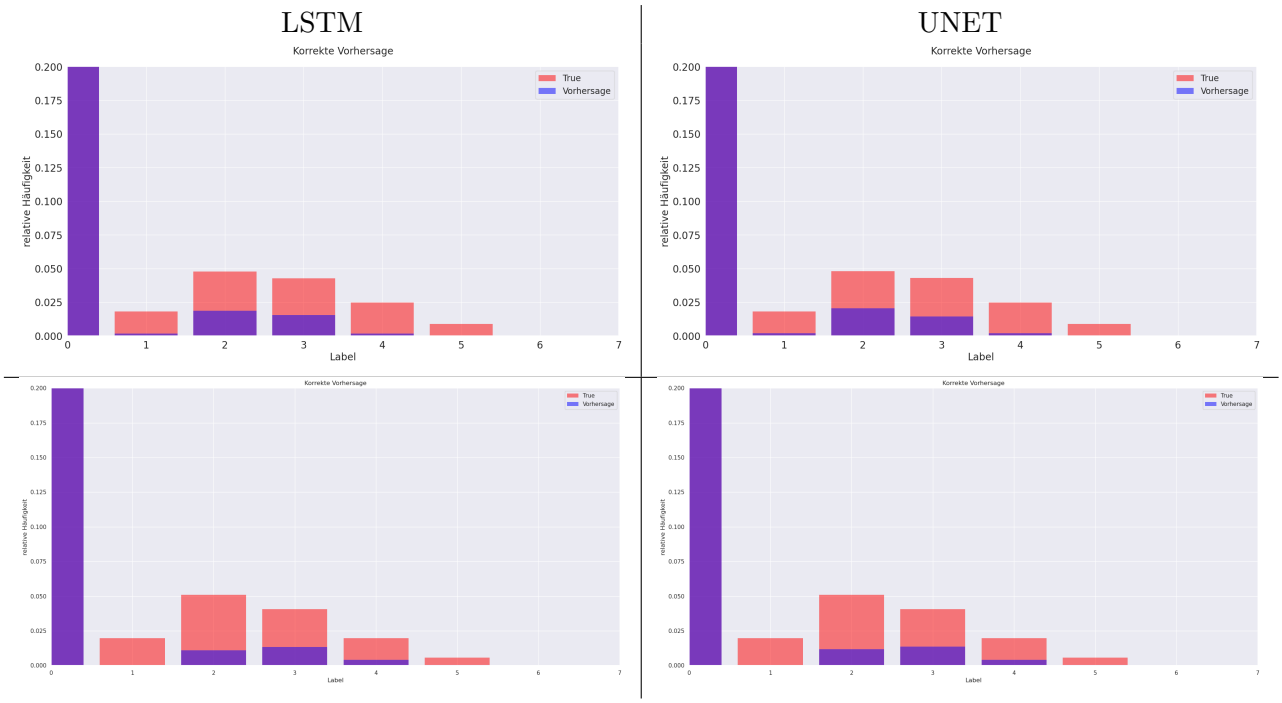


Abbildung 21: Histogramm der Vorhersagen für beide Verteilungen und Architekturen. Die erste Zeile entspricht der ZINB, die letzte der Multinomialverteilung

Anhand des Histogramms ist zu erkennen, dass die ZINB eine leicht bessere Vorhersage liefert. Das Histogramm wurde in Richtung y-achse bei 0.2 begrenzt um eine bessere Darstellung zu erreichen. Stellt man die Beziehung in einer Tabelle dar, so kommt man zu folgendem Ergebnis.

	exklusive Regen		inklusive Regen	
	LSTM	UNET	LSTM	UNET %
ZINB	26 %	27 %	78 %	78 %
Multinom	21 %	22 %	88 %	88 %

Tabelle 3: Relative Anzahl der korrekt vorhergesagten Regenwerte

Anhand der Tabelle 3 ist ersichtlich, dass die ZINB eine bessere Vorhersage für die Regenstärke liefert. Die Multinomialeverteilung hingegen liefert eine bessere Vorhersage im allgemeinen.

4.3.3 Fazit der Auswertung

Für die binäre Regenvorhersage ist Multinomialverteilung die Verteilung welche die besten Resultate liefert. Laut Tabelle 1 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 Unet-Architektur ist(siehe Abbildung 13), bevorzugen wir hier die

LSTM-Architektur. Für die Klassifikation der Regenvorhersage liefert die Verteilung der ZINB die besten Resultate (siehe Tabelle 3). Hier ist laut Tabelle 3 die Unet-Architektur marginal besser. Jedoch ist der Loss der LSTM-Architektur besser (siehe Abbildung 12). Zudem liegt laut Tabelle 2 die ground truth für kleinere Konfidenzintervalle öfters in eben diesen Intervallen. Daraus schließen wir, dass der Mittelwert näher an der ground truth liegt und dieser dadurch eine bessere Vorhersage liefert. 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.

