Optimierung und Evaluierung eines automatischen Rehkitzdetektionsalgorithmus auf Basis eines Convolutional Neural Networks (CNN)

Bachelorarbeit

für die Prüfung zum

Bachelor of Engineering

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

Runa Römke

Abgabe am 11. September 2020

Bearbeitungszeitraum:	22.06.2020 – 13.09.2020
Matrikelnummer, Kurs:	5028315, TINF17ITIN
Abteilung:	Methodik der Fernerkundung - Experimentelle Verfahren
Ausbildungsbetrieb:	Deutsches Zentrum für Luft- und Raumfahrt (DLR) Oberpfaffenhofen
Betreuer des Ausbildungsbetriebs:	Dr. Martin Israel
Gutachter der Dualen Hochschule:	Dr. Holger Gerhards

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem

Thema

Optimierung und Evaluierung eines automatischen Rehkitzdetektionsalgorithmus auf Basis eines Convolutional Neural Networks (CNN)

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* falls beide Fassungen gefordert sind

Oberpfaffenhofen, den 11. September 2020

Kurzfassung

Die Absicht der Untersuchungen dieser Arbeit ist die Nutzung eines neuronalen Netzes für die maschinelle Rehkitzdetektion auf Thermalbildern. Entwickelt wird ein Convolutional Neural Network für eine Klassifizierung der Bilder als Tier oder kein-Tier, welches im Wildretter-Projekt eingesetzt werden soll.
Die verwendeten Daten zeigen eine Unausgewogenheit, da mehr Bilder ohne Tier als Bilder mit Tier vorliegen. Das neuronale Netz lässt außerdem eine Überanpassung erkennen, da die Evaluierung auf Basis der Metriken Recall und Precision mit Trainingsdaten deutlich bessere Ergebnisse liefert als die Evaluierung mit Testdaten. Das Entfernen von nicht-Tier-Patches aus der Trainingsdatenmenge wird als Methode gewählt, mit Unausgewogenheit umzugehen. Gegen die Überanpassung kommt die Dropout-Methode zum Einsatz.
Das neuronale Netz kann einen Recall von 0,90 und eine Precision von 0,69 erreichen. Dies ist im Vergleich mit der visuellen Detektion ein höherer Recall und eine geringere Precision.

Abstract

The intention of the research of this thesis is to use a neural network for automatic fawn detection on thermal images. A Convolutional Neural Network is being developed to classify the images as animal or non-animal and will be used in the Wildretter-Project.
The used data show an imbalance, because there are more images without animals than images with animals. The neural network also shows an overfitting, since evaluation based on the metrics recall and precision with training data gives significantly better results than evaluation with test data.
Removing non-animal patches from the training data set is chosen as a method to deal with imbalance. The dropout method is used to deal with overfitting.
The neural network can achieve a recall of 0.90 and a precision of 0.69. This is a higher recall and a lower precision compared to visual detection.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CNN Convolutional Neural Network

CPU Central Processing Unit

CRISP-DM CRoss Industry Standard Process for Data Mining

EDA Exploratory Data Analysis

GPU Graphics Processing Unit

IMF Institut zur Methodik der Fernerkundung

UAV Unmanned Aerial Vehicle

ReLU Rectified Linear Unit

RUS Random Under Sampling

ROS Random Over Sampling

Abbildungsverzeichnis

1. Einleitung

1.1. Einführung in das Thema

Mit der Entwicklung von unbemannten Flugkörpern (Unmanned Aerial Vehicle (UAV)) innerhalb des letzten Jahrhunderts eröffneten sich viele verschiedene Anwendungsgebiete für eine solche Technologie. Erste Entwicklungen unbemannter Flugkörper wurden bereits im ersten Weltkrieg vorangetrieben. Das Militär hatte damals das Bestreben, entfernte Ziele mit „fliegenden Bomben“ in Form von UAV zu treffen. Auch im zweiten Weltkrieg wurden unbemannte Flugkörper als Waffen eingesetzt. [22] [6, Kapitel 2]
Nach Weiterentwicklung ergaben sich Chancen, UAV nicht nur als Waffe sondern auch für Aufklärungsmissionen zu verwenden. Systeme wie beispielsweise die CL-89 Midge (1960er) waren in der Lage, eine definierte Strecke zu fliegen und Fotos zu machen. [6, Kapitel 2]
Die Technik von UAV entwickelt sich fortlaufend weiter und deren Nutzung ist nicht nur auf Anwendungsgebiete des Militärs beschränkt. Ein zukunftsträchtiges Aufgabengebiet ist die Forschung. NASA's Helios Prototype, der, betrieben von Hochleistungs-Solarzellen, bereits 2001 eine Flughöhe von 96863 Fuß erreichte, ist von Interesse für die Erdfernerkundung oder als Alternative zu Satelliten. [5]
Gleichzeitig entwickelte sich in den letzten Jahrzehnten die Technologie des maschinellen Lernens. Es bieten sich viele verschiedene Anwendungsmöglichkeiten wie beispielsweise die Erkennung von Straßenschildern [57] oder der Einsatz in der Medizin [56] [51].
Das Institut zur Methodik der Fernerkundung (IMF) setzt UAV auch für die Fernerkundung ein und wird hierbei von maschinellem Lernen unterstützt.
Auch das Wildretter-Projekt, ein Forschungsprojekt zum Schutz von Tieren, nutzt UAV, Rehkitze im hohen Gras oder hohen Getreide aufzuspüren, damit sie nicht ins Mähwerk geraten [7]. Hierfür werden Thermalbilder während des Fluges aufgenommen und ausgewertet. Damit die Auswertung der Bilder in Echtzeit während des Flugs stattfinden kann, kommt ein neuronales Netz zum Einsatz.

Diese Bachelorarbeit befasst sich mit der Generierung eines neuronalen Netzes für das Wildretter-Projekt.

1.2. Problemstellung

Der Inhalt dieser Bachelorarbeit baut auf den beiden Arbeiten von Martin Israel [20] und Michael Sorg [46] auf. Martin Israel stellt in seiner Dissertation die Entwicklung eines UAV-basierenden Systems für die maschinelle Rehkitzerkennung dar. Michael Sorg zeigt in seiner Bachelorarbeit die Ansätze auf, wie maschinelles Lernen zur maschinellen Erkennung von Rehkitzen auf Thermalbildern auf dem UAV-System integriert werden kann.
Im Rahmen dieser Bachelorarbeit sollen nun weitere Untersuchungen zur Automatisierung der Rehkitzdetektion in Form eines neuronalen Netzes erfolgen.
Die zur Verfügung stehende Flugdatenmenge hat sich seit den beiden Arbeiten von Martin Israel und Michael Sorg deutlich ausgeweitet. Die Menge an verfügbaren Daten ist im maschinellen Lernen für die Qualität des Ergebnisses essentiell. Für diese Bachelorarbeit sollen die neuen Daten zusätzlich zu den bereits vorhandenen Flugdaten zum Generieren eines neuronalen Netzes verwendet werden. Dies beinhaltet eine Aufbereitung der Daten in eine für das maschinelle Lernen verwendbare Form.
Bestandteil dieser Arbeit ist außerdem das Trainieren eines neuronalen Netzes. Dabei erfolgen Untersuchungen mit dem Ziel, eine Erhöhung der Genauigkeit der Klassifizierung hervorzubringen. Das Netz soll zuverlässig alle Rehkitze erkennen und dabei nicht zu viele Fundstellen fälschlicherweise als Rehkitz anzeigen.
Teil der Dissertation Martin Israels, der Vergleich der maschinellen Detektion mit der visuellen Detektion, wird in dieser Arbeit weitergeführt, um die Qualität des entwickelten neuronalen Netzes zu bemessen. Damit ein Vergleich möglich ist, soll ein Evaluierungstool generiert werden.
Diese aufgeführten Punkte sind in Kapitel 2 als Aufgabenstellung zusammengefasst.

1.3. Aufbau der Arbeit

Diese Arbeit ist in sieben Kapitel unterteilt. Das erste Kapitel ist die Einleitung, zu der auch diese Erläuterung zum Aufbau der Arbeit gehört.
Kapitel zwei stellt die in diesem Kapitel erläuterte Problemstellen in Form einer Aufgabenstellung dar.
Im dritten Kapitel werden die theoretischen Grundlagen dieser Arbeit erläutert. Dabei wird auf das maschinelle Lernen und auf neuronale Netze eingegangen. Dies beinhaltet die Architektur, das Training und die Evaluierung von neuronalen Netzen. Eine besondere Architektur, ein Convolutional Neural Network (CNN), wird genauer beschrieben. Probleme im maschinellen Lernen werden erläutert und Werkzeuge für maschinelles Lernen in der Programmiersprache Python vorgestellt.
Das vierte Kapitel präsentiert das Konzept für die Bearbeitung der Punkte der Aufgabenstellung.
Mit Kapitel fünf werden die Ergebnisse des vorgestellten Konzepts dargestellt.
Im Anschluss wird in Kapitel sechs die Arbeit zusammengefasst.
Kapitel sieben reflektiert diese Arbeit und gibt einen Ausblick.

2. Aufgabenstellung

Diese Bachelorarbeit soll folgende Punkte erfüllen:

1. Integration weiterer Datensätze
2. Trainieren des neuronales Netzes
    a) Verwendung von Python
    b) Erhöhung der Genauigkeit der Klassifizierung
    c) Verringerung der Fehldetektionsrate, ohne dabei Kitze zu übersehen
3. Evaluierung
    a) Es soll ein Evaluierungstool geschaffen werden, um die Qualität des neuronalen Netzes zu messen.
    b) Das Evaluierungstool soll die Vergleichbarkeit der Ergebnisse von Mensch und Maschine erhöhen.

3. Grundlagen

Dieses Kapitel beschreibt den Stand der Technik in einer Ausführlichkeit, die für das Verständnis dieser Arbeit benötigt wird.

3.1. Maschinelles Lernen

Als künstliche Intelligenz (engl. artificial intelligence) bezeichnet man die Intelligenz einer Maschine. Ein Teilgebiet der künstlichen Intelligenz ist maschinelles Lernen. Das Ziel von maschinellem Lernen ist, mithilfe von Algorithmen von Daten zu lernen und Vorhersagen auf Basis von Daten zu treffen. [32]
Daten können sehr unterschiedlich sein und es können auch verschiedene Arten von Vorhersagen gewünscht werden. Aus diesem Grund wird maschinelles Lernen in der Regel in vier verschiedene Kategorien eingeteilt. Diese Kategorien sind überwachtes Lernen, unüberwachtes Lernen, verstärkendes Lernen und teilüberwachtes Lernen. [40, ML techniques]

Überwachtes Lernen: Im überwachten Lernen (engl. Supervised Learning) werden gekennzeichnete Daten eingesetzt. Das bedeutet, zum Input ist der gewünschte Output bekannt. Anhand dieser Daten wird eine Funktion generiert, die bei Eingabe der Input-Daten einen Output erzeugt. Der generierte Output wird mit dem tatsächlichen Output verglichen und kann auf diese Weise Fehler in der Funktion finden und ausgleichen. Nach Abschluss des Lernens wird die Funktion auf unbekannte Daten angewandt, um deren Output zu ermitteln.
Zwei der am häufigsten verwendeten Algorithmen im überwachten Lernen sind die Klassifikation und die Regression [40, Supervised learning]. Die Klassifikation wird für die Einordnung der Input-Daten in kategoriale Klassen verwendet. Ein Spam-Filter arbeitet auf diese Weise. Der Input in Form einer Email wird entweder der Klasse „Spam“ oder „kein-Spam “ zugeordnet. Der Output ist diskret. [35, Kapitel 1.2]
Im Gegensatz dazu wird die Regression für die Vorhersage von stetigen Werten genutzt. Eine Beispielanwendung ist die Vorhersage des Bremsweges anhand der

Fahrzeuggeschwindigkeit [40, Regression]. Das Training bestimmt eine Kurve, die das Verhältnis zwischen Fahrzeuggeschwindigkeit und Bremsweg am besten beschreibt. [23, Kapitel 22]

Unüberwachtes Lernen: Für das unüberwachte Lernen (engl. Unsupervised Learning) werden Daten verwendet, die nicht gekennzeichnet sind. Das Ziel ist nicht wie beim überwachten Lernen das Vorhersagen von Klassen oder einem stetigen Wert, sondern die eigenständige Erkennung von Mustern und Beziehungen innerhalb der Daten [40, ML techniques]. Eine Anwendung ist beispielweise das Clustering. Hier werden die Inputdaten in Gruppen unterteilt, ohne dass vorher bekannt ist, zu welcher Gruppe die Daten gehören. [35, Kapitel 1.2]

Verstärkendes Lernen: Das verstärkende Lernen (engl. Reinforcement Learning) nutzt die Interaktion mit der Umgebung, um eine bestimmte Leistung zu verbessern. Diese Technik wird oftmals in der Robotik eingesetzt und beruht auf einem Belohnungssystem. Ein Agent führt innerhalb einer Umgebung Aktionen aus und erhält eine Belohnung. Das Ziel ist die Maximierung der Belohnung. [32]

Teilüberwachtes Lernen: Die Algorithmen im teilüberwachten Lernen (engl. Semi-Supervised Learning) verwenden sowohl gekennzeichnete als auch ungekennzeichnete Daten. Dabei sind mehr ungekennzeichnete als gekennzeichnete Daten vorhanden [40, ML techniques]. Die Algorithmen können, wie auch im überwachten Lernen, für Klassifikationsprobleme verwendet werden [28].

Für die Umsetzung der Aufgabe 2 wird ein Klassifikationsalgorithmus angewandt. Die folgenden Kapitel beziehen sich auf die Umsetzung einer Klassifikation.

3.2. Neuronale Netze

3.2.1. Architektur von neuronalen Netzen

Klassifikationsalgorithmen verwenden neuronale Netze, um eine Verbindung zwischen Input und Output herzustellen. Dabei basieren diese Netze auf der Architektur von Neuronen. Ein Neuron empfängt einen elektrischen Input und führt eine Prozessierung durch. Hat das Ergebnis der Prozessierung einen gewissen Schwellenwert erreicht, „feuert“ das Neuron. Hat das Ergebnis nicht diesen Schwellenwert erreicht, bleibt das Neuron inaktiv. [34]
Dieser Prozess, innerhalb eines Neurons Informationen zu bekommen, zu prozessieren und zu senden, wird im maschinellen Lernen durch ein Perzeptron simuliert [35, Kapitel 2.1.2]. Ein Modell für das Perzeptron ist in Abbildung 3.1 dargestellt. Am Ende des Abschnitts werden die einzelnen Bestandteile nochmal genauer erläutert. Eingabewerte (x_i) werden empfangen und prozessiert (Gewichtungskoeffizienten w_i, Nettoeingabefunktion und Aktivierungsfunktion). Die Nettoeingabefunktion und die Aktivierungsfunktion werden am Ende dieses Abschnittes noch genauer erläutert. Die Ausgabe ($\hat{y}$) ist -1 oder 1 und spiegelt die zwei Zustände „inaktiv“ und „feuert“ wieder. Der Schwellenwert wird durch die Bias-Einheit w_0 repräsentiert. [35, Kapitel 2.1.1]

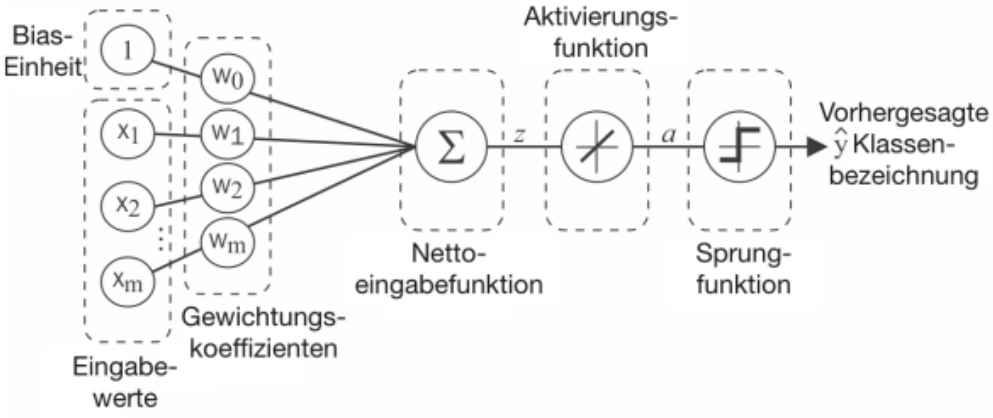


Abbildung 3.1.: Konzept eines Perzeptrons [35, Kapitel 12.1.1]

Eine solche Architektur wird auch als einschichtiges neuronales Netz bezeichnet. Durch das Zusammensetzten von mehreren Neuronen entstehen komplexere Strukturen im Form von mehrschichtigen neuronalen Netzen. [35, Kapitel 12.1.2]
Diese mehrschichtigen Netze haben folgenden Aufbau:

- Die erste Schicht ist die Eingabeschicht (engl. input layer).
- Es folgt eine beliebige Anzahl an verdeckten Schichten (engl. hidden layer).
- Die letzte Schicht ist die Ausgabeschicht (engl. output layer).

Jede Schicht besteht aus einer Menge von Aktivierungseinheiten a_i. Die Aktivierungseinheiten funktionieren wie ein Perzeptron. Gewichte und Eingabewerte der vorherigen Schicht werden durch die Nettoeingabefunktion zusammengefasst und mit der Aktivierungsfunktion verarbeitet. Die Ausgabe der Aktivierungsfunktion dient als Eingabe für die nächste Schicht. Die Aktivierungseinheit a_0 mit den dazugehörigen Gewichten stellt den Schwellenwert dar.
Dieser Aufbau ist in Abbildung 3.2 dargestellt. Da jede Aktivierungseinheit einer Schicht mit jeder Aktivierungseinheit der nächsten Schicht verbunden ist, spricht man auch von vollvermaschten Schichten.
Jede Aktivierungseinheit in der Ausgabeschicht repräsentiert eine Klasse. Das Objekt gehört zur der Klasse, deren Aktivierungseinheit den größten Wert liefert.

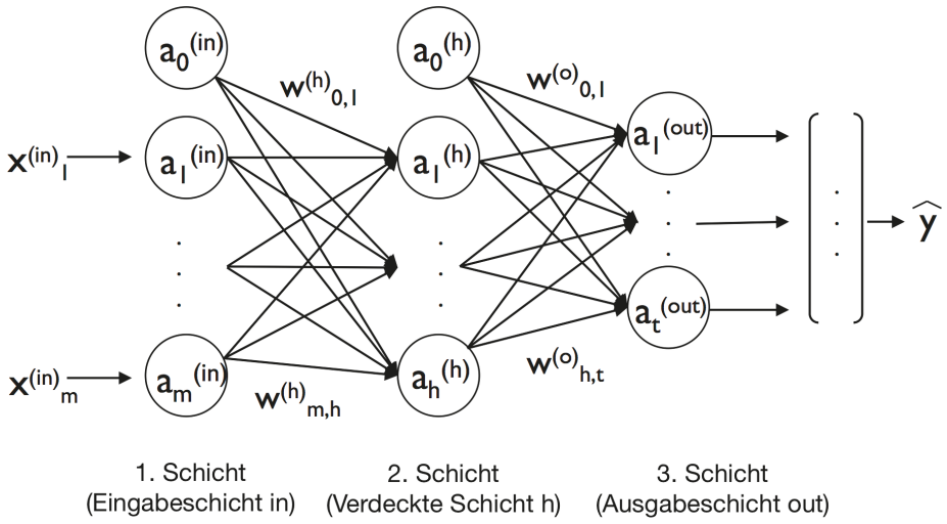


Abbildung 3.2.: Konzept eines mehrschichtigen neuronalen Netzes [35, Kapitel 12.1.2]

Nettoeingabefunktion

Die Nettoeingabefunktion rechnet die Summe der Produkte der Werte von x und w. [35, Kapitel 2.1.1]

$$z = w_0x_0 + w_1x_1 + ... + w_mx_m = \sum_{i=0}^{m} w_ix_i = w^Tx \tag{3.1}$$

Dabei entspricht w_0 dem Schwellenwert. x_0 hat den Wert 1.

Aktivierungsfunktion

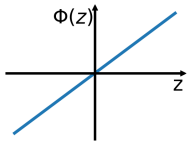
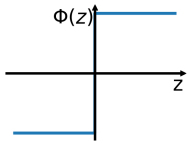
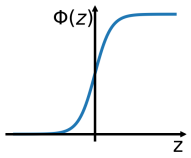
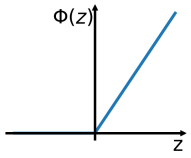
Es gibt verschiedene Arten von Aktivierungsfunktionen, die in neuronalen Netzen eingesetzt werden können. Tabelle 3.1 zeigt einige Beispiele von Aktivierungsfunktionen. Typischerweise werden die Linear- und Signum-Aktivierungsfunktion in einschichtigen neuronalen Netzen verwendet. Die Aktivierungsfunktionen Sigmoid und Rectified Linear Unit (ReLU) kommen in mehrschichtigen neuronalen Netzen zum Einsatz. [35, Kapitel 13.3.4]

Klassenzuordnung

Einschichtige neuronale Netze nutzen eine Sprungfunktion für die Zuordnung in eine von zwei Klassen. Ist das Ergebnis der Aktivierungsfunktion positiv, gehört das Objekt zur positiven Klasse und ist das Ergebnis negativ, gehört das Objekt zur negativen Klasse. [35, Kapitel 2.1.2]
In mehrschichtigen neuronalen Netzen wird für die Ausgabeschicht die Softmax-Funktion als Aktivierungsfunktion verwendet. Diese Funktion gibt die Wahrscheinlichkeit an, dass ein Objekt mit der Nettoeingabe z zur i-ten Klasse gehört. [35, Kapitel 13.3.2]

$$p(y = i|z) = \Phi(z) = \frac{e^{z_i}}{\sum_{i=1}^{M} e^{z_i}} \tag{3.2}$$

Tabelle 3.1.: A table arranging images

Aktivierungsfunktion	Gleichung	Graph
Linear	$\Phi(z) = z$	
Signum	$\Phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0. \end{cases}$	
Sigmoid	$\Phi(z) = \frac{1}{1 + e^{-z}}$	
ReLU	$\Phi(z) = \begin{cases} 0, & z < 0, \\ z, & z > 0. \end{cases}$	

3.2.2. Training eines neuronalen Netzes

Neuronale Netze können sich sehr stark unterscheiden. Manche Netze bestehen aus weniger als zehn Schichten [44] und manche Netze verwenden über 100 Schichten [13]. Egal welche Größe ein neuronales Netz hat, es wird auf die gleiche Art und Weise trainiert, nämlich durch die Anpassung der Gewichtungskoeffizienten. [35, Kapitel 12.3]

Zu Beginn des Trainings werden die Gewichte initialisiert. Hierfür gibt es unterschiedliche Strategien. Möglich ist eine Initialisierung mit 0 oder kleinen zufälligen Werten [35, Kapitel 2.1.2]. Danach wird folgender Vorgang wiederholt, bis das Training beendet wird:

1. Eingabedaten propagieren durch das Netz und erzeugen eine Ausgabe.

2. Anhand der berechneten Ausgabe und der tatsächlichen Ausgabe/Klassenzugehörigkeit wird ein Fehler berechnet. Für diese Fehlerberechnung kommt eine Loss-Funktion zum Einsatz.