5 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. Dazu gehört die App selbst, die Datenbank und die Serverkomponenten, welche für das Veröffentlichen der Vorhersagen verantwortlich ist.

5.1 Übersicht

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

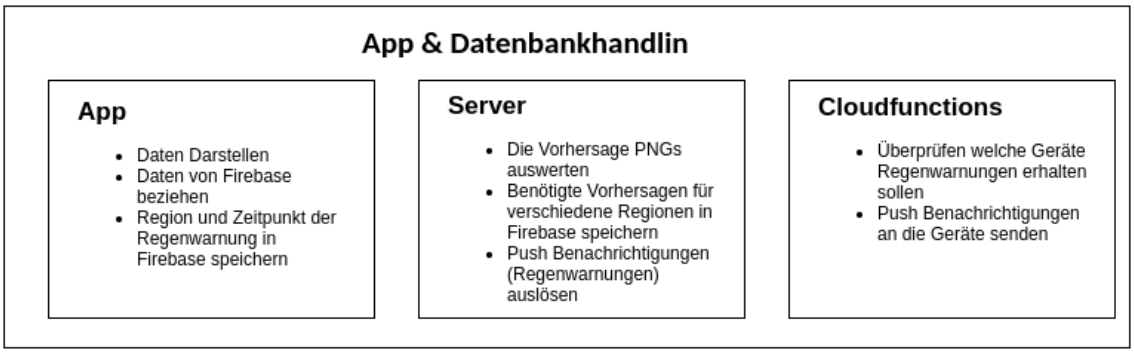


Abbildung 22: Die Komponente App und Datenbankhandling

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. 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 das Senden der Pushbenachrichtigungen, welche für die Regenwarnungen gebraucht werden.

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 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 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 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.

5.2.1 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 23 ist der Datenbankaufbau zu sehen.

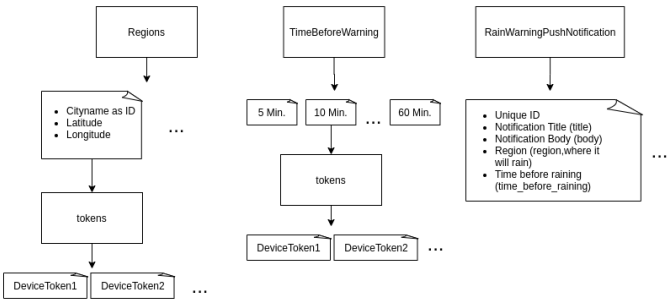


Abbildung 23: Der Aufbau der Kollektionen und Dokumente in der Firebase

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 5.5.1 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 5.5.1) 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.

5.3 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 Pushbenachrichtigungen. 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 berechnet haben, werden die daraus resultierenden Vorhersagebilder 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 Grauwert 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.

5.4 Die App

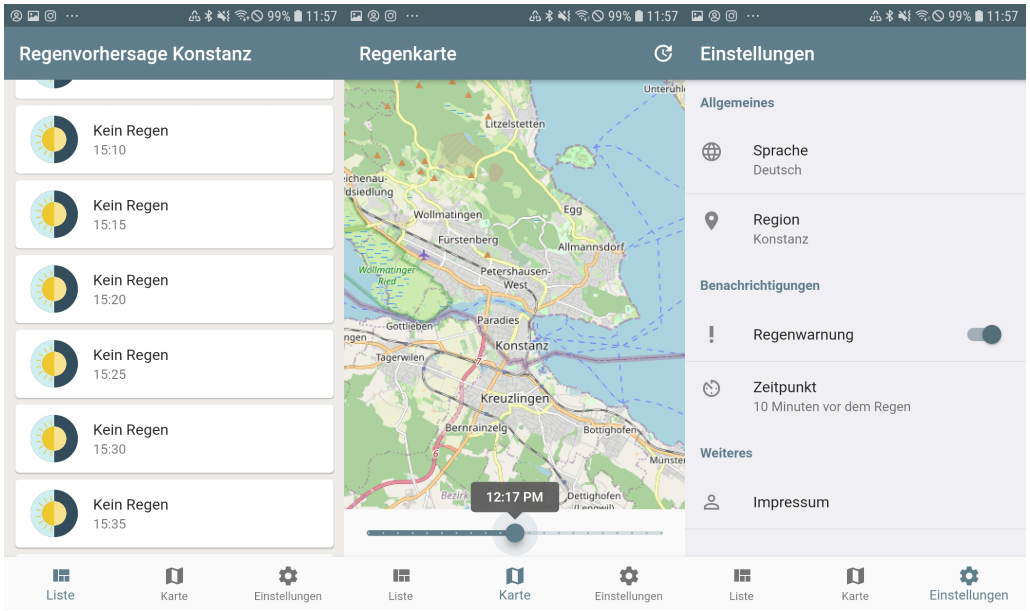
5.4.1 Funktionen der App

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.