3. Die Gewichte innerhalb des Netzes werden durch ein Gradientenabstiegsverfahren aktualisiert. Dies geschieht mithilfe der Ableitung der Fehlerfunktion bezüglich der Gewichtungen.

Es folgt eine genauere Erläuterung der angesprochenen Bestandteile des Trainings.

Loss-Funktion

Eine Loss-Funktion $J(w)$ ermittelt anhand des vorhergesagten und des tatsächlichen Outputs einen Fehler für ein Netz mit den Gewichten w. Es existieren verschiedene Loss-Funktionen, die den Fehler des Netzes unterschiedlich ermitteln. Für diese Arbeit werden nur die zwei hierfür relevanten Funktionen dargestellt. Weitere mögliche Funktionen werden in der Quellenangabe [40, Loss functions] genannt.
Eine Loss-Funktion ist die quadrierte Abweichung (engl. Squared Error) [31, The loss function]. Der Fehler für ein einzelnes Objekt wird folgendermaßen berechnet:

$$J(w) = SE(y, \hat{y}) = (y - \hat{y})^2 \tag{3.3}$$

y ist die berechnete Klasse des Objekts und $\hat{y}$ ist die tatsächliche Klassenzugehörigkeit. Eine weitere Funktion ist die Kreuzentropie (engl. Cross Entropy), dargestellt in Formel 3.4 für eine binäre Klassifizierung. [30]

$$J(w) = CE(p, y) = CE(p_t) = -log(p_t) \tag{3.4}$$

$$\text{mit } p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \tag{3.5}$$

y ist die tatsächliche Klassenzugehörigkeit (1 oder 0) und p ist die vorhergesagte Wahrscheinlichkeit zur Klasse 1 zu gehören.

Die Loss-Funktion $J(w)$ von mehreren Objekten wird anhand des Mittelwerts der Fehler der einzelnen Objekte ermittelt.

Gradientenabstieg und Backpropagation

Das Gradientenabstiegverfahren sucht eine Lösung für das Optimierungsproblem „Minimum der Loss-Funktion finden“. Hierfür wird ein Gradient berechnet. Dies entspricht der partiellen Ableitung der Loss-Funktion J nach den Gewichten. Der Gradient gibt an, in welcher Richtung sich der tiefste Punkt der Funktion befindet. Jedes Gewicht w_j wird anhand des Gradienten aktualisiert. Formel 3.6 drückt dies mathematisch aus. [35, Kapitel 2.3.1]

$$w_j = w_j + \Delta w_j \tag{3.6}$$

$$\text{mit } \Delta w_j = -\eta \frac{\partial J}{\partial w_j} \tag{3.7}$$

η entspricht der Lernrate. Es gibt an, um welche Schrittweite sich die Gewichte in Richtung des Minimums bewegen. Die Wahl der Lernrate ist entscheidend. Ist sie zu klein, besteht die Gefahr in lokalen Minima stecken zu bleiben. Ein zu großer Wert bedeutet ein Hinausschießen über das globale Minimum. [35, Kapitel 2.3.2]
Die Suche nach dem Minimum der Loss-Funktion anhand des Gradientenabstiegverfahrens ist in Abbildung 3.3 dargestellt.
Der Fehler der Loss-Funktion wird im Gradientenabstieg anhand der gesamten Trainingsdatenmenge n berechnet. Im Stochastischen Gradientenabstieg wird der Fehler nur anhand einer Teilmenge k der Trainingsdatenmenge ermittelt. Dies führt zu einer schnelleren Konvergenz des Netzes, da die Gewichte öfters aktualisiert werden. „Online Learning“ berechnet den Gradienten anhand eines Trainingsobjekts ($k = 1$) und „Mini Batch Learning“ nutzt eine Teilmenge k mit $1 < k < n$. [35, Kapitel 12.4]
Werden alle Trainingsdaten einmal durchlaufen, spricht man von einer Epoche [35, Kapitel 2.2.1].
Die Backpropagation liefert einen Weg, die partiellen Ableitungen für jedes Gewicht im neuronalen Netz zu ermitteln. Für eine genaue Darstellung der zugrundeliegenden

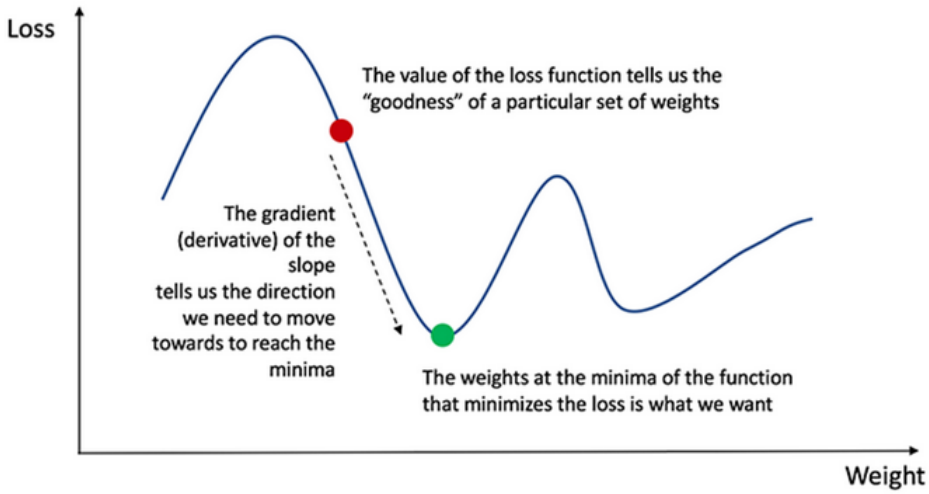


Abbildung 3.3.: Konzept des Gradientenabstiegverfahrens [31, Backpropagation]

Formeln der Backpropagation wird auf Quelle [35, Kapitel 12.3.3] verwiesen. Für das Verständnis dieser Arbeit ist eine genauere Betrachtung der Backpropagation nicht nötig.

3.3. Convolutional Neural Network

Neuronale Netze besitzen Merkmale als Input. Im Falle einer Klassifizierung von Blumen können solche Merkmale beispielsweise die Farbe der Blüten oder die Größe der Blätter sein. Zur Klassifizierung von Bilddaten werden diese Merkmale nicht mehr vom Menschen vorgegeben. Das Netz soll die Merkmale automatisch erkennen. Dies funktioniert mit Konvolutionalen Neuronalen Netzen (engl. CNN). [35, Kapitel 15.1.1]
Hierfür werden zwei weitere Arten von Schichten, die Faltungsschichten und die Poolingschichten, in das neuronale Netz eingefügt. Diese Schichten werden im Folgenden näher erläutert.

Faltungsschicht: Eine Faltungsschicht nutzt Faltungsoperationen (engl. convolution operation). Abbildung 3.4 zeigt, wie die Faltung bei zwei Funktionen f und g funktioniert. Die Faltung gibt an, wie stark sich zwei Funktionen überlappen, während eine Funktion über die andere geführt wird. Die Funktion, die sich bewegt, wird auch als Filter bezeichnet. [34, The convolution operation]
Im neuronalen Netz liegt der Filter als zweidimensionale Matrix vor. Dieser Filter wird über die Eingabe der Schicht (zweidimensionale Matrix) geführt. Die Werte des Filters werden elementweise mit den Werten des Inputs multipliziert. Die Summe der elementweisen Multiplikation ergibt den Wert des nächsten Pixels im Output. Anschließend bewegt sich der Filter um eine festgelegte Schrittweise (engl. stride) weiter und die Faltung wird erneut durchgeführt. [34, Feature extraction using filters]
Während des Trainings des neuronalen Netzes werden die Werte im Filter über die Backpropagation aktualisiert. [59]
Eine solche Faltung ist in Abbildung 3.6, hier beispielsweise mit festgelegter Schrittweite 1, zu sehen. Abbildung 3.5 zeigt den verwendeten Filter.
Filter und Eingabedaten können auch aus mehr als zwei Dimensionen bestehen [34, Feature extraction using filters]. Zur besseren Darstellung der Funktionsweise der Faltung wird in diesem Abschnitt nur das zweidimensionale Verhalten dargestellt.

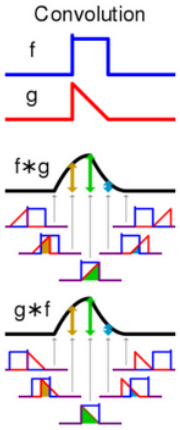


Abbildung 3.4.: Faltung zweier Funktionen f und g [34, The convolution operation]

Abbildung 3.5.: Beispiel eines Filters in einer Faltungsschicht [8]

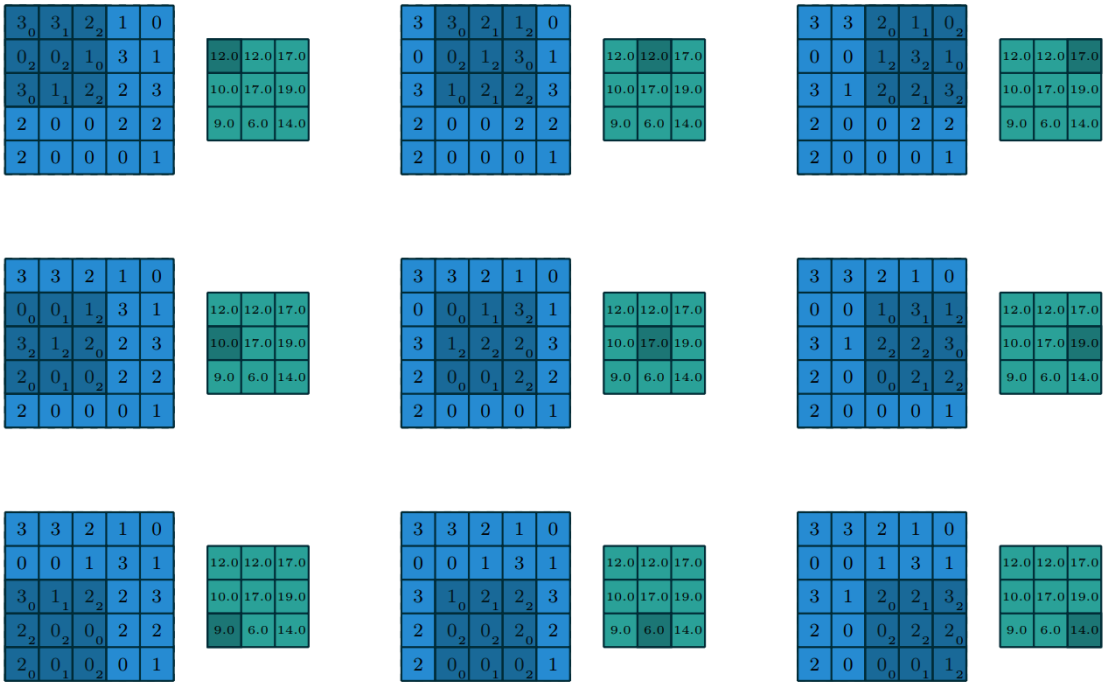


Abbildung 3.6.: Funktionsweise einer Faltungsschicht mit Schrittweite 1 [8]

Poolingschicht: Nach einer Faltungsschicht folgt meist eine Poolingschicht. Diese Schicht dient der Reduzierung der Anzahl an Merkmalen. Abbildung 3.7 zeigt die Funktionsweise einer Pooling-Schicht mit einer 3x3 Umgebung. Bei Verwendung von max-Pooling (Abbildung 3.7b) wird von einer festgelegten Umgebung der maximale Wert genommen. Bei mean-Pooling (Abbildung 3.7a) wird der Mittelwert gebildet. [35, Kapitel 15.1.3]
Dabei wird die Schrittweite meist so gewählt, dass es keine Überschneidungen der Umgebungen gibt.[35, Kapitel 15.1.3]

(a) Verwendung von mean-Pooling

(b) Verwendung von max-Pooling

Abbildung 3.7.: Funktionsweise einer Pooling-Schicht mit Schrittweite 1 [8]

Bildung eines CNN: Abbildung 3.8 zeigt, wie Faltungs- und Poolingschicht zu einem CNN zusammengesetzt werden können.
Die Eingabedaten bestehen aus einem 28x28 Pixel Bild. Es folgt eine Faltungsschicht mit Filter der Größe 5x5. Es gibt 32 Filter in dieser Schicht. Das Ergebnis wird mit einem Pooling-Filter der Größe 2x2 verkleinert. Die zweite Faltungsschicht besteht aus 64 Filtern der Größe 5x5. Die zweite Poolingschicht besteht auch wieder aus einem Filter der Größe 2x2. [35, Kapitel 15.3.1]
Es folgt eine Umformung, um eine Verbindung zu einer vollvermaschten Schicht mit 1024 Aktivierungseinheiten zu schaffen. Diese Umformung wird als „flatten Layer“ bezeichnet und wandelt die mehrdimensionale Ausgabe der Pooling-Schicht in einen eindimensionalen Vektor um [36, Kapitel 5.3]. Damit kann jeder Wert der zweiten Poolingschicht mit jeder Aktivierungseinheit der nächsten Schicht verbunden werden. Nach dieser vollständigen Verknüpfung folgt eine Output-Schicht mit 10 Elementen.

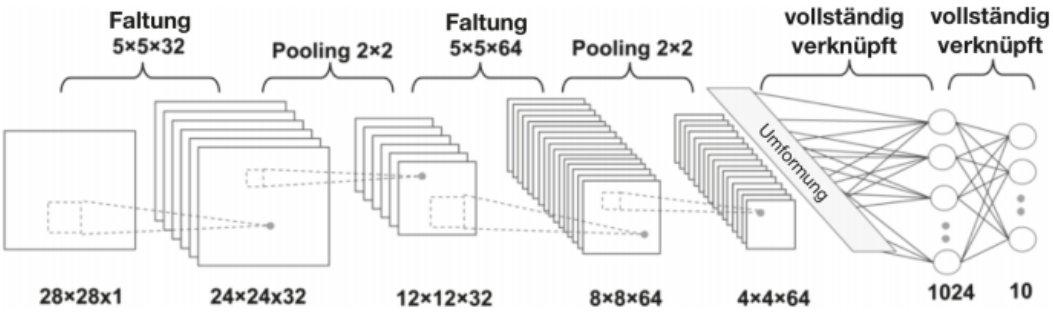


Abbildung 3.8.: Beispielnetz eines CNN [35, Kapitel 15.3.1]

3.4. Evaluierung eines neuronalen Netzes

Kapitel 3.2.2 beschreibt, wie anhand von Trainingsdaten ein neuronales Netz trainiert wird. In jedem Durchgang werden die Gewichte anhand der Ableitung der Loss-Funktion aktualisiert. Dieser Abschnitt beschreibt, wie die Qualität des resultierenden Modells bewertet werden kann.

3.4.1. Bewertung der Ergebnisse

Ein neuronales Netz besitzt zahlreiche Parameter, die geändert werden können. Innerhalb der Architektur des neuronalen Netzes sind dies beispielsweise die Anzahl der Schichten oder die Aktivierungsfunktion innerhalb einer Schicht. Auch die Loss-Funktion lässt sich ändern. Solche Parameter werden als Hyperparameter bezeichnet. Diese Parameter werden nicht vom Netz trainiert, sondern vom Mensch vorgegeben. [40, Hyperparameters of the neural network]
Durch die Änderung von Hyperparametern können viele verschiedene Netze entstehen. Es ist nötig ein Verfahren festzulegen, um ein trainiertes neuronales Netz zu bewerten.
Die Bewertung eines Netzes erfolgt auf Basis einer Konfusionsmatrix. Eine solche Konfusionsmatrix im Falle einer binären Klassifizierung ist in Abbildung 3.9 dargestellt.

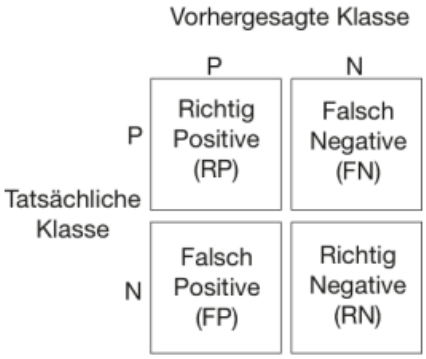


Abbildung 3.9.: Konzept einer Konfusionsmatrix [35, Kapitel 6.5.1]

In einer Konfusionsmatrix wird zwischen folgenden vier Zuständen unterschieden:

- Richtig Positiv (RP): Ein Objekt der positiven Klasse wird der positiven Klasse zugeordnet.
- Falsch Positiv (FP): Ein Objekt der positiven Klasse wird fälschlicherweise der negativen Klasse zugeordnet.
- Richtig Negativ (RN): Ein Objekt der negativen Klasse wird der negativen Klasse zugeordnet.
- Falsch Negativ (FN): Ein Objekt der negativen Klasse wird fälschlicherweise der positiven Klasse zugeordnet.

Anhand der Zuordnung der Elemente in diese vier Zustände lässt sich das trainierte Modell bewerten. Formeln 3.8 - 3.12 zeigen verschiedene Metriken und deren Berechnung anhand der Konfusionsmatrix.

$$Accuracy = \frac{RP + RN}{RP + RN + FP + FN} \tag{3.8}$$

$$ErrorRate = 1 - Accuracy \tag{3.9}$$

$$Precision = \frac{RP}{RP + FP} \tag{3.10}$$

$$Recall = \frac{RP}{RP + FN} \tag{3.11}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.12}$$

Im Deutschen wird die Accuracy als Korrektklassifizierungsrate und die Error Rate als Fehlerquote bezeichnet. Sie geben an, welcher Prozentteil der Objekte in der Datenmenge richtig bzw. falsch klassifiziert wurde. Precision kann als Genauigkeit und der Recall als Trefferquote bezeichnet werden. [35, Kapitel 6.5]
Die Precision teilt die Anzahl an Richtig Positiven Elementen durch die Anzahl an Elementen, die als positiv klassifiziert wurden. Die Metrik liefert einen Wert, der angibt, wie viele Elemente einen Fehlalarm bilden. Dabei liegt der Wert in einem Intervall von 0 bis 1. Bei einem Wert von 1 gehören alle vom Modell als positiv klassifizierten Elemente auch wirklich zur positiven Klasse. Der Wert 0 gibt an, dass keins der als positiv klassifizierten Elemente zur positiven Klasse gehört. [35, Kapitel 6.5]
Der Recall teilt die Anzahl an Richtig Positiven Elementen durch die Anzahl der tatsächlich Elemente der positiven Klasse. Der Wert liegt auch hier im Intervall von 0 bis 1. Diese Metrik gibt an, wie viele Elemente übersehen werden. Ein Wert von 1 bedeutet, dass alle positiven Elemente auch vom Modell als positiv klassifiziert wurden. Ein Wert von 0 bedeutet, keines der positiven Elemente wurde auch als positiv klassifiziert. [35, Kapitel 6.5]
Oftmals wird in der Praxis auch eine Kombination aus Precision und Recall in Form des F1-Maßes verwendet. [35, Kapitel 6.5]
Seliya et al. [43] stellt noch weitere Metriken vor, die auf Basis der Konfusionsmatrix ermittelt werden können. Es wird auch die Aussage getroffen, dass nicht zu viele Metriken für die Bewertung des Modells eingesetzt werden sollten, da keine neuen Informationen gewonnen werden [43]. Welches Maß für die Auswahl des neuronalen Netzes relevant ist, ist vom jeweiligen Anwendungsfall abhängig [35, Kapitel 6.5].
Die beiden in diesem Kapitel dargestellten Metriken Precision und Recall werden für die Evaluierung des neuronalen Netzes verwendet. Mit Recall wird ermittelt, wie viele Kitze übersehen werden und mit Precision werden die Fehlalarme ermittelt.

3.4.2. Feinabstimmung der Hyperparameter

Ein neuronales Netz kann mit den in Kapitel 3.4.1 dargestellten Metriken bewertet werden. Dieser Abschnitt stellt dar, auf welche Weise die Werte für die Hyperparameter ermittelt werden können.

Rastersuche

Welche Hyperparameter optimiert werden sollen, kann über eine Rastersuche (engl. Gridsearch) vorgegeben werden. Für eine Rastersuche wird eine Menge an Hyperparametern angegeben und mit jeder möglichen Kombination der Hyperparameter wird ein Netz trainiert und bewertet [35, Kapitel 6.4]. Beispielsweise bei einer Lernrate-Menge von $\{0.1, 0.01, 0.001, 0.0001\}$ und Anzahl der Neuronen in einer Schicht von $\{50, 100, 200, 500, 1000\}$ werden $4 \cdot 5 = 20$ Netze trainiert. [40, Hyperparameters of the neural network]
Dies kann leicht zu einer großen Menge an Netzen führen, die trainiert werden müssen. Kirori [25, S. 3] verwendet aus diesem Grund eine zufällig gewählte Untermenge an Kombinationen für die Rastersuche.

k-fache Kreuzvalidierung

Um die optimale Kombination von Hyperparametern zu ermitteln, ist die k-fache Kreuzvalidierung (engl. cross-validation) eine mögliche Methode. [48]
Die k-fache Kreuzvalidierung teilt die Datenmenge in k Teile und trainiert das Netz k mal. Jedes der k Teile wird einmal als Testdaten verwendet, während die restlichen $k - 1$ Teile die Trainingsdaten bilden. Die Trainingsdaten werden verwendet, um die Gewichte des Netzes anzupassen. Anhand der Testdaten und einer Metrik wird das Netz evaluiert.
Diese Einschätzung der Leistung des Netzes wird in jedem der k Durchgänge durchgeführt. Anschließend erfolgt aus diesen k Einschätzungen die Bildung eines Mittelwertes. Dieser Mittelwert wird zur Bewertung der Hyperparameter verwendet. Somit

kann eine verallgemeinerungsfähige Aussage über die Qualität des Netzes getroffen werden. [35, Kapitel 6.2]
Für jede Kombination der Hyperparameter in der Rastersuche wird mit der k-fachen Kreuzvalidierung ein Mittelwert der Leistung gebildet. Die Kombination mit dem höchsten Mittelwert wird ausgewählt und mit der gesamten Datenmenge trainiert. Das auf diese Art und Weise generierte Modell wird für Vorhersagen verwendet. [42]
In Abbildung 3.10 ist eine solche k-Kreuzvalidierung mit einer Einteilung von $k = 10$ Teilen dargestellt.
Welcher Wert als k gewählt wird, ist abhängig von der Datenmenge. Bei kleineren Datenmengen wird das k größer gewählt, um eine ausreichend große Untermenge an Trainingsdaten zu erreichen. Bei großen Datensätzen kann das k auch kleiner gewählt werden, um den Rechenaufwand gering zu halten. [35, Kapitel 6.2]

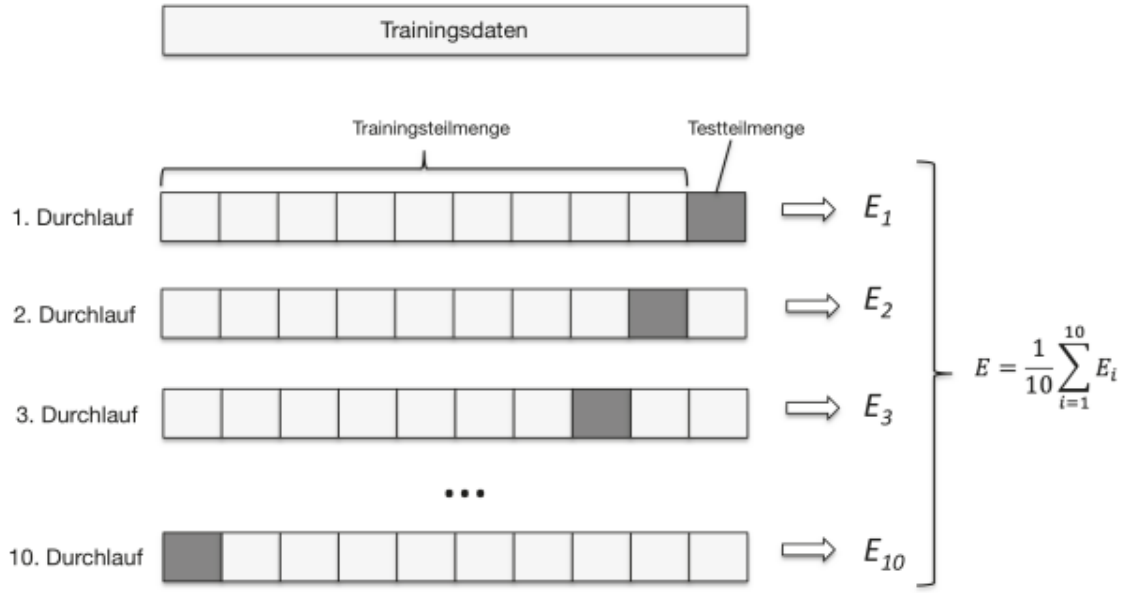


Abbildung 3.10.: Beispiel einer Kreuzvalidierung mit Wert $k = 10$ [35, Kapitel 6.2.2]

Verschachtelte Kreuzvalidierung

Die Kreuzvalidierung lässt sich noch in der Form einer verschachtelten Kreuzvalidierung (engl. nested cross-validation) erweitern. Eine äußere Schleife teilt die Datenmenge in Trainings- und Testdaten ein. Die innere Schleife nutzt die Kreuzvalidierung, um die beste Kombination an Hyperparametern zu finden. Das Netz mit den besten Hyperparametern wird nochmal mit der Trainingsdatenmenge der äußeren Schleife trainiert und anhand der Testdaten bewertet.
In Abbildung 3.11 ist die verschachtelte Kreuzvalidierung mit einer äußeren Schleife von $k = 5$ und einer inneren Schleife von $k = 2$ dargestellt. Mit einem solchen Modell lässt sich gut abschätzen, wie gut das Netz mit den jeweiligen Hyperparametern auf unbekannte Daten angewandt werden kann [35, Kapitel 6.4]. Im Vergleich zur k-fachen Kreuzvalidierung ist hiermit aber auch ein erhöhter Rechenaufwand verbunden [49].

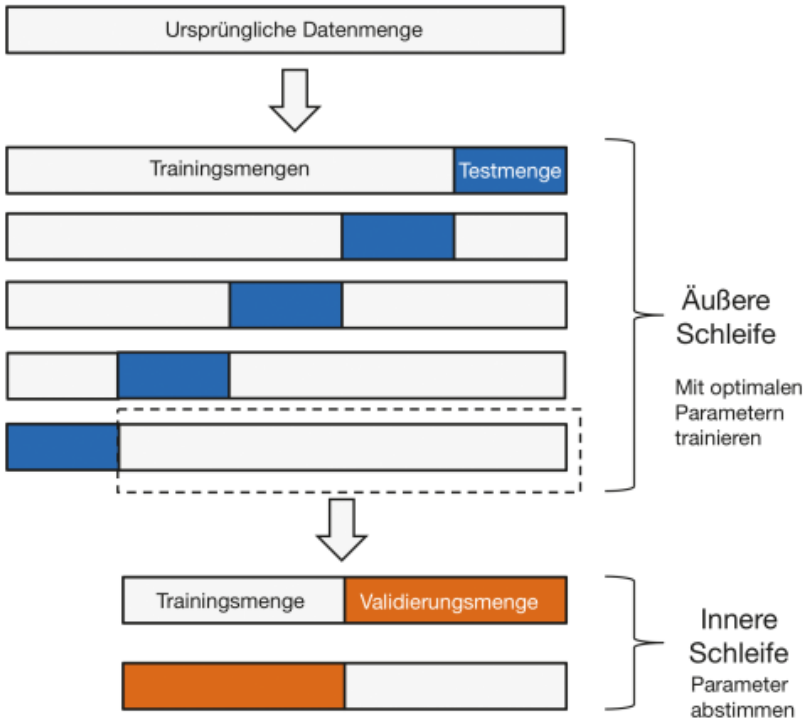


Abbildung 3.11.: Beispiel einer verschachtelten 5x2 Kreuzvalidierung [35, Kapitel 6.4.2]

3.5. Probleme im maschinellen Lernen

Sarkar et al. [34, The pitfalls of ML] erwähnt fünf typische Probleme im Umgang mit maschinellem Lernen. Dies sind unausgewogene Datenmengen, Overfitting, Underfitting, schlechte Daten sowie irrelevante Merkmale und Klassen.
Diese fünf Punkte werden im Folgenden näher erläutert.

3.5.1. Unausgewogene Datenmenge

Trainings- und Testdaten im überwachten Lernen besitzen Label. Solche Label ordnen die Daten einer Klasse zu. Nicht immer sind von jeder Klasse die gleiche Menge an Daten verfügbar. In solchen Fällen spricht man von unausgewogenen Datenmengen (engl. unbalanced dataset). [35, Kapitel 6.6]
Durch die Unausgewogenheit kann es zu einer Klassifizierung zugunsten der überrepräsentierten Klasse kommen. Dabei ist es gerade in Fällen von unausgewogenen Datenmengen meist wichtig, ein Element der unterrepräsentierten Klasse nicht zu übersehen, wenn sie auftauchen. Als typisches Beispiel gilt die Detektion von Krankheiten im medizinischen Umfeld. Ein Großteil der Menschen ist gesund, aber falls eine Krankheit vorliegt, sollte diese auch zuverlässig erkannt werden. [21]
Nicht jede Evaluierungsmetrik ist im Falle von unausgewogenen Daten gut geeignet. Beispielsweise ist die Accuracy hier keine gute Metrik. Gehören 1% des Datensatzes zur positiven Klasse, kann eine Accuracy von 99% erreicht werden, wenn alle Objekte als negativ vorhergesagt werden. Kein Objekt der positiven Klasse wird erkannt. Welche Metrik stattdessen verwendet wird, ist vom Anwendungsfall abhängig. Recall und Precision sind mögliche Metriken. [54]
Johnson et al. [21, S. 4] bestimmt ein Maß für die Unausgewogenheit durch das Verhältnis ρ in Formel 3.13.

$$\rho = \frac{max_i\{|C_i|\}}{min_i\{|C_i|\}} \tag{3.13}$$

C_i gibt an, wie viele Elemente zu einer Klasse i gehören. $max_i\{|C_i|\}$ bestimmt die maximale Klassengröße von allen Klassen und $min_i\{|C_i|\}$ die minimale Klassengröße. Im binären Fall gibt ρ das Verhältnis zwischen der Anzahl an Elementen in der

überrepräsentierten Klasse (engl. majority group) und der unterrepräsentierten Klasse (engl. minority group) an.
Beim Umgang mit unausgewogenen Daten sollte unter anderem beachtet werden, aus welchem Grund die Unausgewogenheit besteht. Bei einer intrinsic imbalance liegt eine Unausgewogenheit aufgrund des natürlichen Auftretens der Daten vor. Dies tritt beispielsweise bei der Vorhersage von Krankheiten auf. Von einer extrinsic imbalance ist die Rede, wenn externe Probleme während der Datensammlung oder der Datenspeicherung zu einer Unausgewogenheit führen. Setzt beispielsweise während der Übertragung von balancierten Daten die Übertragung zwischendrin aus, kann das zu Unausgewogenheit führen. [12, Kapitel 2]
Wird die Unausgewogenheit erkannt und entsprechend beim Training beachtet, können neuronale Netze die gewünschten Ergebnisse liefern. Beispielgebend ist hier die Arbeit von Reza et al [38] die zeigt, dass es möglich ist, mit einem CNN und unausgewogenen Daten zu trainieren.

3.5.2. Überanpassung und Unteranpassung

Im überwachten Lernen werden neuronale Netze mithilfe von Daten trainiert, bei denen ein Output zu einem bestimmten Input bekannt ist. Das Ziel ist es, dass das trainierte Modell auch mit unbekannten Daten (Testdaten) funktioniert.
Ist das trainierte Modell zu spezialisiert, liefert es bei den verwendeten Trainingsdaten eine gute Vorhersagekraft, aber bei neuen und unbekannten Daten eine hohe Fehlerrate. In solchen Fällen spricht man von einer Überanpassung (engl. overfitting). Ein häufiger Grund für eine Überanpassung ist, wenn es zu viele Parameter gibt. Es entsteht ein komplexes Modell, was nicht verallgemeinerungsfähig ist. [35, Kapitel 3.3.5]
Im Gegensatz dazu steht die Unteranpassung (engl. underfitting). Zu einer Unteranpassung kommt es, wenn das Modell nicht komplex genug ist. In diesen Fällen besitzt das Modell auch bei Trainingsdaten eine hohe Fehlerrate. [35, Kapitel 3.3.5]
Bei einem überangepassten neuronalen Netz spricht man auch von einem Netz mit hoher Varianz, bei Unteranpassung von einem Netz mit hohem Bias. [35, Kapitel

3.3.5]
Da Unteranpassung bereits mit den Trainingsdaten eine hohe Fehlerrate liefert, ist Unteranpassung einfacher zu erkennen als eine Überanpassung. [1, S. 27]
Um dieses Problem zu verdeutlichen, ist in Abbildung 3.12 ein Klassifizierungsalgorithmus dargestellt, welcher anhand einer Entscheidungsgrenze (gestrichelte Linie in der Abbildung) Objekte der Kreis-Klasse oder der Kreuz-Klasse zuordnet.
Im Falle der Unteranpassung wird eine lineare Entscheidungsgrenze gewählt. Mit dieser sehr einfachen Grenze werden vier der insgesamt zwölf Objekte nicht der richtigen Klasse zugeordnet. Die Entscheidungsgrenze im Falle der Überanpassung ordnet alle Objekte der richtigen Klasse zu. Die Grenze ist zu stark an die vorhandenen Objekte angepasst. So ist es wahrscheinlich, dass neue Daten öfters falsch klassifiziert werden. Das mittlere Modell bildet einen guten Kompromiss zwischen Under- und Overfitting. Ein Objekt in der Trainingsmenge wird zwar falsch klassifiziert, dafür ist aber die parabelförmige Entscheidungsgrenze gleichmäßiger als beim überangepassten Modell.

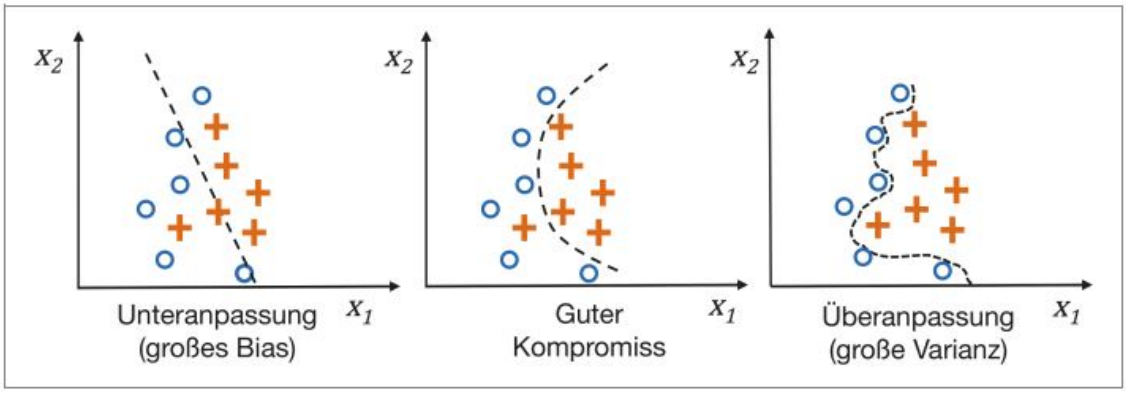


Abbildung 3.12.: Über- und Unteranpassung bei Bestimmung einer Entscheidungsgrenze [35, Kapitel 3.3.5]

3.5.3. Schlechte Daten

Die Qualität der Daten hat einen Einfluss auf die Vorhersagekraft eines neuronalen Netzes. Schlechte Daten können beispielsweise durch fehlerhaft platzierte Sensoren

entstehen [34, The pitfalls of ML]. Es ist wichtig, schlechte Daten zu erkennen und festzulegen, wie damit umgegangen werden soll. [35, Kapitel 4]
Diese Vorverarbeitung von Daten wird in Kapitel 4.1.1 weiter thematisiert.

3.5.4. Irrelevante Merkmale und Klassen

Beim Sammeln von Daten für das Training eines neuronalen Netzes kann es zum „Fluch der Dimensionalität“ kommen [23, Kapitel 12]. Je mehr Merkmale oder Klassen für das neuronale Netz verwendet werden, desto mehr Daten müssen gesammelt werden. Sonst erfolgt die Abschätzung des Netzes nicht gut genug. [35, Kapitel 3.7]
Demzufolge ist darauf zu achten, dass nur relevante Klassen definiert und auch nur relevante Merkmale an das neuronale Netz übergeben werden.

3.6. Werkzeuge für maschinelles Lernen in Python

Gemäß Aufgabenstellung Punkt 2a soll Python für das Trainieren eines neuronalen Netzes verwendet werden. Um maschinelles Lernen umzusetzen, bieten sich in Python Tools wie Keras und Tensorflow an.
Keras ist eine Python Bibliothek für Deep Learning. Mit Keras ist es möglich, Modelle zu definieren, zu trainieren und Vorhersagen zu treffen. Keras folgt einem modularen Grundprinzip. Die Bestandteile eines Modells können beliebig zusammengesetzt werden. Mit Keras sind bereits viele Bestandteile von neuronalen Netzen vordefiniert. Unter anderem sind dies verschiedene Arten von Schichten, Loss-Funktionen und Aktivierungsfunktionen. Sollten diese vordefinierten Bestandteile nicht ausreichen, ist Keras auch erweiterbar. Neue Komponenten können definiert und dem Keras Modell hinzugefügt werden. [2, Kapitel 4] [10]
Somit bietet Keras eine high-level Schnittstelle für neuronale Netze. Für eine schnelle und effiziente Berechnung der definierten Keras-Netze werden Backend-Engines wie Tensorflow, Theano oder CNTK verwendet. Für diese Arbeit wird Tensorflow eingesetzt. Diese von Google entwickelte Python Bibliothek für schnelles numerisches

Rechnen ist ausgereifter und am weitesten verbreitet. [2, Kapitel 3] [3, Kapitel 3] Sowohl Keras als auch Tensorflow unterstützen die Berechnung auf Central Processing Unit (CPU) und Graphics Processing Unit (GPU). Eine Berechnung auf einer GPU kann die benötige Rechenzeit gegenüber einer Berechnung auf der CPU deutlich verkürzen. [4] [16]

4. Konzeption

Kapitel 3 beschreibt verschiedene Bestandteile des maschinellen Lernens. Das Grundkonzept vom maschinellen Lernen sieht es vor, ein Modell zu generieren, welches Eingaben entgegennimmt und zu diesen Eingaben eine Ausgabe erzeugt. Nun soll ein Verfahren festgelegt werden, um Aufgabenstellung 2 umzusetzen.

4.1. Workflow bei der Generierung eines neuronalen Netzes

Yu et al [58] beschreibt die Problematik, dass viele Arbeiten im Zusammenhang mit neuronalen Netzen nur wenig Zeit in die Datenvorbereitung investierten. Gleichzeitig wird die Aussage getroffen, dass die Qualität der Daten entscheidend für die Ergebnisse des Trainings ist. Es wird empfohlen, ein Framework für Datenanalysen zu verwenden. [58]
Für diese Arbeit wird das CRoss Industry Standard Process for Data Mining (CRISP-DM) Framework verwendet, ein weit verbreitetes und beliebtes Prozess-Modell für Data Mining und analytische Projekte. [40, CRISP-DM]
In Abschnitt 4.1.1 wird zunächst CRISP-DM im Detail erläutert. Daraufhin wird in Abschnitt 4.1.2 beschrieben, wie dies für die Bestimmung eines Rehkitzdetektionsalgorithmus angewandt wird.

4.1.1. CRISP-DM

CRISP-DM definiert eine Reihe an Arbeitsschritten, die im Rahmen der Bearbeitung des Projekts durchgeführt werden. Diese Arbeitsschritte werden in sechs Phasen unterteilt, dargestellt in Abbildung 4.1. Dies wird auch als Data Mining Lifecycle bezeichnet. [41, The CRISP-DM Process Model] (Quelle erwähnt stellvertretend für

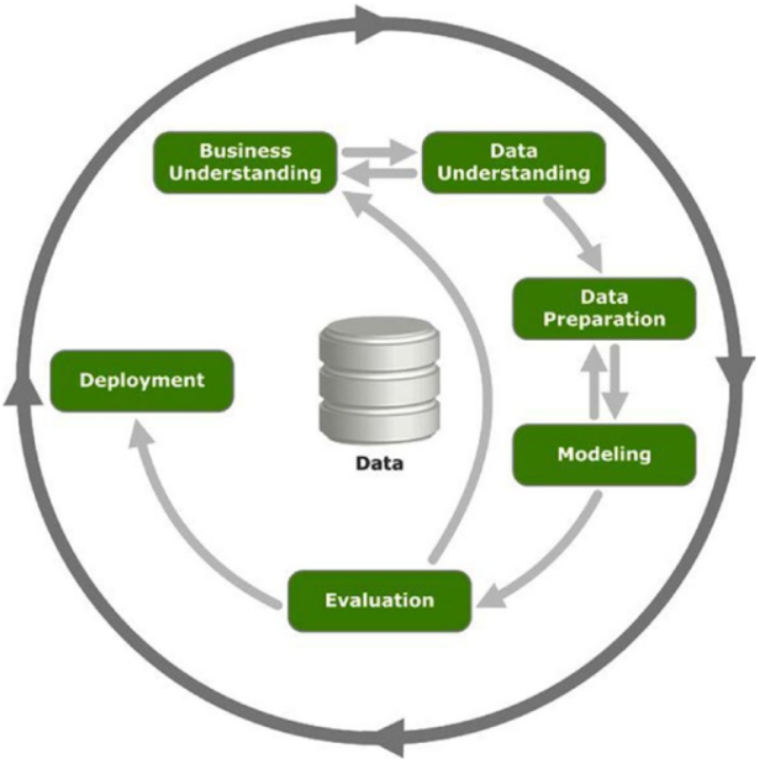


Abbildung 4.1.: Data Mining Lifecycle nach CRISP-DM [41, The CRISP-DM Process Model]

den gesamten Abschnitt 4.1.1)
Im Folgenden werden die sechs Phasen näher erläutert.