5.4.2 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 24: 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 Benachrichtigungs Kategorie 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 25 ist der Datenfluss beim Anpassen des Zeitpunktes für die Regenwarnung dargestellt.

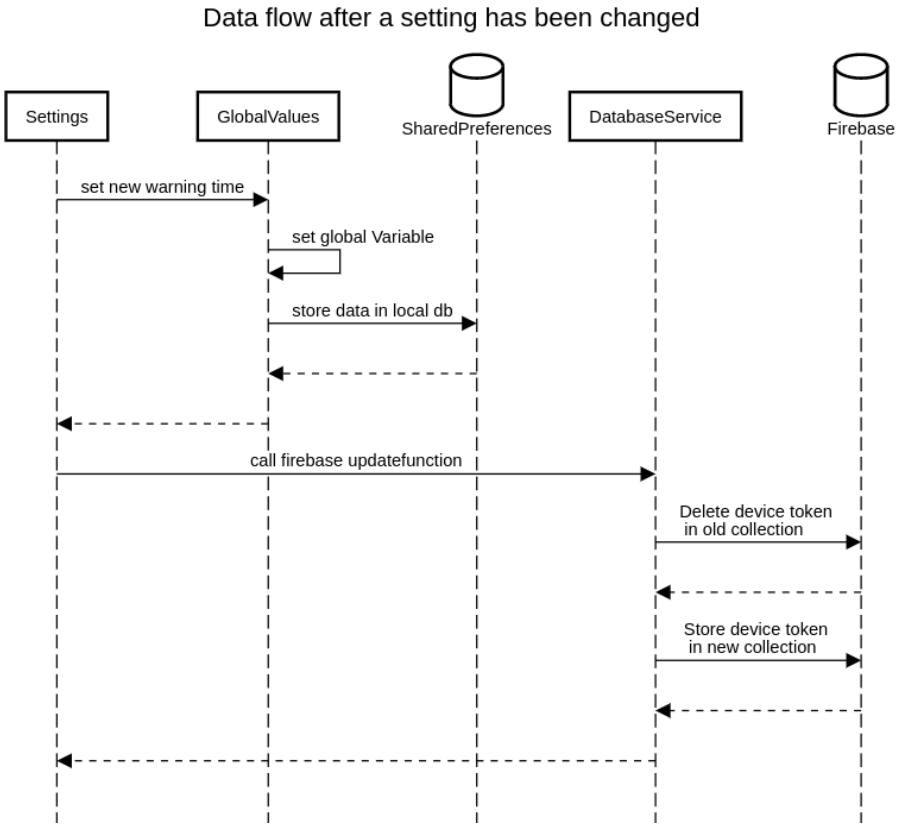


Abbildung 25: Der Datenfluss beim ändern des Zeitpunktes der Regenwarnung

5.4.3 Der Appstart

Während dem Appstart wird die App für die Verwendung vorbereitet, Einstellungen werden wieder hergestellt 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.

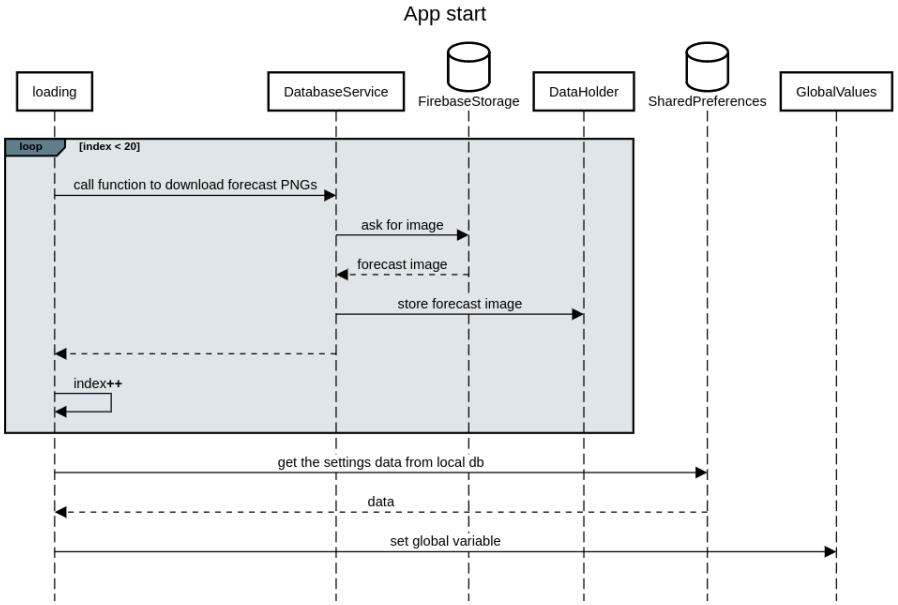


Abbildung 26: Datenfluss beim starten der App inklusive der Datenbankabfragen (Server und Lokale Datenbank)

5.4.4 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 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 5.4.5 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. Aufgrund der verhältnismäßig hohen Datenmenge der 20 Vorhersage PNGs, werden aktuell nur beim App Start oder durch manuelles Auslösen eines Updates die Vorhersage Daten aktualisiert.