Verstehen der Aufgabe

Beginnend mit der ersten Phase (Business Understanding) erfolgt die Definition eines Kontexts und der Voraussetzungen für das zu lösende Problem. Außerdem werden Erwartungen und Erfolgskriterien ermittelt. Dabei sollten folgende Punkte durchgeführt werden:

- Definition des Geschäftsproblems:
  Zu Beginn des Projekts wird der Kontext des zu bearbeiteten Problems untersucht. Teil dieser Untersuchung ist die Beschreibung bereits vorhandener

Lösungen. Dabei wird auch darauf eingegangen, was an diesen Lösungen fehlt und verbessert werden kann.
Ein weiterer Bestandteil ist das Festlegen eines Erfolgskriteriums auf geschäftlicher Ebene.

- Beurteilung und Analyse der Situation:
  Es wird eine Übersicht über vorhandene und benötigte Ressourcen erstellt. Diese Ressourcen können unter anderem Daten, Personal, Hardware und Software sein. Falls vorhanden, sollten Annahmen und Einschränkungen auch erwähnt werden.
  Ein weiterer Bestandteil ist die Beschreibung von Risiken im Zusammenhang mit dem Projekt.
- Definition des Daten-Modellierungs-Problems:
  Die verfügbaren möglichen Methoden des maschinellen Lernens werden beschrieben. Hierfür werden Tools, Algorithmen und Techniken untersucht. Außerdem erfolgt die Bestimmung eines Erfolgskriteriums aus technischer Sicht (z.B. 80% Accuracy).
- Projekt Plan:
  In einem Projekt Plan wird das weitere Vorgehen dokumentiert. Dies beinhaltet beispielsweise die Dokumentation der erwarteten Zeit und der verwendeten Ressourcen in den weiteren Phasen.

Verständnis der Daten

Die Phase Data Understanding untersucht die vorhandenen Daten, sodass ein grundlegendes Verständnis für die Daten folgendermaßen aufgebaut wird:

- Datenerfassung:
  Anhand der vorhandenen Datenquellen wird ermittelt, welche Daten vorliegen und was noch beschafft werden muss.
- Datenbeschreibung:
  Die erfassten Daten werden beschrieben. Dies beinhaltet beispielsweise das Datenformat und die Datengröße. Auch Verhältnisse zwischen den Attributen werden beschrieben.

- Datenanalyse:
  Diese Aufgabe, die oft auch als Exploratory Data Analysis (EDA) bezeichnet wird, gibt einen näheren Einblick in die vorhandenen Daten. Durch Visualisierungen, Statistiken und Plots wird ein Verständnis für die Daten aufgebaut. Dies deckt beispielsweise Assoziationen und Korrelationen zwischen den Attributen auf.
- Datenqualität:
  Als Abschluss der Phase werden die Daten auf ihre Qualität untersucht. Potentielle Fehler, Mängel und Probleme werden dokumentiert und in der nächsten Phase behoben. Möglich sind unter anderem die Untersuchung auf fehlende bzw. inkonsistente Werte oder falsche Metadateninformationen.

Datenvorbereitung

In der Phase Data Preparation werden die Daten in eine Form gebracht, die für das neuronale Netz verwendet werden kann. Dabei werden folgende Aufgaben bearbeitet:

- Datenintegration:
  Werden die verwendeten Daten aus verschiedenen Quellen bezogen, müssen sie in diesem Arbeitsschritt zusammengeführt werden.
- Datenselektion:
  Nicht jede verfügbare Information ist auch für das Training relevant. Auf Basis der EDA werden die relevanten Bestandteile der Daten ausgewählt.
- Datenaufbereitung:
  Die Datenaufbereitung bringt die Daten in die für das neuronale Netz benötigte Form. Probleme in der Datenqualität, die in der letzten Phase aufgedeckt wurden, werden hier behoben. Mögliche Schritte sind hier das Einfügen von fehlenden Werten oder das Entfernen von problematischen Daten.
  Abschließend werden die Daten in das nötige Speicherformat gebracht.

Modellierung

Die Phase der Modellbildung (Modeling) ist ein iterativer Prozess, in dem mehrere Modelle gebildet werden. Das Ziel ist das Finden eines Modells, welches das definierte Erfolgskriterium erreicht. Folgendermaßen wird vorgegangen:

- Auswahl der Modellierungstechnik:
  Für diese Phase wird zu Beginn eine Liste an relevanten Techniken und Algorithmen für die Modellbildung bestimmt.
- Modellgenerierung:
  Die festgelegten Modelle werden generiert und trainiert. Wichtig ist das Festhalten der verwendeten Parameter und der Ergebnisse.
- Modellevaluierung und -abstimmung:
  Jedes Modell wird auf Basis von festgelegten Metriken evaluiert. Teil dieser Aufgabe ist auch die Optimierung der Hyperparameter mithilfe von Techniken wie Rastersuche und Kreuzvalidierung.
- Modellbewertung:
  Es wird untersucht, ob die Performance mit dem definierten Erfolgskriterium übereinstimmt und ob die Ergebnisse zufriedenstellend sind.
  Die Bewertung des Modells sollte auch die Reproduzierbarkeit und Erweiterbarkeit des Modells betrachten.

Evaluierung

Mit der Phase der Modellierung wurde ein Modell ermittelt, welches das Erfolgskriterium erfüllt. In der Phase der Evaluierung (Evaluation) wird eine detaillierte Beurteilung des ermittelten Modells durchgeführt.
Zusätzlich können auch Annahmen und Einschränkungen betrachtet werden, die durch das Modell entkräftet wurden oder auch Berichte über die Datengenügsamkeit, welche Probleme aufgetreten sind und was vermieden werden sollte.

Einführung ins Projekt

Das evaluierte Modell wird in dieser Phase (Deployment) in das Projekt eingeführt. Zusätzlich erfolgt die Festlegung eines Planes für regelmäßige Überwachungen. Nach Möglichkeit soll die Performance des Netzes überprüft werden, um es bei Bedarf auszutauschen oder zu aktualisieren, wenn es nicht die gewünschten Ergebnisse liefert.

4.1.2. Umsetzung für die Rehkitzdetektion

Die beschriebenen sechs Phasen des CRISP-DM werden für diese Arbeit eingesetzt, um ein neuronales Netz für die Rehkitzdetektion zu generieren. Dabei wird ein neuronales Netz auf Basis der Arbeit von Michael Sorg [46] erstellt. Es wird vor allem auf die in Kapitel 3.5 vorgestellten Probleme im maschinellen Lernen eingegangen. Die schlechten Daten (vgl. Abschnitt 3.5.3) werden mit der Phase der Datenvorbereitung behandelt. Das Problem mit irrelevanten Merkmalen und Klassen (vgl. Abschnitt 3.5.4) ist Thema der Datenanalyse. Verschiedene Lösungen für unausgewogene Datenmengen (vgl. Abschnitt 3.5.1) und Über- bzw. Unteranpassung (vgl. Abschnitt 3.5.2) werden in der Modellbildung analysiert.
Die Ausführung ist in Kapitel 5 dargestellt. Die Abschnitte orientieren sich dabei an den sechs Phasen des CRISP-DM.
Die ersten drei Phasen werden wie im letzten Abschnitt beschrieben durchgeführt. Es wird lediglich auf das Aufführen eines Projekt Plans in der ersten Phase verzichtet, da das weitere Vorgehen bereits in diesem Abschnitt beschrieben wird.
Die Modellbildung wird daraufhin in zwei Schritten durchgeführt. Zuerst werden verschiedene Modelle mit Lösungen für die Probleme im maschinellen Lernen trainiert und analysiert. Auf Basis der Untersuchungen werden im nächsten Schritt die vielversprechendsten Hyperparameter ausgesucht und in einer Kreuzvalidierung trainiert. Das nach der Kreuzvalidierung beste Modell wird ausgewählt und weiter evaluiert.
Die Auswahl eines Modells soll primär die Ergebnisse darstellen. Im Rahmen dieser Arbeit erfolgt keine Untersuchung der Gründe für eine schlechte Performance.

Die Evaluierung bewertet die Ergebnisse anhand der beiden Metriken Recall und Precision. Zusätzlich erfolgt die Angabe der Kovarianzmatrix. Außerdem wird das ausgewählte Modell mit der visuellen Detektion verglichen. Das Konzept der visuellen Konzeption ist in Kapitel 4.2 näher erläutert.
Für die Phase der Einführung ins Projekt wird lediglich beschrieben, wie das neuronale Netz im Wildretter-Projekt verwendet und überwacht werden kann. Die eigentliche Einführung ins Projekt wird nicht im Rahmen dieser Arbeit durchgeführt.

4.2. Vergleichbarkeit mit visueller Detektion

Das mit dem im vorherigen Abschnitt beschriebenen Workflow ermittelte neuronale Netz soll nicht nur mit dem in Kapitel 3.4 vorgestellten Metriken (Precision und Recall) evaluiert werden. Die Vorhersagekraft des neuronalen Netzes wird auch mit der visuellen Detektion verglichen (siehe Aufgabenstellung 3). Die visuelle Detektion ermittelt, wie gut der Mensch Kitze auf Bildern erkennen kann.
Für eine vorangegangene Arbeit von Martin Israel [20, Kapitel 4.6.2] wurde bereits für eine Menge von Thermalbildern eine visuelle Detektion durchgeführt. Anhand der Daten eines Flugs wird bestimmt, an welchen GPS-Positionen ein Tier vermutet wird. Zusammen mit den tatsächlichen Positionen von Tieren wird die Anzahl an falsch positiven, richtig positiven und falsch negativen Fundstellen ermittelt.
Dabei werden die Fundstellen nicht anhand der einzelnen Bilder festgelegt. Die Sequenz der Bilder ist für die Klassifizierung einer Fundstelle entscheidend. Eine Stelle wird auf mehreren Bildern hintereinander begutachtet und gegebenfalls als Fundstelle markiert.
Das in Kapitel 3.3 vorgestellte CNN setzt ein anderes Verhalten für die Ermittlung von Recall und Precision um. Jedes Bild wird einzeln auf einen Kitzfund vom neuronalen Netz überprüft. Die ermittelte visuelle Detektionsrate ist nicht mit der Detektionsrate des CNN vergleichbar.
Um eine Vergleichbarkeit zu schaffen, wird die visuelle Detektion für diese Arbeit wiederholt. Wie bereits in der erwähnten vorangegangenen Arbeit [20, Kapitel 4.6.2]

übernimmt auch in dieser Arbeit der fachliche Betreuer (Martin Israel) die visuelle Detektion. Nach der Aufbereitung der Daten (siehe Kapitel 5.3) werden Testdaten ermittelt. Die Beurteilung dieser Testdaten erfolgt sowohl anhand des neuronalen Netzes als auch anhand der visuellen Einschätzung. Jedes Bild wird einzeln als Tier oder kein-Tier eingestuft.
Das entwickelte Tool für die visuelle Detektion besitzt folgende Eigenschaften:

- Die Bilder werden in einer zufälligen Reihenfolge angezeigt.
- Die Bilder werden nach Flügen aufgeteilt.
- Neun Bilder werden gleichzeitig dargestellt.
- Durch einen Klick auf ein Bild wird die Klassifizierung geändert. Standardmäßig ist ein Bild als kein-Tier gekennzeichnet.
- Die nächsten neun Bilder werden erst angezeigt, wenn dies durch eine Tastatur-Taste veranlasst wird.
- Es ist möglich, bereits klassifizierte Bilder anzusehen und die Klassifizierung zu ändern.

5. Lösungsgenerierung

Die in Kapitel 4.1.2 beschriebene Vorgehensweise wird durchgeführt und im Folgenden dargestellt.

5.1. Rehkitzerkennung im Wildretter-Projekt

Geschäftsproblem Das in dieser Arbeit entwickelte neuronale Netz wird im Rahmen des Wildretter-Projekts eingesetzt. Das Wildretter-Projekt hilft, Felder nach Rehkitzen abzusuchen und aufzuspüren. Es soll verhindert werden, dass Rehkitze von landwirtschaftlichen Maschinen, sehr häufig von Mähmaschinen, erfasst werden. Das neuronale Netz wird eingesetzt, Rehkitze auf Thermalbildern zu erkennen. Diese Bilder werden mit einer Kamera auf einem UAV aufgenommen.
Durch den Einsatz eines neuronalen Netzes kann die Suche nach Rehkitzen automatisiert werden. Im Flug kann das neuronale Netz das aufgenommene Bild in Echtzeit beurteilen und die vermuteten Fundstellen markieren.
Die Arbeit von Michael Sorg [46] bestimmt bereits eine Architektur eines neuronalen Netzwerkes mit der Java Bibliothek DeepLearning4J. Das erstellte Netz kann nicht direkt auf dem UAV eingesetzt werden, da auf dem UAV die Programmiersprache Python verwendet wird. Diese Arbeit wird ein neuronales Netz in Python umsetzen und dabei die vorhandene Architektur als Grundlage nehmen und weiterführend das Netz auf verschiedene Probleme (vgl. Kapitel 3.5) untersuchen.
Seit der Arbeit von Michael Sorg hat sich das Ziel nicht verändert. Das Ziel ist primär die sichere automatisierte Erkennung von Rehkitzen. Sekundär sollte die Anzahl an Fehlalarmen reduziert werden (siehe Aufgabenstellung 2).

Analyse der Situation Für die Bearbeitung der Aufgabe stehen eine Reihe von Datensätzen zur Verfügung, die bereits die im überwachten Lernen notwendigen gekennzeichneten Daten enthalten. Aufgabenstellung 1 bestimmt, dass noch weitere Datensätze verwendet werden sollen. Diese müssen im Rahmen dieser Arbeit gesammelt und gekennzeichnet werden. Welche Datensätze vorhanden sind, wird in

Abschnitt 5.2 näher erläutert.
Weitere Ressourcen liegen in Form von Hard- und Software bereit. Bereits vorhanden ist ein Rechner mit GPU (GeForce GTX 1060 3GB/PCIe/SSE2) für ein effizientes Training. Außerdem ist Python-Software `Poitagger` und `Poitagger 2` verfügbar. Diese Software ist in der Lage, die Flugdaten und GPS-Positionen der Fundstellen visuell darzustellen. Weitere Ressourcen werden nicht benötigt.
Eine Annahme ist, dass die neuen unbekannten Daten in einer ähnlichen Form vorliegen wie die verwendeten Trainingsdaten. Dies bedeutet, dass Bildauflösung und Flughöhe des UAV sich nicht wesentlich von den Trainingsdaten unterscheiden.
Ein Risiko im Zusammenhang mit neuronalen Netzen ist, dass die ermittelte Performance auf neuen unbekannten Daten nicht erreicht werden kann. Für diese Arbeit bedeutet dies, dass Rehkitze übersehen werden.

Daten-Modellierungs-Problem Das neuronale Netz soll mit Python-Tools entwickelt werden (siehe Aufgabenstellung 2). Zwei weit verbreitete Tools Tensorflow und Keras sind in Kapitel 3.6 näher erläutert. Das Kapitel 3 enthält außerdem die Beschreibung der relevanten Algorithmen und Methoden im maschinellen Lernen.
Es wird festgelegt, dass das Projekt als erfolgreich gilt, wenn ein Recall von 90 % erreicht wird. Die Precision soll unter dieser Voraussetzung so hoch wie möglich sein. Der Recall gibt an, wie viele Rehkitze übersehen werden, während die Precision bestimmt, wie viele Fundstellen fälschlicherweise als Rehkitze klassifiziert werden. Der Recall ist wichtiger als die Precision, da es für die Rehkitzrettung entscheidend ist, keine Rehkitze zu übersehen.

5.2. Verständnis der Daten

Datenerfassung Im überwachten Lernen werden gekennzeichnete Bilder benötigt, um ein neuronales Netz zu trainieren. Innerhalb dieser Arbeit werden Daten von Wildretter-Befliegungen verwendet. Während dieser Befliegungen fliegt ein UAV mit einer Thermalkamera über ein Feld und nimmt Bilder auf, die die Rohdaten für das Training des neuronalen Netzes bilden.

Ein Flug beginnt mit Start des UAV und endet mit dessen Landung. Alle aufgenommenen Bilder während dieses Flugs werden in einem Ordner gespeichert, die Flüge eines Jahres einer Region zusammengefasst in einem weiteren Ordner. Dies wird in der folgenden Arbeit als Datensatz bezeichnet. Es stehen folgende Datensätze zur Verfügung:

- `2015_DISS`: Zu jedem Flug ist eine csv-Datei vorhanden, die die Kitzpositionen innerhalb der Bilder kennzeichnet.
- `2017_NAN`: Zu jedem Flug ist eine csv-Datei vorhanden, die die Kitzpositionen innerhalb der Bilder kennzeichnet.
- `2019_NAN`: Es sind keine Kitzpositionen gekennzeichnet.
- `2019_Pelz`: Es sind keine Kitzpositionen gekennzeichnet.
- `2020_NAN`: Es sind keine Kitzpositionen gekennzeichnet.
- `2020_Erfde`: Die GPS-Posititionen und der Zeitpunkt der Kennzeichnung einer Kitz-Fundstelle sind vorhanden.

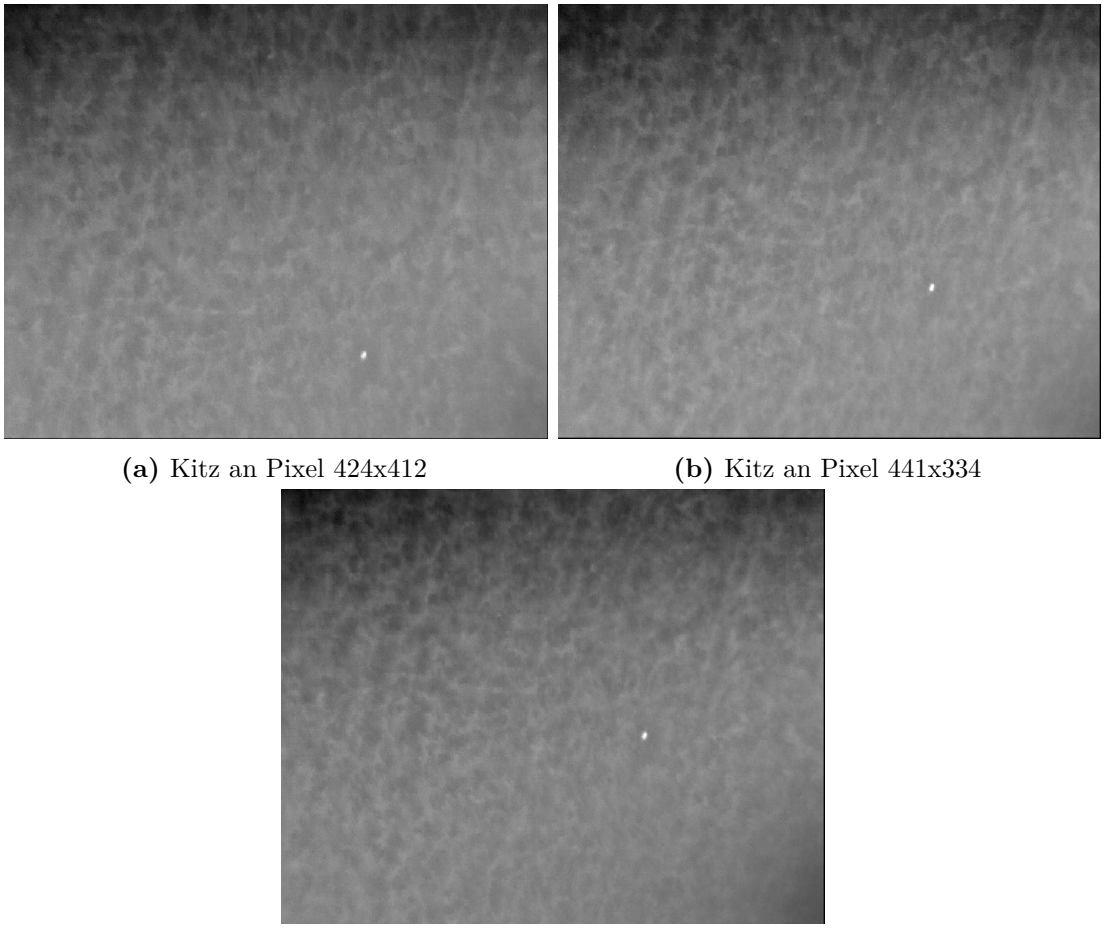
Aus den Jahren 2015 bis 2020 sind noch weitere Datensätze verfügbar, die jedoch aufgrund fehlender Informationen über die tatsächlichen Kitzpositionen nicht für diese Arbeit verwendet werden. Für `2019_NAN`, `2019_Pelz` und `2020_NAN` gilt das zwar auch, hier ist es aber möglich visuell zu entscheiden, ob es sich auf den Bildern um Kitze handelt.

Datenbeschreibung Die Rohdaten bestehen aus zwei Teilen, einem Header und einem Body. Der Header enthält Metainformationen über den Flug, wie beispielsweise Latitude und Longitude des UAV zum Zeitpunkt der Bildaufnahme. Im Body sind die Pixelwerte mit 16 Bit Auflösung gespeichert. Jedes Bild besitzt eine Auflösung von 640x512 Pixeln.
Die aufgenommenen Bilder zeigen folgende Eigenschaften:

- Das selbe Tier kann auf mehr als einem Bild zu erkennen sein.
- Auf einem Bild kann gar kein Tier zu erkennen sein.

- Auf einem Bild können mehr als ein Tier zu erkennen sein.

Abbildung 5.1 zeigt drei nacheinander aufgenommene Bilder mit einem Kitz. Das selbe Tier ist auf allen drei Bildern zu erkennen. Abbildung 5.2 zeigt eine Aufnahme, auf der kein Kitz zu erkennen ist. In Abbildung 5.3 sind zwei Kitze zu sehen.
Zur Darstellung von Beispielbildern in den Abbildungen 5.1 bis 5.3 werden die 16 Bit Rohdaten auf 8 Bit durch Normalisierung reduziert.



(a) Kitz an Pixel 424x412 **(b)** Kitz an Pixel 441x334

(c) Kitz an Pixel 429x289

Abbildung 5.1.: Bilder von einem Rehkitz aus dem Datensatz 2015 auf drei aufeinanderfolgenden Bildern

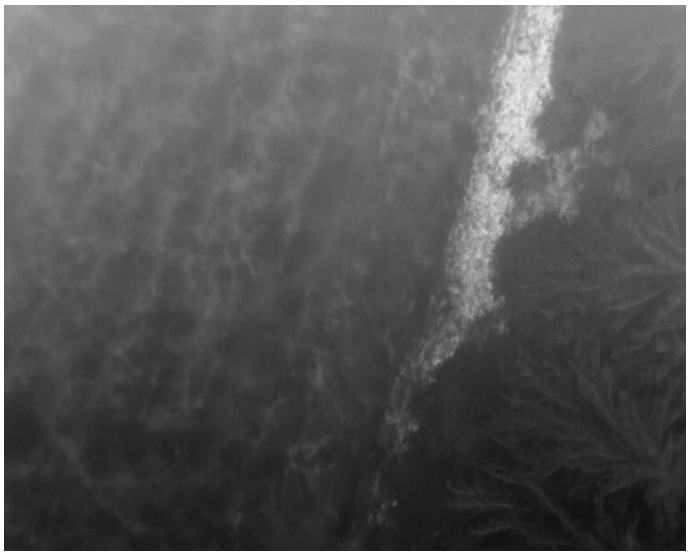


Abbildung 5.2.: Bild aus dem Datensatz `2019_Pelz` ohne Kitz

Abbildung 5.3.: Bild aus dem Datensatz `2017_NAN` mit zwei Kitzen (Pixel 436x85 und 175x497)

Datenanalyse Die beiden Datensätze `2015_DISS` und `2017_NAN` besitzen eine csv-Datei zur Zuordnung von Kitzpositionen in den Bildern. Für die Analyse der restlichen vier Datensätze muss zuerst eine Zuordnung von Kitzpositionen in den Bildern vorgenommen werden.
Die Datensätze `2019_NAN`, `2019_Pelz` und `2020_NAN` werden visuell abgesucht und gekennzeichnet. Ist nach Meinung des Autors auf einem Bild ein Kitz zu erkennen, werden die x- und y-Koordinaten und der Dateiname notiert. Dieser Prozess wird unterstützt durch ein eigen entwickeltes Programm. Die Bilder werden eingelesen und angezeigt. Alle wichtigen Informationen über eine Fundstelle werden automatisch bei einem Mausklick in eine Datei geschrieben.
Für den Datensatz `2020_Erfde` sind nur die GPS-Positionen der Kitze vorhanden. Die Zuordnung der GPS-Positionen zu den Bildern wird durch das von Martin Israel entwickelte Programm `Poitagger 2` unterstützt. Dieses Programm liest anhand der Metadaten der Bilder den Flugpfad des UAV aus und überträgt die GPS-Positionen der Kitze auf die Bilder. Diese Übertragung ist nicht genau, da die ermittelte Kitz-Position mehrere Pixel von der tatsächlichen Position entfernt ist. Diese Abweichung wird manuell korrigiert.
Tabelle 5.1 zeigt die Anzahl an Bildern pro Datensatz. Zusätzlich werden alle gefundenen Bilder mit Tieren und die Tiere insgesamt aufgezählt.

	Bilder insgesamt	Bilder mit Tieren	Tiere insgesamt
2015_DISS	13762	1112	1356
2017_NAN	26443	717	797
2019_NAN	9518	400	499
2019_Pelz	12180	514	644
2020_NAN	24665	445	470
2020_Erfde	45804	563	946
$\sum$	132372	3751	4712

Tabelle 5.1.: Anzahl der Bilder in jedem Datensatz

Datenqualität Es wird visuell überprüft, ob alle Kitzpositionen in den Bildern erkannt wurden. Für die Überprüfung wird `Poitagger 2` eingesetzt. Anhand des dargestellten Flugpfades des UAV in `Poitagger 2` ist es leichter zu erkennen, ob das gefundene Kitz später im Flug nochmal auf einem der Bilder vorkommt.
Mit einer visuellen Überprüfung wird außerdem untersucht, ob die Positionen zentral getroffen werden.

5.3. Datenvorbereitung

Datenselektion Ein Flug beginnt mit dem Start des UAV und endet mit dessen Landung. Während des Fluges hat das UAV meist eine Höhe von 80 Metern. Für das Training werden die Daten nicht anhand der Höhe aussortiert. Es ist aber zu erkennen, dass die meisten Bilder von Kitzen in einer Höhe von 70-100 Metern gemacht werden (siehe Tabelle 5.2). Eine Selektion findet nur anhand des Winkels der Kamera statt. Dieser Winkel lässt sich aus den Metadaten im Header der Rohdaten auslesen. Es werden nur Bilder für das Training verwendet, die einen Kamerawinkel von 60° bis 120° haben. Ein Wert von 90° bedeutet, die Kamera sieht senkrecht nach unten.

Datenaufbereitung und Datenintegration Für das Trainieren des neuronalen Netzes werden keine Bilder mit 640*512 Pixeln verwendet. Michael Sorg [46] erkennt, dass die Bilder in kleinere Teile zerlegt werden sollten. Diese „kleineren Teile“ werden innerhalb dieser Arbeit als Patches bezeichnet und haben eine Größe von 64x64

	0-35m	35-70m	70-100m	ab 100m
2015_DISS	224	160	972	0
2017_NAN	0	92	701	0
2019_NAN	14	42	439	4
2019_Pelz	2	32	610	0
2020_NAN	0	92	378	0
2020_Erfde	0	257	655	34

Tabelle 5.2.: Anzahl der Kitzbilder in jedem Datensatz je nach Höhe

Pixeln.
Für Bilder ohne Tier wird das Bild in 80 Teile mit der Größe von 64x64 Pixeln zerlegt. Dies ist in Abbildung 5.4 zu sehen.
Bilder mit Tieren werden auf eine andere Art und Weise zerlegt. Die erzeugten Patches überlappen sich. Das 64x64 Fenster wird nicht immer um 64 Pixel verschoben, sondern um 8. Dadurch entstehen mehr Bilder, die Tiere enthalten.
Diese Schrittweite von 8 bezieht sich nur auf den Teil des Bildes, auf dem eine Kitzposition markiert ist. Eine solche Kitzposition ist in Abbildung 5.5 zu sehen. Das restliche Bild wird mit einer Schrittweite von 64 Pixeln in Patches unterteilt.
Diese Zerlegung in Patches wird durch ein bereits existierendes Python-Skript durchgeführt.
Tabelle 5.3 stellt dar, wie viele Patches in den jeweiligen Datensätzen ein Tier enthalten und wie viele nicht. Die Patches aller Datensätze werden gemeinsam in einer Datei mit dem HDF5-Speicherformat gespeichert. Lim et al [29] erkennt durch die Nutzung eines HDF5-Speicherformates einen Performance Vorteil bezüglich der Trainingszeit eines neuronalen Netzes. Teil dieser Arbeit ist es nicht, zu vergleichen, ob auch wirklich ein Vorteil vorliegt.
In der HDF5-Datei werden die Patches als 8 Bit unsigned Integer gespeichert. Vor dem Speichern werden die Rohdaten auf 8 Bit durch Normalisierung reduziert und dann in Patches unterteilt. Damit hat jedes Pixel einen Wert zwischen 0 und 255. Es

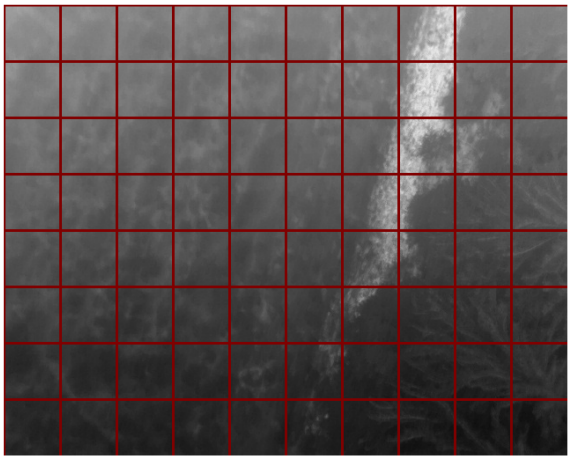
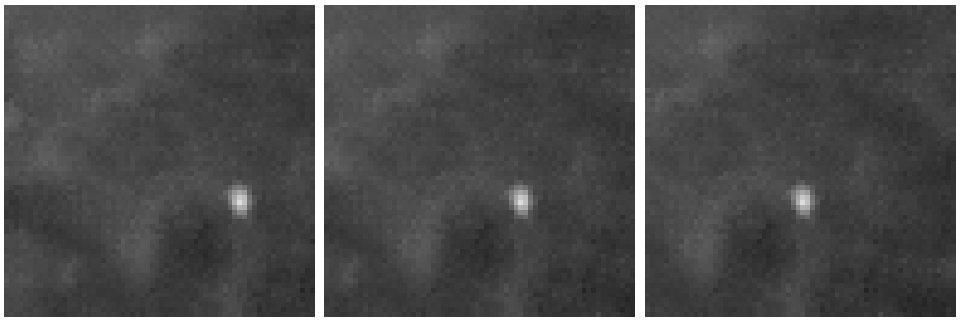


Abbildung 5.4.: Einteilung eines Bildes in Patches



(a) (b) (c)

Abbildung 5.5.: Bilder von einem Rehkitz aus dem Datensatz `2015_DISS` auf drei aufeinanderfolgenden Patches

	Tiere	ohne Tier
2015_DISS	55955	1003890
2017_NAN	31124	1357656
2020_NAN	19076	1484714
2019_NAN	18305	692859
2019_Pelz	25039	885704
2020_Erfde	35850	3331635
Σ	185.349	8.756.155

Tabelle 5.3.: Anzahl an Patches in den Datensätzen

ist üblich, die Daten auf einen Wertebereich von [0, 1] zu skalieren. Hierfür werden die Pixelwerte aus der HDF5-Datei beim Einlesen ins neuronale Netz durch 255 dividiert.

5.4. Auswahl eines Modells

Für die Auswahl eines Modells werden die Daten der Datensätze `2015_DISS` und `2017_NAN` verwendet. Zum Zeitpunkt der Bearbeitung waren die weiteren Datensätze noch nicht vorhanden. Es wird eine Aufteilung in Trainings- und Testdaten anhand der Flüge vorgenommen. Da ein Tier auf mehreren Bildern zu erkennen ist, wird durch diese Aufteilung sichergestellt, dass ein Tier nicht sowohl im Training als auch beim Test vorkommt. Im Anhang C.1 ist aufgelistet, welche Flüge für das Training

und welche für das Testen verwendet werden.
Für diese Arbeit wird ein Keras-Modell erstellt und durch Parametervariationen an die im jeweiligen Abschnitt vorgestellten Methoden angepasst.

5.4.1. Ausgangsmodell

Michael Sorg [46] stellt eine Architektur zur Erkennung von Rehkitzen in Bildern vor. Dieses Modell wurde in ein Keras-Modell übertragen und ist im Anhang B dargestellt. Abbildung 5.6 zeigt das Netz visuell.
Die Eingangsdaten haben eine Form von 64x64 Pixeln. Es folgt eine Faltungsschicht mit 16 Filtern der Größe 5x5. Die Max-Poolingschicht hat eine Größe von 2x2. Die zweite Faltungschicht besitzt 32 Filter der Größe 3x3 und wird gefolgt von einer Max-Poolingschicht der Größe 2x2. Für die dritte Faltungsschicht werden 64 Filter der Größe 3x3 verwendet. Die Max-Poolingschicht hat eine Größe von 2x2.
In zwei vollvermaschten Schichten gibt es jeweils 8 Aktivierungseinheiten. Die Ausgabeschicht besitzt 2 Aktivierungseinheiten für die Klassen „Tier “ und „kein Tier “.
Wie auch schon in der Arbeit von Michael Sorg, wird dieses Netz 30 Epochen mit der Trainingsmenge der Datensätze `2015_DISS` und `2017_NAN` trainiert. Für jede Epoche

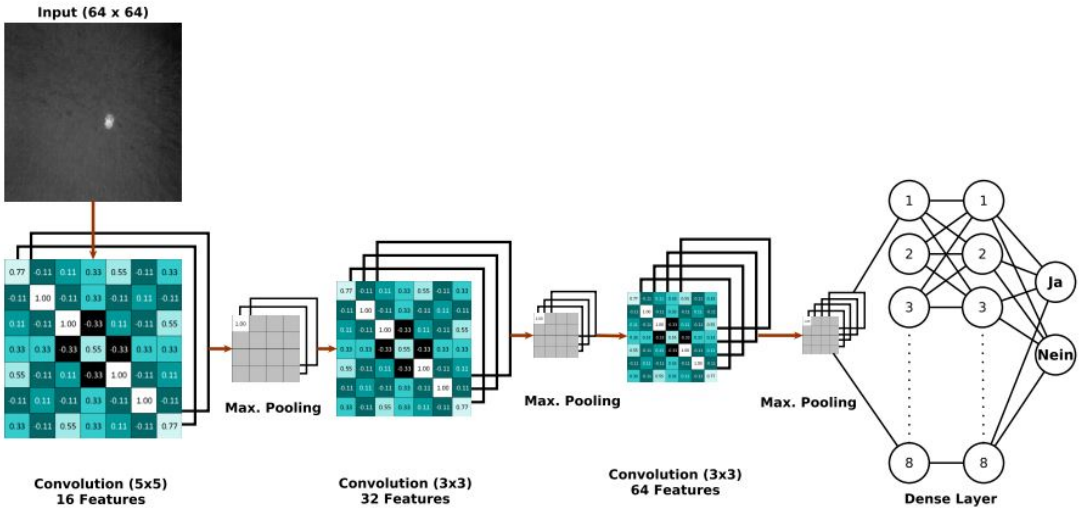


Abbildung 5.6.: Ausgangsarchitektur des neuronalen Netzes [46])

wird Recall und Precision anhand der Testmenge der Datensätze `2015_DISS` und `2017_NAN` ermittelt. Das Ergebnis ist in Abbildung 5.7 dargestellt.
Der Recall liegt auf einem niedrigen Niveau, während die Precision hoch ist. In Epoche 14 erreicht der Recall seinen niedrigsten Wert von 0,2611. Der höchste Wert von 0.6707 liegt in Epoche 4 vor. In Epoche 4 hat die Precision den niedrigsten Wert von 0.6414. Der Wert 0.8539 in Epoche 16 ist der höchste Wert.
Der Recall ist zu niedrig, um das Erfolgskriterium zu erfüllen. Dieses Netz würde Rehkitze nicht zuverlässig erkennen.
Zum Vergleich wird in Abbildung 5.8 Recall und Precision des Trainingsdatensatzes gezeigt. Nach Epoche 3 erreichen beide Metriken Werte über 0,9. Das Ergebnis der Trainingsdatenmenge ist demnach deutlich besser als das Ergebnis der Testdatenmenge. Wie in Kapitel 3.5.2 beschrieben, deutet dies auf eine Überanpassung hin.

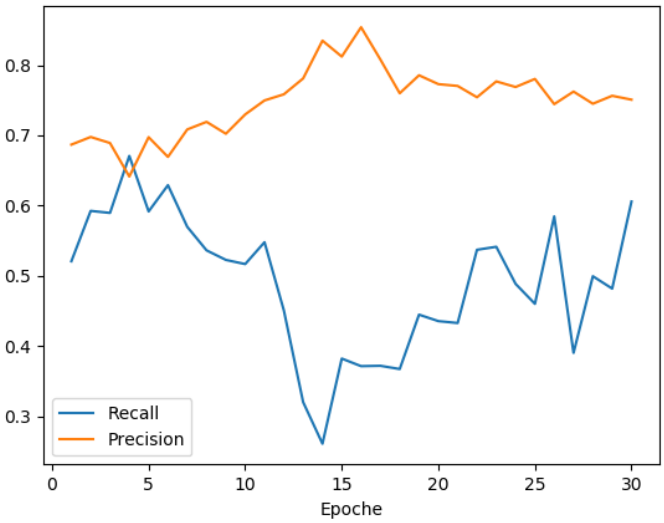


Abbildung 5.7.: Verlauf von Precision und Recall des Testdatensatzes beim Training mit dem kompletten Datensatz

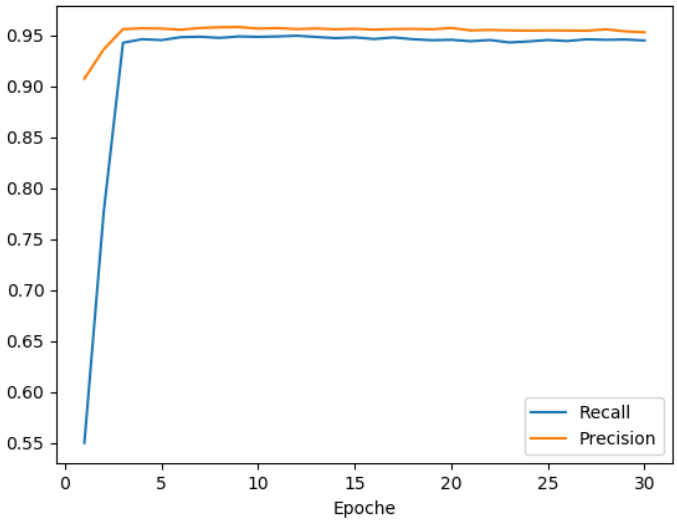


Abbildung 5.8.: Verlauf von Precision und Recall des Trainingsdatensatzes beim Training mit dem kompletten Datensatz

5.4.2. Umgang mit Unausgewogenheit

Wie in Kapitel 5.3 dargestellt, liegen 185349 Patches mit Tieren und 8756155 Patches ohne Tiere vor. Dies bedeutet eine unausgewogene Datenmenge, die nach Formel 3.13 einer Unausgewogenheit von $\rho = 27$ entspricht.
Das Thema der unausgewogenen Datenmengen wird in vielen Arbeiten aufgegriffen. Als Grundlage für diese Arbeit werden die Quellen [37], [14], [24] und [21] herangezogen.
Folgende Methoden im Umgang mit Unausgewogenheit wurden ausgewählt und werden anschließend erläutert:

- Ausbalancierung der Datenmenge
- Two-Phase Learning
- Dynamic-Sampling
- Focal-Loss
- Mean False Error

Diese Methoden werden nun im einzelnen vorgestellt und die darin vorgeschlagenen

Netze mit 30 Epochen trainiert.

Ausbalancierung der Datenmenge Damit eine Datenmenge ausbalanciert wird, gibt es die Möglichkeit des Random Under Sampling (RUS) und des Random Over Sampling (ROS). Im Under Sampling wird die Datenmenge der überrepräsentierten Klasse verkleinert. Im Gegensatz dazu vergrößert das Over Sampling die Datenmenge der unterrepräsentierten Klasse. [21, S. 7]
Zu beachten ist, dass ROS zu Overfitting führen kann und RUS die Menge an Daten verringert, wodurch möglicherweise Informationen verloren gehen [26, S. 3]. Over Sampling kann auch durch das künstliche Generieren von Daten erziehlt werden. In Bilddaten kann dies aber zu unrealistischen, synthetischen Bildern führen [26, S. 3].
Abbildung 5.9 zeigt den Verlauf von Recall und Precision unter Verwendung einer ausbalancierten Datenmenge. Es werden zufällig Patches aus der nicht-Kitz-Datenmenge ausgewählt, sodass gleich viele Elemente vorhanden sind wie in der Kitz-Datenmenge ($\rho = 1$).
Mit dieser Methode kann ein hoher Recall erreicht werden. Die Precision hingegen liegt auf einem niedrigen Niveau. Der Recall verändert sich ab Epoche 4 mit dem Wert von 0,98 kaum. Die Precision ist mit einem Maximum von 0,6327 in Epoche 13 und einem Minimum von 0,4183 in Epoche 21 weniger konstant.
Da die Verwendung einer ausgewogenen Datenmenge ($\rho = 1$) die nicht-Kitz-Datenmenge stark verkleinert, wird ein weiteres Netz trainiert, bei dem 10-mal so viele nicht-Kitz-Daten enthalten sind wie Kitz-Daten ($\rho = 10$). Die Ergebnisse sind in Abbildung 5.10 dargestellt.
Dies konnte den Wert für die Precision erhöhen (Maximim: 0,7433 in Epoche 22 und Minimum: 0,6250 in Epoche 5). Der Recall sinkt aber unter das festgelegte Erfolgskriterium von 0,9. Das Maximum von 0,8780 in Epoche 5 kommt dem Kriterium sehr nahe. In Epoche 16 liegt aber ein Minimum von 0,4786 vor.
Das nächste Netz soll besser zwischen Kitz- und nicht-Kitz-Bildern unterscheiden. Es wird die Annahme getroffen, dass Bilder mit einer hohen Differenz zwischen dem minimalen und dem maximalen Pixelwert schwieriger zu klassifizieren sind, da sie Kitz-Bildern ähneln. Als Grundlage für diese Annahme dienen die Histogramme im Anhang A.