5.4.5 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 27: 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 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. Dieses Verfahren ist aber sehr Zeitaufwändig und daher nicht geeignet um es während der Laufzeit der App anzuwenden.
Der erste Ansatz basierte auf dem Teile und Herrsche Prinzip. Dabei wurde das vorhersage PNG in 4 gleichgroße teile aufgeteilt und überprüft in welchem Quadrant sich das Gerät befindet. Alle Quadranten, in denen sich das Gerät nicht befindet, wurden verworfen. Der übrige Quadrant wurde erneut in vier teile aufgeteilt. Dieser Vorgang wurde so oft wiederholt, bis nur noch ein Pixel übrig blieb. Wie bereits in 2 beschrieben, ist der abgedeckte Kartenbereich nicht rechteckig. Die daraus resultierenden nichtlinearität verursachte 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 Unterschied 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.

5.4.6 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. 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 doppelt geschrieben werden. Daher wird normalerweise auf Frameworks zurückgegriffen, 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 Frameworkwahl 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 rechenaufwendigen 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 28: Entwicklung der Suchanfragen für den Begriff 'Flutter'

5.4.7 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 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.

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

5.4.8 Projektstruktur

Der gesamte Code der App ist in die fünf Ordner DataObjects, global, screens, services und Widgets aufgeteilt. Die erste verwendete Datei ist main.dart. In dieser wird festgelegt welcher Screen als erstes nach dem Start aufgerufen wird und die Bottom Navigation wird konfiguriert. Für jeden Screen der App gibt es eine .dart datei in dem Ordner Screens. Welche Files gibt es?
Welche Klassen gibt es?
Wie arbeitet was zusammen?

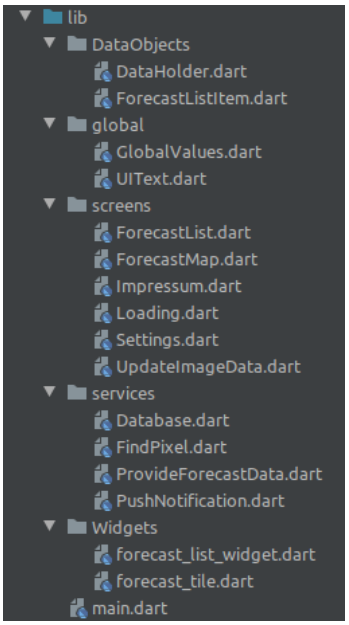


Abbildung 29: Die Projektstruktur der App

5.5 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.

5.5.1 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 Push-

benachrichtigung erhalten sollen.

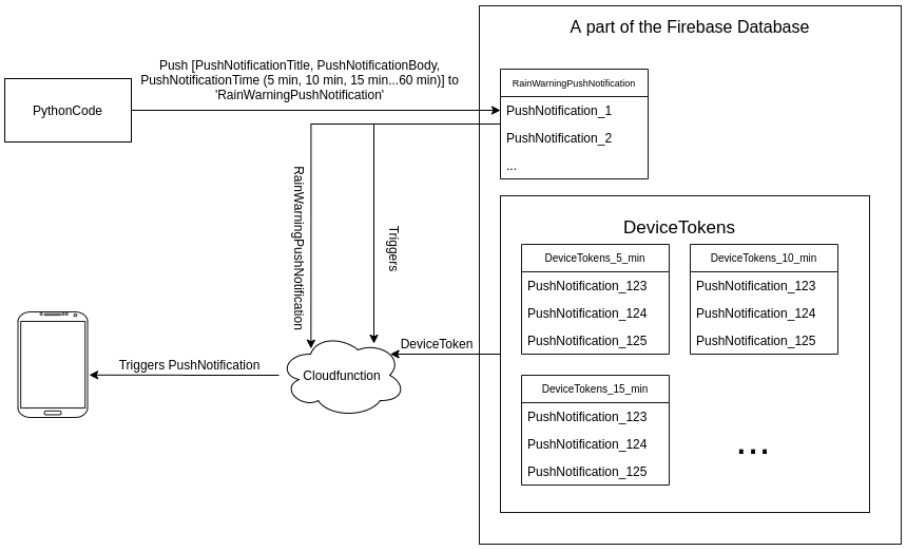


Abbildung 30: Funktionsweise des Prozesses zum senden von Push - Benachrichtigungen.

5.6 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 Linuxrechner 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.