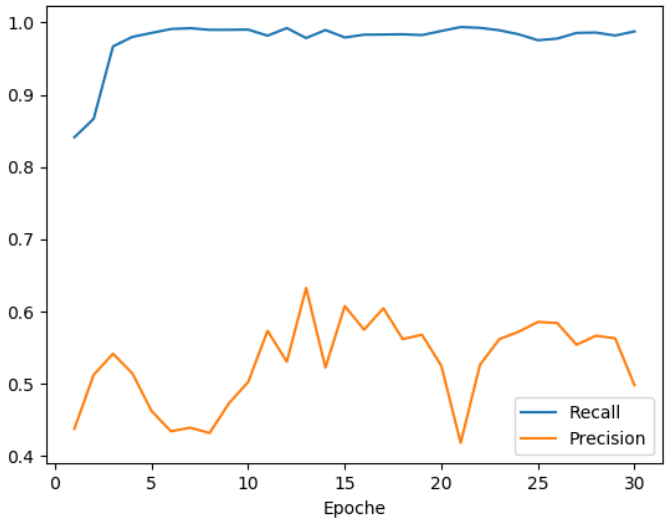


Abbildung 5.9.: Under Sampling der nicht-Kitz-Datenmenge auf ein 1 zu 1 Verhältnis mit der Kitz-Datenmenge ($\rho = 1$)

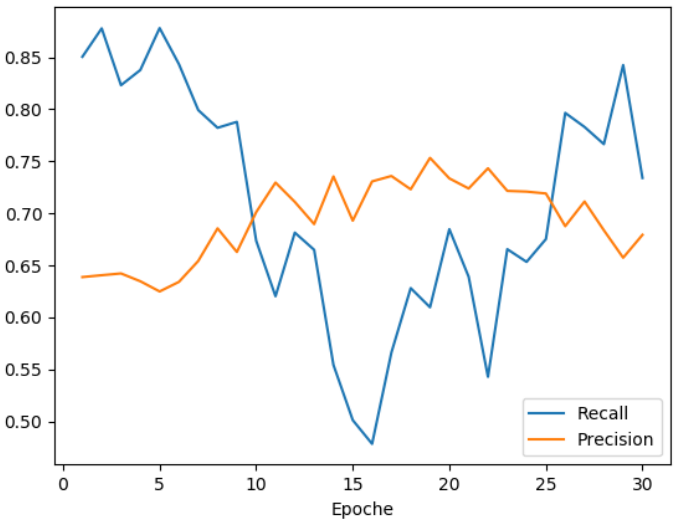


Abbildung 5.10.: Under Sampling der nicht-Kitz-Datenmenge auf ein 1 zu 10 Verhältnis mit der Kitz-Datenmenge ($\rho = 10$)

Je höher die Differenz zwischen dem minimalen und dem maximalen Pixelwert im Bild ist, desto wahrscheinlicher wird das Bild beim Under Sampling ausgewählt.
Abbildung 5.11 zeigt den Verlauf von Recall und Precision bei einem Verhältnis von $\rho = 1$. Abbildung 5.12 zeigt den Verlauf bei einem Verhältnis von $\rho = 10$.
Hiermit kann das Erfolgskriterium erfüllt werden, da ein Recall von mindestens 0,9 erreicht wird.
Für $\rho = 1$ liegt der Recall konstant über 0,97. In Epoche 20 erreicht der Recall sein Minimum von 0,9792, die Precision dagegen ein Maximum von 0,7554. In Epoche 2 liegt das Minimum bei 0,4747.
Für $\rho = 10$ kann eine Precision von 0,8569 in Epoche 16 erreicht werden. Das Minimum in Epoche 27 kommt nur auf einen Wert von 0,6730. Der Recall hat einen maximalen Wert von 0,9989 in Epoche 8 und einen minimalen Wert von 0,9394 in Epoche 2.
Auf ein ROS der Kitz-Datenmenge wurde verzichtet, da durch die Wahl der Schrittweite bei der Erstellung der Patches bereits die Kitz-Datenmenge vergrößert wurde (vgl. Kapitel 5.3).

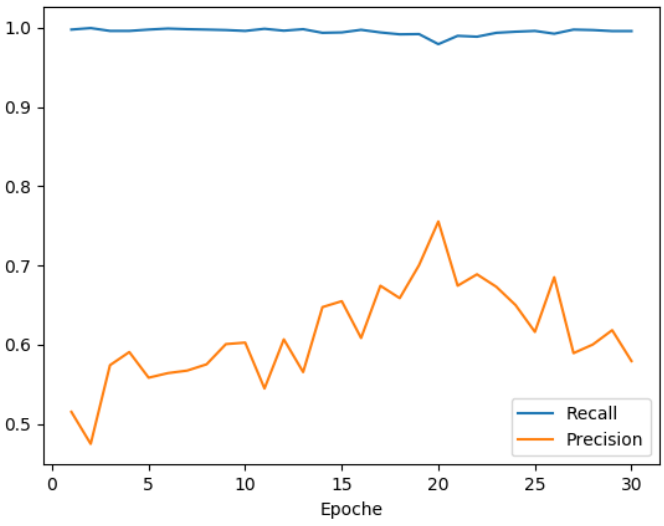


Abbildung 5.11.: Under Sampling der nicht-Kitz-Datenmenge auf ein 1 zu 1 Verhältnis mit der Kitz-Datenmenge anhand der min-max Differenz der Patches ($\rho = 1$)

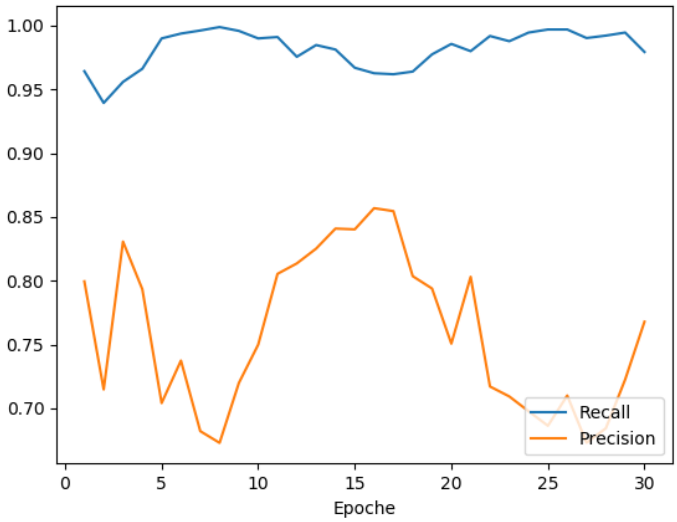


Abbildung 5.12.: Under Sampling der nicht-Kitz-Datenmenge auf ein 1 zu 10 Verhältnis mit der Kitz-Datenmenge anhand der min-max Differenz der Patches ($\rho = 10$)

Two-Phase Learning Im Two-phase learning nach Lee et al. [27] wird das neuronale Netz in zwei Phasen mit zwei Datensätzen trainiert. Ein Datensatz besteht aus der gesamten ursprünglichen Datenmenge. Dies bildet den `full`-Datensatz. Ein zweiter Datensatz besteht aus einer Untermenge des ursprünglichen Datensatzes. Hierfür wird ein Schwellenwert N festgelegt. Besitzt eine Klasse mehr als N Elemente, werden per Zufall N Elemente ausgewählt und dem neuen Datensatz hinzugefügt. Besitzt eine Klasse weniger als N Elemente, fließen alle Elemente in den neuen Datensatz ein. Dies bildet den `thresh`-Datensatz.
Das Netz wird in der ersten Phase mit dem verringerten `thresh`-Datensatz trainiert. Die zweite Phase sieht es vor, das Netz mit dem gesamten `full`-Datensatz zu trainieren.
Für das Ergebnis in Abbildung 5.13 wird das N so gewählt, dass sich gleich viele Kitz- und nicht-Kitz-Bilder im `thresh`-Datensatz befinden. Der `full`-Datensatz besteht aus der gesamten Trainingsdatenmenge.
Der Verlauf von Recall und Precision zeigt den Wechsel zwischen `thresh`-Datensatz und `full`-Datensatz. Bei den Phasen des `thresh`-Datensatzes liegt der Recall über

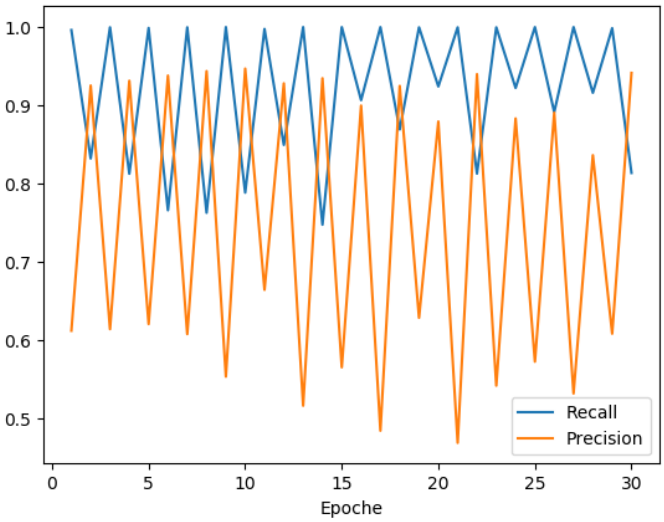


Abbildung 5.13.: Two-Phase Learning

0,99 in allen Epochen. Epochen 9, 13, 15, 17, 25 und 27 besitzen sogar einen Recall von 1. Die Precision liegt aber meist auf einem niedrigen Niveau, mit einem minimalen Wert von 0,4694 in Epoche 21 und einem maximalen Wert von 0,6647 in Epoche 11. Bei den Phasen des `full`-Datensatzes liegt der Recall ab Epoche 16 über einem Wert von 0,9 bei einer Precision über 0,84.
Das Erfolgskriterium wird erfüllt.

Dynamic-Sampling Eine weitere Möglichkeit, um ein neuronales Netz zu trainieren, ist das von Pouyanfar et al [33] vorgestellte Dynamic Sampling. Auch hier wird der Ansatz des Over- und Under-Sampling genutzt. Die überrepräsentierte Klasse wird mit Under Sampling verringert und die unterrepräsentierte Klasse mit Over Sampling vergrößert. In jeder Epoche erfolgt eine neue Bestimmung, auf welche Größe die Klassen vergrößert bzw. verkleinert werden sollen.
Zu Beginn erfolgt die Vergrößerung bzw. Verkleinerung aller Klassen auf einen bestimmten Wert N^*. Der Wert N^* ergibt sich aus der Gesamtmenge des Datensatzes, geteilt durch die Anzahl an Klassen. Es erfolgt ein Training des Netzes mit diesem Datensatz für eine Epoche. Daraufhin wird mit einem Testdatensatz das F1-Maß für

jede der Klassen ermittelt. Das F1-Maß bestimmt nach Formel 5.1 die Größe, auf die die Klassen vergrößert bzw. verkleinert werden.

$$UpdateSampleSize(F1, c_j) = \frac{1 - f1_j}{\sum_{c_k \in C}(1 - f1_k)} \cdot N^* \tag{5.1}$$

Eine spezifische Klasse c_j ermittelt die neue Klasse durch das Teilen der Differenz von 1 und dem $f1_j$-Wert der Klasse durch die Summe dieser Differenzen jeder Klasse C, dann multipliziert mit der durchschnittlichen Größe der Klassen N^*.
Die Klassen werden erneut over- bzw. undersampled und das Netz wird erneut trainiert.
Dies hat innerhalb dieser Arbeit nicht das gewünschte Ergebnis erzielt. Das Netz klassifiziert jedes Bild als kein-Kitz. Demnach ist das Erfolgskriterium nicht erreicht.

Focal-Loss Lin et al [30] zeigt eine alternative Loss-Funktion, die bei der Klassifizierung von unausgewogenen Klassen helfen soll.
Die Grundlage ist hier die Cross-Entropy ($CE(p, y)$), dargestellt in Formel 3.4 in Kapitel 3.2.2. Für den Focal-Loss wird die Cross-Entropy um einen Faktor $(1 - p_t)^\gamma$ ergänzt, dargestellt in Formel 5.2.

$$FL(p_t) = -(1 - p_t)^\gamma \cdot log(p_t) \tag{5.2}$$

Durch die Verwendung des Focal-Loss wird ein größerer Wert auf Elemente gelegt, die schwieriger zu klassifizieren sind. Bei einem kleinen p_t-Wert liegt der Faktor nahe 1 und das Ergebnis entspricht der Cross-Entropy. Bei einem großen p_t-Wert nahe 1 ist der Faktor nahe 0. Der p_t-Wert gibt die Wahrscheinlichkeit an, mit welcher das Objekt zu einer Klasse gehört. Ein großer p_t-Wert bedeutet, dass das Objekt aus Sicht des Netzes als einfach zu klassifizieren gilt. Demnach fällt mit dem Focal-Loss der Loss von einfach zu klassifizierenden Elementen weniger ins Gewicht.
Mit dem focusing parameter γ wird der Einfluss des Faktors gesteuert. Bei einem Wert von 0 liefert der Focal-Loss das gleiche Ergebnis wie die Cross-Entropy. Bei höheren Werten steigt der Einfluss des Faktors.
In Abbildung 5.14 wird dargestellt, wie sich Recall und Precision verhalten, wenn

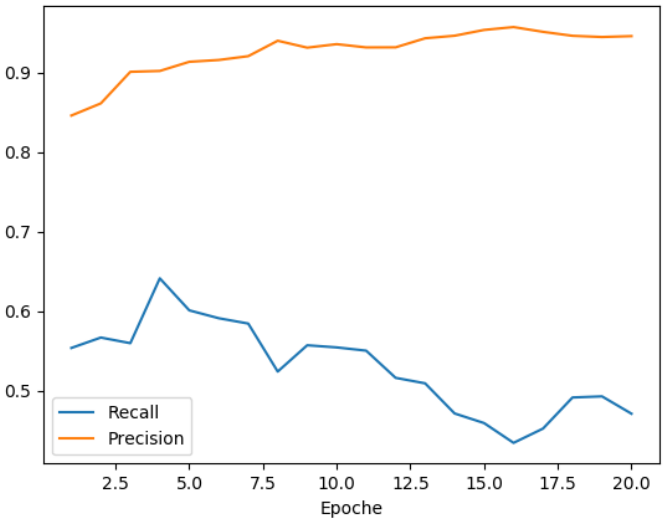


Abbildung 5.14.: Focal-Loss mit $\gamma = 2$

das Netz mit dem Focal-Loss trainiert wird. Dabei wird ein γ von 2 gewählt. Dieser Faktor wurde von Lin et al. [30, Kapitel 3] als bester Wert ermittelt.
Das Netz wird nicht wie die vorher vorgestellten Modelle für 30 Epochen trainiert, da bereits nach 20 Epochen ersichtlich ist, dass das Erfolgskriterium nicht erreicht wird. Der Recall kann nur ein Maximum von 0,6413 in Epoche 4 erreichen, ein Minimum von sogar nur 0,4343 in Epoche 17. Die Precision liegt ab Epoche 3 über 0,90 und erreicht das Maximum von 0,9574 in Epoche 16.

Mean False Error Der Mean False Error (MFE) basiert auf dem Mean Squared Error (MSE) nach Formel 5.3 und wird vorgestellt von Wang et al. [52]. Formel 3.3 zeigt den Squared Error.

$$MSE = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{2} (y^{(i)} - \hat{y}^{(i)})^2 \tag{5.3}$$

M ist die Anzahl an Objekten, $y^{(i)}$ die berechnete Klasse eines Objektes i und $\hat{y}^{(i)}$ die tatsächliche Klassenzugehörigkeit eines Objektes i.
Für den Mean False Error wird zwischen 2 Fällen unterschieden. Der Fehler nach

Formel 5.4 wird für alle Objekte ausgerechnet, die tatsächlich der negativen Klasse (N) angehören. Dies liefert den False Positive Error. Der False Negative Error nach Formel 5.5 wird mit allen Objekten ausgerechnet, die tatsächlich der positiven Klasse (P) angehören.
Der Mean False Error nach Formel 5.6 wird durch die Summe des False Positive Error und des False Negative Error ermittelt. Alternativ kann auch ein Mean Squared False Error (MSFE) bestimmt werden. False Positive Error und False Negative Error werden hier quadriert.

$$FPE = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2 \tag{5.4}$$

$$FNE = \frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2 \tag{5.5}$$

$$MFE = FPE + FNE \tag{5.6}$$

$$MSFE = FPE^2 + FNE^2 \tag{5.7}$$

Hier kann nicht das gewünschte Ergebnis erzielt werden, da alle Bilder als kein-Kitz klassifiziert werden.

Zusammenfassung Die Ergebnisse in diesem Abschnitt zeigen, dass nur das Ausbalancieren der Datenmenge und Two-Phase Learning das Erfolgskriterium erreichen können. Mit den anderen Methoden werden entweder gar keine oder zu wenige Tiere erkannt.
Außerdem kann folgender Zusammenhang beobachtet werden: Ist der Recall hoch, liegt eine niedrige Precision vor. Bei einer hohen Precision ist der Recall niedrig.
Nur RUS scheint ein Netz zu trainieren, dass die Tiere zuverlässig erkennt. Es liegt ein guter Wert für den Recall vor. Gleichzeitig ergeben sich mehr Fehldetektionen. Je stärker die überrepräsentierte Datenmenge verringert wird, desto höher wird der Recall und desto geringer die Precision.
Werden die Elemente anhand der Differenz zwischen minimalem und maximalem Pixelwert ausgewählt, steigt sowohl der Recall als auch die Precision. Das beste

Verhältnis von Recall und Precision liegt bei einem Verhältnis von $\rho = 10$ vor.
Im Two-Phase Learning können auch gute Ergebnisse erzielt werden. Bei einem Recall von 0,9223 erreicht die Precision einen Wert von 0,8834 in Epoche 24. Obwohl dies gute Ergebnisse sind, wird Two-Phase Learning für die Modellanpassung nicht weiter verwendet. Nach Meinung des Autors ist diese Vorgehensweise nicht stabil genug. Der Sprung von einem schlechten Recall von 0,7478 in Epoche 14 zu einem guten Recall von 0,9067 in Epoche 16 kommt sehr plötzlich. Es ist fraglich, ob dies auch bei einer anderen Datenmenge nochmal wiederholt werden kann.

5.4.3. Umgang mit Überanpassung

In Abschnitt 5.4.1 wird eine Überanpassung erkannt. Die Trainingsdaten haben sehr hohe Werte für Precision und Recall. Mit Testdaten können diese hohen Werte aber nicht erreicht werden.
Die verwendete Architektur setzt bereits eine Möglichkeit zur Verhinderung von Überanpassung um. Dies ist die L2-Regularisierung. Eine Regularisierung führt der Loss-Funktion einen weiteren Term hinzu, der zu hohe Gewichtungen verhindern soll. [35, Kapitel 3.3.5]
In der Literatur werden noch weitere Möglichkeiten genannt, eine Überanpassung zu verhindern. Wan et al [50] beschreibt die Methode Dropout und DropConnect, Wang et al [53] Dropout und Batch-Normalisation und Wu et al [55] die spezielle Methode für CNN Max-pooling Dropout.
Innerhalb dieser Arbeit werden nur Dropout und Batch-Normalisation verwendet, da diese beiden Methoden bereits in Tensorflow integriert sind [19] [18]. Die Funktionsweise und die Ergebnisse der beiden Methoden werden im Folgenden dargestellt. Bezüglich der Funktionsweise der beiden restlichen Methoden (DropConnect und Max-pooling Dropout) wird auf die jeweiligen Quellen verwiesen.
Das Training erfolgt mit der gesamten Trainingsdatenmenge. Die Netze werden unabhängig von den Ergebnissen aus Abschnitt 5.4.2 trainiert.

Dropout Dropout beschreibt ein Vorgehen, bei dem per Zufall Neuronen einen Wert von 0 annehmen. Sie werden „deaktiviert“. Über eine Dropout-Rate p wird festgelegt, wie viele Neuronen in einer Schicht deaktiviert werden [47, S. 5]. Die Deaktivierung von Neuronen erfolgt nur während des Trainings. Bei der Anwendung des Netzes auf unbekannte Daten sind alle Neuronen aktiv [35, Kapitel 15.2.2].
Die Wahl der Dropout-Rate p ist für die Performance entscheidend. Ist der Wert zu gering, kann die Überanpassung nicht verhindert werden und ein zu hoher Wert führt zu Unteranpassung [47, Kapitel A.4]. Ein weit verbreiteter Wert ist 0,5 [47, S. 5].
Abbildung 5.15 zeigt ein neuronales Netz mit Dropout und einer Dropout-Rate von $p = 0,5$. Deaktivierte Neuronen sind durch ein Kreuz markiert.
Für diese Arbeit wird die Dropout-Schicht wie in Quelle [9] nach der letzten Pooling-Schicht eingefügt. Das Ergebnis mit einem gewählten Dropout-Faktor von 0,5 ist in Abbildung 5.16 dargestellt.
Durch Verwendung von Dropout ist das Erreichen des Erfolgskriteriums möglich. Der Recall liegt ab Epoche 3 über einem Wert von 0,9185. In Epoche 25 kann ein Maximum von 0,9750 erzielt werden. Die Precision liegt bis auf drei Epochen (Epoche 25 mit dem Wert 0,8642, Epoche 26 mit dem Wert 0,8532 und Epoche 29 mit dem Wert 0.8658) über 0,9. Ein Maximum von 0,9655 liegt in Epoche 3 vor.
Im Vergleich zu den Werten der Trainingsmenge mit Recall und Precision von 0,96 kann die Aussage getroffen werden, dass eine Überanpassung verhindert wird.

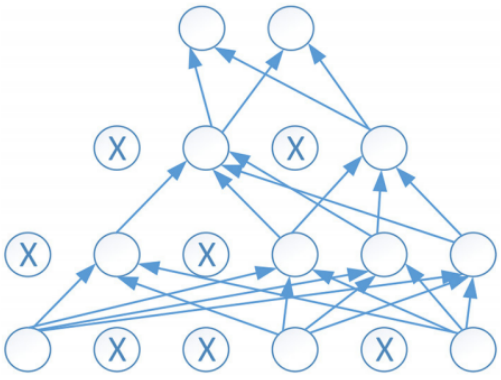


Abbildung 5.15.: Beispiel eines neuronalen Netzes mit Dropout und $p = 0,5$ [53, S. 4]

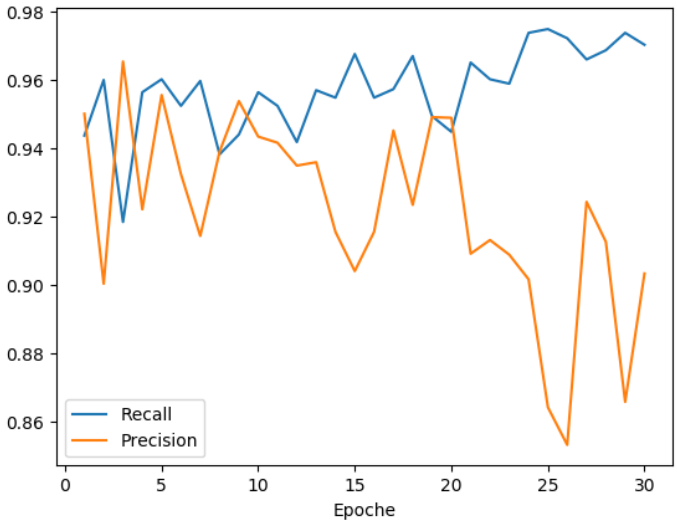


Abbildung 5.16.: Dropout mit $p = 0,5$

Batch-Normalisation Das Einfügen einer Batch-Normalisation-Schicht führt zu einer Normalisierung der Daten über eine definierte Submenge der Trainingsdaten, einem Mini-Batch.
Abbildung 5.17 zeigt eine solche Normalisierung zwischen zwei Schichten (W und $\hat{\mathcal{L}}$). σ ist die Varianz vom Mini-Batch, γ ein Skalierungsfaktor, β ein Offset und μ der Durchschnittswert im Mini-Batch.
Für eine genauere Beschreibung der Formel wird auf die Quelle [53, S. 5] verwiesen. Beschrieben ist dort auch, wie Varianz und Durchschnittswert im Mini-Batch bestimmt werden. Die Aktualisierung von γ und β erfolgt während des Trainings.
σ und μ werden nur im Training anhand des Mini-Batches bestimmt. Wird das Netz auf unbekannten Daten für eine Vorhersage angewandt, werden diese beiden Werte anhand von trainierten Statistiken bestimmt. [18]
Für diese Arbeit wird ein Batch von 64 verwendet. Das Ergebnis ist in Abbildung 5.18 abgebildet.
In vielen Epochen liegt ein Recall von über 0,95 vor. Das Erfolgskriterium kann aber nicht in jeder Epoche erreicht werden. In Epoche 11 fällt der Recall sogar auf ein Minimum von 0,3080. In dieser Epoche erreicht die Precision ihr Maximum von

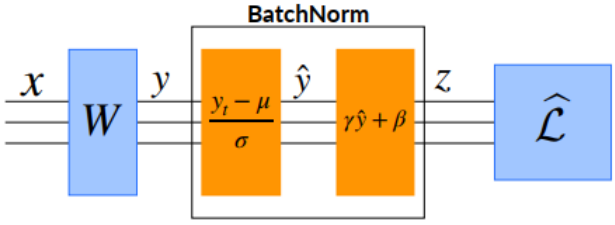


Abbildung 5.17.: Verwendung von Batch-Normalisation [39, Kapitel 3.3.]

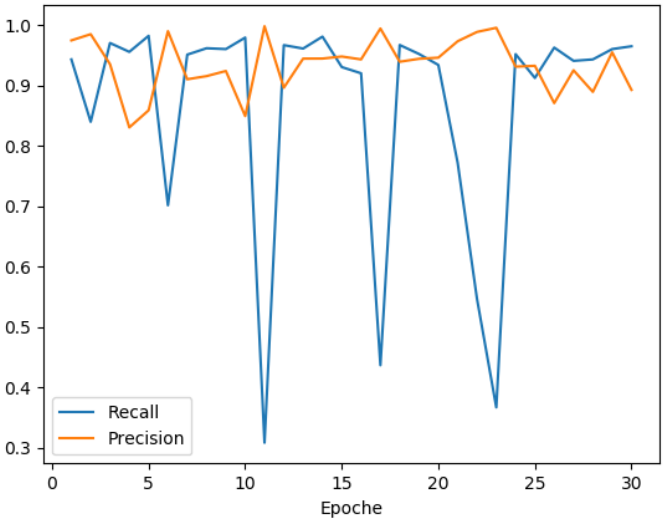


Abbildung 5.18.: Batch-Normalisation

0,9982. Der minimale Wert von 0,8305 liegt in Epoche 4 vor.
Im Vergleich dazu liefert die Trainingsmenge im Allgemeinen einen Recall von 0,99 und eine Precision von 0,96.

Zusammenfassung Beide vorgestellten Methoden können eine Überanpassung verhindern. In vielen Epochen erreichen Recall und Precision gemeinsam Werte über 0,9.
Dabei scheint die Verwendung von Batch-Normalisierung nicht stabil zu sein. In mehreren Epochen sinkt der Recall plötzlich sehr stark ab (Epoche 6: 0.7015, Epoche

11: 0.3080, Epoche 17: 0.4364, Epoche 22: 0.5467, Epoche 23: 0.3666). Ein solches Verhalten ist nicht zufriedenstellend.
Die Verwendung von Dropout kann in allen Epochen das Erfolgskriterium erfüllen und wird für die Modellanpassung in Abschnitt 5.5 verwendet.

5.4.4. Möglichkeiten der Datenmanipulation

Es werden noch zwei weitere neuronale Netze durch die Manipulation der Daten trainiert.
Für die erste Methode erfolgt keine Normalisierung der Rohdaten vor dem Erstellen der Patches. Demnach liegen die Pixelwerte nun als 16 Bit unsigned Integer vor. Beim Einlesen in das neuronale Netz werden die Daten mit einer Division durch 65535 auf einen Wertebereich von [0, 1] gebracht.
Das Erfolgskriterium kann nicht erreicht werden. Alle Objekte werden als kein-Kitz klassifiziert.
Die zweite Methode nutzt wieder die 8 Bit Daten. Es wird eine Histogrammspreizung auf die Daten angewandt. Hiermit werden die Pixelwerte eines Bildes gleichmäßig auf den Wertebereich [0, 255] verteilt. Dies dient der Kontrasterhöhung innerhalb eines Bildes. Die Dokumentation der verwendeten Operation ist in der Quelle [15] zu finden.
Auch hiermit kann das Erfolgskriterium nicht erreicht werden. Die Objekte werden als kein-Kitz klassifiziert.
Da diese beiden Modelle nicht die gewünschten Ergebnisse erzielen, werden keine weiteren Möglichkeiten der Datenmanipulation vorgenommen.

5.5. Modellanpassung

Anhand der Ergebnisse des letztes Abschnitts wird nun mithilfe der Kreuzvalidierung die optimale Kombination von Hyperparametern gesucht. Verwendung findet nur eine k-fache Kreuzvalidierung. Nach Wainer et al [49] ist das Durchführen einer

verschachtelten Kreuzvalidierung nicht unbedingt nötig und bedeutet nur einen erhöhten Rechenaufwand.
Die Möglichkeiten hier sind durch die verwendete Technik begrenzt. Je größer das k in der Kreuzvalidierung gewählt wird, desto öfters muss ein Netz trainiert werden. Da ein Netz mehrere Stunden braucht, um zu trainineren, ist es nicht möglich ein hohes k zu wählen.
Da in dieser Arbeit eine Aufteilung anhand der Flüge stattfindet, können bereits implementierte Lösungen wie beispielsweise `sklearn.model_selection.StratifiedKFold` [45] für die k-fache Kreuzvalidierung nicht verwendet werden. Diese Lösungen würden die Patches in k Gruppen aufteilen. Die einzelnen Schritte einer Kreuzvalidierung werden eigens implementiert.
Die gesamte Datenmenge wird für die Kreuzvalidierung verwendet und folgendermaßen aufgeteilt:

- 1/4 der Flüge werden zurückgehalten als Testmenge für die Evaluierung in Kapitel 5.6,
- die restlichen Flüge werden mit $k = 3$ in drei Teile aufgeteilt.

Im Anhang C.2 ist aufgeführt, welche Flüge in welcher Datenmenge vorkommen.
Folgende Hyperparameter werden verwendet:

- Verhältnis von Kitz- zu kein-Kitz-Daten (anhand der min-max Differenz der Patches):
  - 1:1
  - 1:2
  - 1:5
  - 1:10
- Dropout-Rate:
  - kein Dropout
  - 0.2
  - 0.5
- Learnrate:
  - 0.1
  - 0.01

– 0.001

Dies würde das Trainieren von 36 Netzen bedeuten. Da dies zeitlich zu aufwändig ist, wird nur eine zufällige Untermenge der Kombinationsmöglichkeiten trainiert. Die Untermenge besteht aus 18 Kombinationsmöglichkeiten.
Außerdem wird ein Netz nur 20 Epochen trainiert. Die Ergebnisse aus Kapitel 5.4 zeigen, dass mehr Epochen nicht unbedingt nötig sind. Anhand jeder fünften Epoche erfolgt die Evaluierung der Qualität des Netzes.
Die Ergebnisse sind in Tabelle 5.4 dargestellt. Der obere Wert in einem Feld zeigt den Recall und der untere Wert die Precision.

Tabelle 5.4.: Ergebnisse der Kreuzvalidierung

Epoche	**Kreuzvalidierung**				**Mittelwert**			
	5	**10**	**15**	**20**	**5**	**10**	**15**	**20**
Netz 1: 1:1 kein dropout learnrate:0.01	0.93	0.918	0.931	0.931				
	0.416	0.56	0.418	0.401				
	0.933	0.958	0.96	0.947	0.951	0.956	0.96	0.957
	0.493	0.299	0.275	0.421	0.368	0.355	0.32	0.349
	0.989	0.992	0.989	0.992				
	0.194	0.206	0.267	0.225				
Netz 2: 1:1 dropout: 0.2 learnrate:0.01	0.927	0.882	0.882	0.889				
	0.442	0.648	0.639	0.655				
	0.93	0.935	0.927	0.94	0.95	0.937	0.932	0.938
	0.506	0.259	0.507	0.367	0.376	0.381	0.463	0.419
	0.993	0.993	0.987	0.986				
	0.181	0.237	0.242	0.234				
Netz 3: 1:1 dropout: 0.5 learnrate:0.01	0.917	0.869	0.885	0.898				
	0.422	0.767	0.664	0.603				
	0.928	0.952	0.936	0.936	0.946	0.938	0.936	0.94
	0.573	0.275	0.533	0.327	0.411	0.422	0.493	0.394
	0.993	0.993	0.988	0.986				
	0.238	0.225	0.283	0.252				

Tabelle 5.4 weitergeführt: Ergebnisse der Kreuzvalidierung

Netz 4: 1:1 kein dropout learnrate:0.1	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0	0.333	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.006	0.0	0.0	0.0
	1.0	0.0	0.0	0.0				
	0.018	0.0	0.0	0.0				
Netz 5: 1:1 dropout: 0.5 learnrate:0.1	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
Netz 6: 1:1 dropout: 0.5 learnrate:0.001	0.915	0.914	0.921	0.92				
	0.534	0.612	0.521	0.493				
	0.927	0.933	0.939	0.92	0.944	0.946	0.951	0.945
	0.437	0.44	0.454	0.477	0.382	0.426	0.398	0.385
	0.991	0.992	0.993	0.995				
	0.175	0.227	0.22	0.184				
Netz 7: 1:2 kein dropout learnrate:0.01	0.887	0.882	0.887	0.893				
	0.7	0.708	0.664	0.602				
	0.938	0.934	0.923	0.91	0.937	0.919	0.931	0.931
	0.429	0.392	0.458	0.57	0.446	0.549	0.466	0.484
	0.986	0.942	0.982	0.989				
	0.209	0.546	0.275	0.281				
Netz 8: 1:2 dropout:0.2 learnrate:0.01	0.874	0.883	0.901	0.901				
	0.711	0.674	0.602	0.596				
	0.934	0.946	0.922	0.919	0.926	0.927	0.936	0.93
	0.511	0.451	0.587	0.415	0.586	0.558	0.498	0.44
	0.971	0.952	0.986	0.971				
	0.536	0.548	0.306	0.309				

Tabelle 5.4 weitergeführt: Ergebnisse der Kreuzvalidierung

Netz 9: 1:2 dropout:0.5 learnrate:0.01	0.872	0.905	0.888	0.884	0.926	0.931	0.93	0.932
	0.724	0.562	0.645	0.714				
	0.916	0.917	0.939	0.923	0.571	0.549	0.501	0.564
	0.708	0.69	0.333	0.653				
	0.989	0.971	0.963	0.99				
	0.282	0.395	0.526	0.325				
Netz 10: 1:2 dropout: 0.5 learnrate:0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
Netz 11: 1:2 dropout:0.5 learnrate:0.001	0.866	0.901	0.882	0.887	0.917	0.94	0.934	0.931
	0.707	0.562	0.662	0.635				
	0.918	0.94	0.93	0.922	0.566	0.514	0.545	0.563
	0.5	0.535	0.624	0.661				
	0.967	0.979	0.989	0.984				
	0.492	0.446	0.349	0.393				
Netz 12: 1:5 kein dropout learnrate:0.01	0.854	0.856	0.844	0.834	0.889	0.879	0.876	0.865
	0.814	0.799	0.857	0.869				
	0.879	0.872	0.859	0.819	0.683	0.684	0.696	0.753
	0.668	0.541	0.572	0.804				
	0.934	0.909	0.924	0.943				
	0.568	0.712	0.659	0.586				
Netz 13: 1:5 dropout: 0.2 learnrate:0.01	0.834	0.852	0.84	0.81	0.876	0.862	0.873	0.87
	0.859	0.833	0.845	0.899				
	0.85	0.823	0.843	0.89	0.68	0.786	0.716	0.696
	0.774	0.879	0.829	0.628				
	0.944	0.91	0.936	0.91				
	0.408	0.646	0.475	0.562				

Tabelle 5.4 weitergeführt: Ergebnisse der Kreuzvalidierung

Netz 14: 1:5 dropout: 0.5 learnrate:0.01	0.826	0.818	0.831	0.836				
	0.849	0.869	0.844	0.842				
	0.813	0.85	0.855	0.893	0.864	0.873	0.874	0.892
	0.834	0.787	0.717	0.614	0.732	0.736	0.716	0.676
	0.954	0.951	0.937	0.947				
	0.513	0.551	0.588	0.573				
Netz 15: 1:5 kein dropout learnrate:0.1	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0				
	0.0	0.0	0.0	0.0				
Netz 16: 1:5 dropout: 0.5 learnrate:0.001	0.815	0.825	0.83	0.837				
	0.849	0.85	0.85	0.836				
	0.85	0.865	0.885	0.906	0.87	0.885	0.894	0.906
	0.725	0.798	0.773	0.728	0.686	0.706	0.692	0.674
	0.946	0.966	0.967	0.974				
	0.483	0.469	0.452	0.458				
Netz 17: 1:10 kein dropout learnrate:0.01	0.7	0.727	0.732	0.714				
	0.945	0.936	0.93	0.941				
	0.855	0.81	0.761	0.695	0.821	0.782	0.777	0.764
	0.813	0.838	0.868	0.926	0.812	0.839	0.84	0.851
	0.908	0.808	0.838	0.882				
	0.678	0.743	0.723	0.685				
Netz 18: 1:10 dropout: 0.5 learnrate:0.01	0.694	0.682	0.781	0.812				
	0.954	0.946	0.897	0.863				
	0.786	0.858	0.847	0.737	0.751	0.784	0.813	0.791
	0.811	0.693	0.68	0.898	0.849	0.781	0.775	0.833
	0.773	0.812	0.811	0.823				
	0.783	0.704	0.749	0.737				

Es wird nicht auf jeden Wert in der Tabelle eingegangen, sondern nur die wichtigsten Erkenntnisse zusammengefasst:

- Eine Lernrate von 0,1 sollte nicht verwendet werden (Netz 4, 5, 10 und 15), da hier alle Objekte als nicht-Kitz klassifiziert werden.
- Eine Unausgewogenheit von $\rho = 10$ (Netz 17 und 18) erreicht zu geringe Werte für den Recall. Das Maximum liegt bei 0,821 in Epoche 5 von Netz 17.
- Eine Unausgewogenheit von $\rho = 1$ (Netz 1, 2, 3, und 6) erreicht die höchsten Werte für den Recall und erfüllt die Erfolgsbedingung. Dafür ist die Precision auf einem niedrigen Niveau, mit einem maximalen Wert von 0,493 (Netz 3, Epoche 15) und einem minimalen Wert von 0,32 (Netz 1, Epoche 15).
- Die Unausgewogenheit von $\rho = 2$ (Netz 7, 8, 9 und 11) erfüllt auch die Erfolgsbedingung, da der Recall über 0,9 liegt. Im Vergleich zu den Netzen mit $\rho = 1$ erreicht die Precision höhere Werte, mit einem maximalen Wert von 0,586 (Netz 8, Epoche 5) und einem minimalen Wert von 0,44 (Netz 8, Epoche 20).
- Wird eine Unausgewogenheit von $\rho = 5$ verwendet (Netz 12, 13, 14 und 16), kann das Erfolgskriterium fast erreicht werden, beispielsweise Recall von 0,892 (Netz 14, Epoche 20) und Precision von 0,676 (Netz 14, Epoche 20).

Auch wenn das Erfolgskriterium nicht ganz erreicht werden kann, wird Netz 14 als das beste Netz angesehen, da die Precision in den restlichen Netzen, die das Erfolgskriterium erfüllen, zu niedrig ist. Generell sind die Ergebnisse der vier Netze mit $\rho = 5$ (12, 13, 14 und 16) sehr ähnlich. Nach Meinung des Autors liefert aber Netz 14 das beste Verhältnis zwischen Recall und Precision.

5.6. Evaluation

Das im vorherigen Abschnitt ermittelte Netz wird mit 40 Epochen trainiert. In Abschnitt 5.6.1 werden die Ergebnisse des neuronalen Netzes dargestellt. Dies beinhaltet den graphischen Verlauf von Recall und Precision über die Epochen. Außerdem wird

die beste Epoche ausgewählt und eine Konfusionsmatrix ermittelt. Abschnitt 5.6.2 vergleicht die Ergebnisse der besten Epoche mit der visuellen Detektion.

5.6.1. Darstellung der Ergebnisse

Der Verlauf von Recall und Precision ist in Abbildung 5.19 dargestellt. In den Epochen 1, 11, 12, 13, 14, 22, 23, 25, 26, 27, 28, 30 und 32 liegt der Recall über 0,9 und erfüllt das Erfolgskriterium. Von diesen Epochen erreicht Epoche 30 den höchsten Wert für die Precision. In Tabelle 5.5 wird die Konfusionsmatrix für den gesamten Testdatensatz und die Epoche 30 dargestellt. Dies entspricht einem Recall von 0,9006 und einer Precision von 0,6898.
In den 104 Flügen der Testdatenmenge (vgl. Anhang C.2) liegen insgesamt 114 Fundstellen mit Tieren vor. 49 Fundstellen werden auf mindestens einem Patch nicht erkannt. Insgesamt 5 Fundstellen werden auf keinem Patch erkannt.

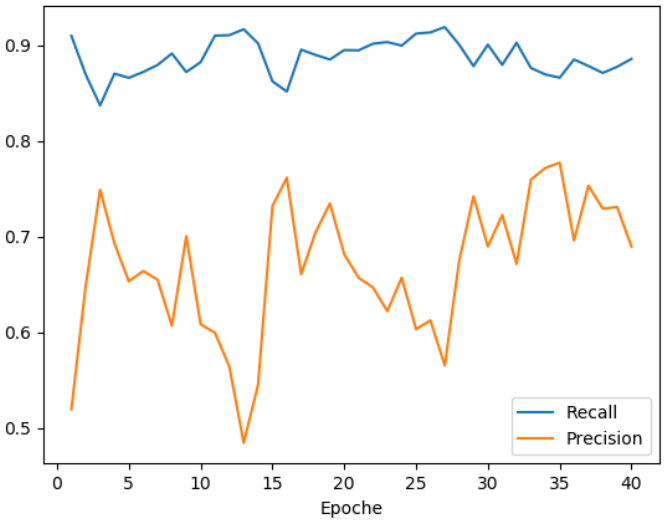


Abbildung 5.19.: Ergebnisse mit $\rho = 5$, Dropout-Rate 0,5 und Lernrate 0,01

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	41654	4598
	kein-Tier	18734	2232963

Tabelle 5.5.: Konfusionsmatrix des Testdatensatzes

5.6.2. Vergleich mit der visuellen Detektion

Die visuelle Detektion wird nach dem in Kapitel 4.2 vorgestellten Konzept durchgeführt.
Da für einen Flug mehrere tausende Patches vorliegen, kann nicht die gesamte Testdatenmenge verglichen werden, sondern nur die Ergebnisse der folgenden drei Flüge:

- `2015-06-02_08-39_Petz_Dachslochhang_2Kitze`
- `200521_0923_DH20-23`
- `200528_0823_AW01`

Die Auswahl der Flüge aus der Testdatenmenge erfolgt zufällig. Es wird aber darauf geachtet, dass im jeweiligen Flug auch wirklich Kitze vorkommen. Die durch das neuronale Netz ermittelten Konfusionsmatrizen der restlichen Flüge sind im Anhang D dargestellt.
Zur Verringerung der Datenmenge wird außerdem eine Vorauswahl getroffen. Alle Patches, bei denen die Differenz zwischen maximalem und minimalem Pixelwert kleiner als 100 ist, werden automatisch als kein-Tier klassifiziert und nicht zur visuellen Detektion angezeigt. Es kommt zu folgender Vorauswahl:

- `2015-06-02_08-39_Petz_Dachslochhang_2Kitze`: von 29834 Patches werden bereits 24422 aussortiert
- `200521_0923_DH20-23`: von 37592 Patches werden bereits 23302 aussortiert
- `200528_0823_AW01`: von 26857 Patches werden bereits 16835 aussortiert

Für das neuronale Netz findet diese Vorauswahl nicht statt.

2015-06-02_08-39_Petz_Dachslochhang_2Kitze: Die Konfusionsmatrix für diesen Flug ist in Tabelle 5.6 dargestellt.
Das neuronale Netz erkennt alle Tiere. Während der visuellen Detektion wird ein Patch mit Tier nicht als Tier erkannt. Die Precision für das neuronale Netz liegt bei 0,4845, da 50 Falsch Positive Patches vorliegen. Die visuelle Detektion erreicht eine Precision von 0,7302 mit nur 17 Falsch Positiven Patches.
Für diesen Flug ist die visuelle Detektion deutlich besser, da weniger Patches fälschlicherweise als Tier klassifiziert werden.

200521_0923_DH20-23: Tabelle 5.7 zeigt die Konfusionsmatrix für diesen Flug.
Auch hier erkennt das neuronale Netz alle Tiere auf den Patches. Dafür liegen 554 Falsch Positive Patches vor. Dies führt zu einer Precision von 0,2097.
Die visuelle Detektion kann 12 Patches nicht als Tier erkennen. Der Recall liegt daher bei 0,9184. Im Vergleich zum neuronalen Netz liegen weniger Falsch Positive Patches (142) vor. Demnach ist die Precision bei einem Wert von 0,4874.

200528_0823_AW01: Für diesen Flug zeigt Tabelle 5.8 die Konfusionsmatrix.
Alle 392 Patches mit Tieren werden mit dem neuronalen Netz erkannt. 472 Patches sind Falsch Positiv. Das entspricht einer Precision von 0,4537.
Mit der visuellen Detektion werden deutlich mehr Patches mit Tieren nicht erkannt. 215 Falsch Negative Patches führt zu einem Recall von 0,4515. Die Precision fällt mit einem Wert von 0,9219 deutlich besser aus. Hier liegen nur 15 Falsch Positive Patches vor.

Eine abschließende Wertung, ob das neuronale Netz bessere Ergebnisse liefert als eine visuelle Detektion wird nicht getroffen, da eine nicht ausreichend belastbare Anzahl an Datensätzen getestet wurde.

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	47	0
	kein-Tier	50	29784

(5.6.1) Neuronales Netz

	vorhergesagt	
	Tier	kein-Tier
Tier	46	1
kein-Tier	17	29817

(5.6.2) Visuell

Tabelle 5.6.: Konfusionsmatrix von `2015-06-02_08-39_Petz_Dachslochhang_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	147	0
	kein-Tier	554	37038

(5.7.1) Neuronales Netz

	vorhergesagt	
	Tier	kein-Tier
Tier	135	12
kein-Tier	142	37450

(5.7.2) Visuell

Tabelle 5.7.: Konfusionsmatrix von `200521_0923_DH20-23`

5.7. Verwendung im Wildretter-Projekt

Das trainierte neuronale Netz kann auf dem UAV eingesetzt werden. Dafür wird das Keras Modell in Form einer HDF5-Datei gespeichert, diese Speicherung wiederum von der Keras-API übernommen. Liegt eine solche HDF5-Datei auf dem UAV vor, kann das Modell geladen und für Vorhersagen verwendet werden. [17]
Bilder, die mit dem neuronalen Netz evaluiert werden, müssen in Form von Graustufenbildern mit einer Auflösung von 64x64 Pixeln vorliegen und auf einen Wertebereich von [0, 1] skaliert werden.
Diese Schnittstelle in Form einer HDF5-Datei ermöglicht ein einfaches Ersetzen des Modells, falls die Performance nicht zufriedenstellend ist. Die HDF5-Datei muss lediglich auf dem UAV ausgetauscht werden.

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	392	0
	kein-Tier	472	26395

(5.8.1) Neuronales Netz

	vorhergesagt	
	Tier	kein-Tier
Tier	177	215
kein-Tier	15	26842

(5.8.2) Visuell

Tabelle 5.8.: Konfusionsmatrix von `200528_0823_AW01`

Um festzustellen, ob ein Austausch nötig ist, kann die Performance bei jedem Flug evaluiert werden. Die Anzahl an falsch positiven Fundstellen wird notiert und auch das Übersehen von Rehkitzen. Aufgrund der Ergebnisse dieser Arbeit wird die Wahrscheinlichkeit eher gering gesehen, dass Rehkitze übersehen werden. Sollte es doch zu solchen Fällen kommen, muss das neuronale Netz ausgetauscht werden. Außerdem ist die Wahrscheinlichkeit gegeben, dass einige der Fundstellen keine Rehkitze enthalten. Sollte die Anzahl an falsch positiven Fundstellen zu hoch werden, wird ein Austausch des neuronalen Netzes als angemessen gesehen.

6. Zusammenfassung

In dieser Arbeit wurde ein neuronales Netz für die Erkennung von Rehkitzen auf Thermalbildern trainiert. Als Architektur wurde ein Convolutional Neural Network verwendet, welches die zur Verfügung stehende Datenmenge in „Tier“ und „kein-Tier“ klassifiziert.

Die von mehreren Flügen eines UAV gesammelten Daten werden in sechs Gruppen nach Jahr und Region aufgeteilt. Die Bilder der Flüge werden in Patches mit 64x64 Pixeln zerlegt und einzeln an das neuronale Netz übergeben. Es liegen mehr Patches ohne Tiere vor als mit Tieren.

Es wurde evaluiert welche Methoden sinnvoll sind, um die Performance des neuronalen Netzes zu verbessern. Die Performance wird anhand der beiden Metriken Recall und Precision bemessen. Dabei zeigen die Untersuchungen, dass das Entfernen von nicht-Tier-Patches aus der Trainingsdatenmenge den Recall verbessert und die Precision verschlechtert. Außerdem erweist sich die Dropout-Methode als die beste Möglichkeit, eine Überanpassung zu verhindern. Damit steigen die Recall-Werte und die Precision-Werte für die Testdatenmenge.

Anhand einer k-fachen Kreuzvalidierung wurden die Hyperparameter Dropout-Rate, Lernrate und Verhältnis der Ausgewogenheit der Klassen ρ in den Trainingsdaten angepasst. Mit einem k von 3 wurden jeweils 3 Netze mit unterschiedlichen Testdaten für 20 Epochen trainiert und ein Mittelwert gebildet. Anhand des Mittelwerts wurde evaluiert, welche Kombination von Hyperparametern die beste ist. Ergeben hat sich ein ρ von 5, eine Dropout-Rate von 0,5 und eine Lernrate von 0,01.

Das neuronale Netz erreicht einen Recall von 0,9006 und eine Precision von 0,6898 bei insgesamt 46252 Patches mit Tieren und 2251697 Patches ohne Tier.

Zusätzlich wurde ein Vergleich der maschinellen mit der visuellen Detektion, die sehr zeitaufwändig ist, durchgeführt. Aus Zeitgründen wurden deshalb von den 400 zur Verfügung stehenden Flügen nur 3 einzeln betrachtet. Für diese Flüge erreicht die visuelle Detektion einen geringeren Recall und eine höhere Precision als das neuronale Netz. Somit werden mehr Patches, die Tiere enthalten, übersehen und weniger Patches fälschlicherweise als Tier klassifiziert.

7. Reflexion und Ausblick

Die visuelle Detektion wurde vom fachlichen Betreuer Martin Israel vorgenommen, der sich langjährige Erfahrung mit der Detektion von Rehkitzen auf Thermalbildern mit einer Auflösung von 640x512 Pixeln angeeignet hat. Weitere Erfahrungen mit kleineren Patches von 64x64 Pixeln lassen Spielraum für ein noch besseres visuelles Ergebnis.

Weiter wurden die zur Verfügung stehenden Daten mit UAV aufgenommen, deren Kameras die gleiche Auflösung besaßen und in ähnlicher Höhen flogen. Somit kann hier nicht beurteilt werden, ob auch ein Tier richtig erkannt wird, wenn sich diese Eigenschaften ändern.

Jede maschinelle Tiererkennung ist zeitintensiv. In die Überlegungen, größere Netze zu verwenden, sollte einbezogen werden, dass dies möglicherweise noch mehr Zeitaufwand bedeutet.

In dieser Arbeit wird nur zwischen Tier und nicht-Tier unterschieden. Über diese simple Klassifizierung hinaus wäre es für die Untersuchung von Fundstellen sinnvoll, komplexere Klassen zu definieren. Die Tier-Datenmenge enthält neben Rehkitzen auch Rehe, deren Erkennung bei einer Unterteilung in Rehkitz, Reh und kein-Tier anstatt einer binären Klassifizierung möglich wäre. Es ist nicht notwendig Rehe auf die gleiche Weise wie Rehkitze zu klassifizieren, da Rehe, anders als Kitze, aufgrund ihres natürlichen Instinkts weglaufen und somit nicht der gleichen Gefahr ausgesetzt sind. Für diese drei Klassen sind genug Daten vorhanden, um die Qualität des Netzes ausreichend zu bewerten.

Die Evaluierung eines Netzes erfolgt auf Basis der beiden Metriken Recall und Precision. Es könnte sinnvoll sein, eine alternative Metrik zu entwickeln, die nicht auf Ebene der Patches ansetzt, da ein Tier nur auf einem Patch erkannt werden muss und nicht auf allen. Die Ergebnisse dieser Arbeit zeigen, dass je weniger Patches mit Tieren erkannt werden, desto mehr Patches werden fälschlicherweise als Tier klassifiziert. Müssen Rehkitze nicht mehr auf allen Patches erkannt werden, besteht die Möglichkeit, dass auch weniger oft eine falsch positive Fundstelle ermittelt wird. Eine alternative Metrik könnte pro Flug ermitteln, wie viele Tiere übersehen werden und diesen Wert über alle Flüge der Testdatenmenge mitteln.

Die Architektur des neuronalen Netzes birgt viel Potential für Anpassungen, Änderungen und Neuerungen. Beispielsweise könnten weitere Informationen aus den Rohdaten an das neuronale Netz übergeben werden. Derzeit ist dies nur auf die Bilddaten beschränkt. Durch die Integration der Zeit der Messung kann das Netz potentiell besser zwischen Tieren und falsch positiven Fundstellen entscheiden. Aufgrund der abweichenden Temperatur und Sonnenintensität in den frühen Morgenstunden im Vergleich zu der Mittagszeit ergibt dies leicht unterschiedliche Thermalbilder. Es sollte untersucht werden, ob durch einen solchen Ansatz die Gefahr besteht, dass Tiere übersehen werden, wenn beispielsweise nicht genug Datensätze vorliegen deren Messung zu einem späteren Zeitpunkt erfolgt ist.
Als eine weitere Änderung der Architektur könnte ein Recurrent Neural Network integriert werden [11]. Hier werden Bilder nicht mehr unabhängig voneinander klassifiziert, sondern anhand einer Folge von Bildern. Für eine solche Vorgehensweise ist eine weitere Analyse der Bilddaten dahingehend nötig, ob beispielsweise die Einteilung in Patches genutzt werden kann.
Die maschinelle Rehkitzdetektion birgt weitreichendes Forschungs- und Anwendungspotenzial. Beleuchtet wurde in diesem Kapitel ein vergleichsweise kleiner Teil der möglichen nächsten Schritte im Zusammenhang mit diesem Thema. Schon allein das Wildretter-Projekt lässt erahnen, wie nützlich weitere Forschungen auf diesem Gebiet sind.

Literatur

[1] Giuseppe Bonaccorso. *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[2] Jason Brownlee. *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Edition: v1.8. Machine Learning Mastery, 2016.

[3] Francois Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[4] Nvidia Corporation. *GPU-Accelerated TensorFlow*. URL: https://www.nvidia.com/en-sg/data-center/gpu-accelerated-applications/tensorflow/ (besucht am 23. 08. 2020).

[5] Marty Curry. *Helios Prototype*. URL: https://www.nasa.gov/centers/dryden/news/ResearchUpdate/Helios/ (besucht am 07. 09. 2020).

[6] Konstantinos Dalamagkidis, Kimon P Valavanis und Les A Piegl. *On integrating unmanned aircraft systems into the national airspace system: issues, challenges, operational restrictions, certification, and recommendations*. Bd. 54. springer science & Business Media, 2011.

[7] *Das Projekt WILDRETTER*. URL: http://www.wildretter.de/projekt-wildretter/wissenswertes.html (besucht am 08. 09. 2020).

[8] Vincent Dumoulin und Francesco Visin. „A guide to convolution arithmetic for deep learning“. In: *arXiv preprint arXiv:1603.07285* (2018). URL: https://arxiv.org/abs/1603.07285v2.

[9] Mohamed Elleuch, Rania Maalej und Monji Kherallah. „A new design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition“. In: *Procedia Computer Science* 80 (2016), S. 1712–1723. URL: https://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html.

[10] Antonio Gulli und Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[11] Yanming Guo u. a. „CNN-RNN: A large-scale hierarchical image classification framework“. In: *Multimedia Tools and Applications* 77.8 (2018), S. 10251–10271. URL: https://link.springer.com/article/10.1007/s11042-017-5443-x.

[12] Haibo He und Edwardo A Garcia. „Learning from imbalanced data“. In: *IEEE Transactions on knowledge and data engineering* 21.9 (2009), S. 1263–1284. URL: https://ieeexplore.ieee.org/abstract/document/5128907/.

[13] Kaiming He u. a. „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[14] Paulina Hensman und David Masko. „The impact of imbalanced training data for convolutional neural networks“. In: *Degree Project in Computer Science, KTH Royal Institute of Technology* (2015).

[15] *Histogram Equalization*. URL: https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html%5C (besucht am 02. 09. 2020).

[16] Google Inc. *GPU support*. URL: https://www.tensorflow.org/install/gpu (besucht am 23. 08. 2020).

[17] Google Inc. *Save and load Keras models*. URL: https://www.tensorflow.org/guide/keras/save_and_serialize (besucht am 04. 09. 2020).

[18] Google Inc. *tf.keras.layers.BatchNormalization*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization (besucht am 02. 09. 2020).

[19] Google Inc. *tf.keras.layers.Dropout*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout (besucht am 02. 09. 2020).

[20] Martin Israel. „Entwicklung eines UAV-basierten Systems zur Rehkitzsuche und Methoden zur Detektion und Georeferenzierung von Rehkitzen in Thermalbildern: Der Fliegende Wildretter“. Diss. Universität Osnabrück, 2016.

[21] Justin M Johnson und Taghi M Khoshgoftaar. „Survey on deep learning with class imbalance“. In: *Journal of Big Data* 6.1 (2019), S. 27. URL: https://link.springer.com/article/10.1186/s40537-019-0192-5.

[22] John F Keane und Stephen S Carr. „A brief history of early unmanned aircraft“. In: *Johns Hopkins APL Technical Digest* 32.3 (2013), S. 558–571.

[23] Kristian Kersting, Christoph Lampert und Constantin Rothkopf. *Wie Maschinen lernen*. 2019.

[24] Salman H Khan u. a. „Cost-sensitive learning of deep feature representations from imbalanced data“. In: *IEEE transactions on neural networks and learning systems* 29.8 (2017), S. 3573–3587. URL: https://ieeexplore.ieee.org/abstract/document/8012579.

[25] Zachary Kirori. „Hyper-parameter optimization: toward Convolutional Neur“. In: *Research Journal of Computer and Informa* 7.2 (2019), S. 1–5.

[26] Michał Koziarski. „Two-Stage Resampling for Convolutional Neural Network Training in the Imbalanced Colorectal Cancer Image Classification“. In: *arXiv preprint arXiv:2004.03332* (2020). URL: https://arxiv.org/abs/2004.03332.

[27] Hansang Lee, Minseok Park und Junmo Kim. „Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning“. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, S. 3713–3717. URL: https://ieeexplore.ieee.org/abstract/document/7533053.

[28] Vivian Lay Shan Lee u. a. „Semi-supervised Learning for Sentiment Classification using Small Number of Labeled Data“. In: *Procedia Computer Science* 161 (2019), S. 577–584.

[29] Seung-Hwan Lim, Steven R Young und Robert M Patton. „An analysis of image storage systems for scalable training of deep neural networks“. In: *system* 5.7 (2016), S. 11.

[30] Tsung-Yi Lin u. a. „Focal loss for dense object detection“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, S. 2980–2988.

[31] James Loy. *Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects*. Packt Publishing Ltd, 2019. URL: https://learning.oreilly.com/library/view/neural-network-projects/9781789138900/.

[32] Pariwat Ongsulee. „Artificial intelligence, machine learning and deep learning“. In: *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. IEEE. 2017, S. 1–6. URL: https://ieeexplore.ieee.org/abstract/document/8259629/.

[33] Samira Pouyanfar u. a. „Dynamic sampling in convolutional neural networks for imbalanced data classification“. In: *2018 IEEE conference on multimedia information processing and retrieval (MIPR)*. IEEE. 2018, S. 112–117.

[34] Niloy Purkait. *Hands-On Neural Networks with Keras: Design and create neural networks using deep learning and artificial intelligence principles*. Packt Publishing Ltd, 2019.

[35] Sebastian Raschka und Vahid Mirjalili. *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*. 2. Aufl. MITP-Verlags GmbH & Co. KG, 2018.

[36] Robin Philip Reiß u. a. „Angewandtes maschinelles Lernen-SS2019“. In: (2019).

[37] Eréndira Rendón u. a. „Data Sampling Methods to Deal With the Big Data Multi-Class Imbalance Problem“. In: *Applied Sciences* 10.4 (2020), S. 1276. URL: https://www.mdpi.com/2076-3417/10/4/1276.

[38] Md Shamim Reza und Jinwen Ma. „Imbalanced histopathological breast cancer image classification with convolutional neural network“. In: *2018 14th IEEE International Conference on Signal Processing (ICSP)*. IEEE. 2018, S. 619–624. URL: https://ieeexplore.ieee.org/abstract/document/8652304.

[39] Shibani Santurkar u. a. „How does batch normalization help optimization?“ In: *Advances in Neural Information Processing Systems*. 2018, S. 2483–2493. URL: http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.

[40] Dipanjan Sarkar, Raghav Bali und Tamoghna Ghosh. *Hands-On Transfer Learning with Python: Implement advanced deep learning and neural network models using TensorFlow and Keras*. Packt Publishing Ltd, 2018.

[41] Dipanjan Sarkar, Raghav Bali und Tushar Sharma. „Practical machine learning with Python“. In: *A problem-solvers guide to building real-world intelligent systems. Apress, Berkely* (2018).

[42] Cullen Schaffer. „Selecting a classification method by cross-validation“. In: *Machine Learning* 13.1 (1993), S. 135–143.

[43] Naeem Seliya, Taghi M Khoshgoftaar und Jason Van Hulse. „A study on the relationships of classifier performance metrics“. In: *2009 21st IEEE international conference on tools with artificial intelligence*. IEEE. 2009, S. 59–66. URL: https://ieeexplore.ieee.org/abstract/document/5364367/.

[44] Alexander Shustanov und Pavel Yakimov. „CNN design for real-time traffic sign recognition“. In: *Procedia engineering* 201 (2017), S. 718–725.

[45] *sklearn.model*$_s$*election.StratifiedKFold*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold (besucht am 10. 09. 2020).

[46] Michael Sorg. „Konzeption und Implementierung eines neuronalen Netzes zur Detektion von möglichen Rehkitzen anhand von Thermalbildern“. Bachelor's Thesis. Dualen Hochschule Baden-Württemberg Mannheim, 2017.

[47] Nitish Srivastava u. a. „Dropout: a simple way to prevent neural networks from overfitting“. In: *The journal of machine learning research* 15.1 (2014), S. 1929–1958.

[48] Gaël Varoquaux u. a. „Assessing and tuning brain decoders: cross-validation, caveats, and guidelines“. In: *NeuroImage* 145 (2017), S. 166–179. URL: https://www.sciencedirect.com/science/article/pii/S105381191630595X.

[49] Jacques Wainer und Gavin Cawley. „Nested cross-validation when selecting classifiers is overzealous for most practical applications“. In: *arXiv preprint arXiv:1809.09446* (2018). URL: https://arxiv.org/abs/1809.09446.

[50] Li Wan u. a. „Regularization of neural networks using dropconnect“. In: *International conference on machine learning*. 2013, S. 1058–1066.

[51] Haishuai Wang u. a. „Predicting hospital readmission via cost-sensitive deep learning“. In: *IEEE/ACM transactions on computational biology and bioinformatics* 15.6 (2018), S. 1968–1978. URL: https://ieeexplore.ieee.org/abstract/document/8338085/.

[52] Shoujin Wang u. a. „Training deep neural networks on imbalanced data sets“. In: *2016 international joint conference on neural networks (IJCNN)*. IEEE. 2016, S. 4368–4374. URL: https://ieeexplore.ieee.org/document/7727770.

[53] Shui-Hua Wang u. a. „Multiple sclerosis identification by 14-layer convolutional neural network with batch normalization, dropout, and stochastic pooling“. In: *Frontiers in neuroscience* 12 (2018), S. 818. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00818/full.

[54] Gary M Weiss. „Mining with rarity: a unifying framework“. In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), S. 7–19. URL: https://dl.acm.org/doi/abs/10.1145/1007730.1007734.

[55] Haibing Wu und Xiaodong Gu. „Towards dropout training for convolutional neural networks“. In: *Neural Networks* 71 (2015), S. 1–10. URL: https://www.sciencedirect.com/science/article/pii/S0893608015001446.

[56] Rikiya Yamashita u. a. „Convolutional neural networks: an overview and application in radiology“. In: *Insights into imaging* 9.4 (2018), S. 611–629. URL: https://link.springer.com/article/10.1007/s13244-018-0639-9.

[57] Djebbara Yasmina, Rebai Karima und O. Azouaoui. „Traffic signs recognition with deep learning“. In: 11/2018, S. 1–5. DOI: 10.1109/ICASS.2018.8652024. URL: https://www.researchgate.net/publication/331847325_Traffic_signs_recognition_with_deep_learning.

[58] Lean Yu, Shouyang Wang und Kin Keung Lai. „An integrated data preparation scheme for neural network data analysis“. In: *IEEE Transactions on Knowledge and Data Engineering* 18.2 (2005), S. 217–230. URL: https://ieeexplore.ieee.org/abstract/document/1563984.

[59] Zhifei Zhang. „Derivation of backpropagation in convolutional neural network (cnn)“. In: *University of Tennessee, Knoxville, TN* (2016).

Anhang A.

Darstellung der Differenz zwischen maximalen und minimalen Pixelwerten

Die folgenden Abbildungen zeigen die Differenz zwischen dem maximalen und dem minimalen Pixelwert innerhalb der Patches in Form eines Histogramms. Abbildung A.1 zeigt das Histogramm von allen Datensätzen. Abbildung A.2 zeigt die Differenzen getrennt nach den verschiedenen Datensätzen.

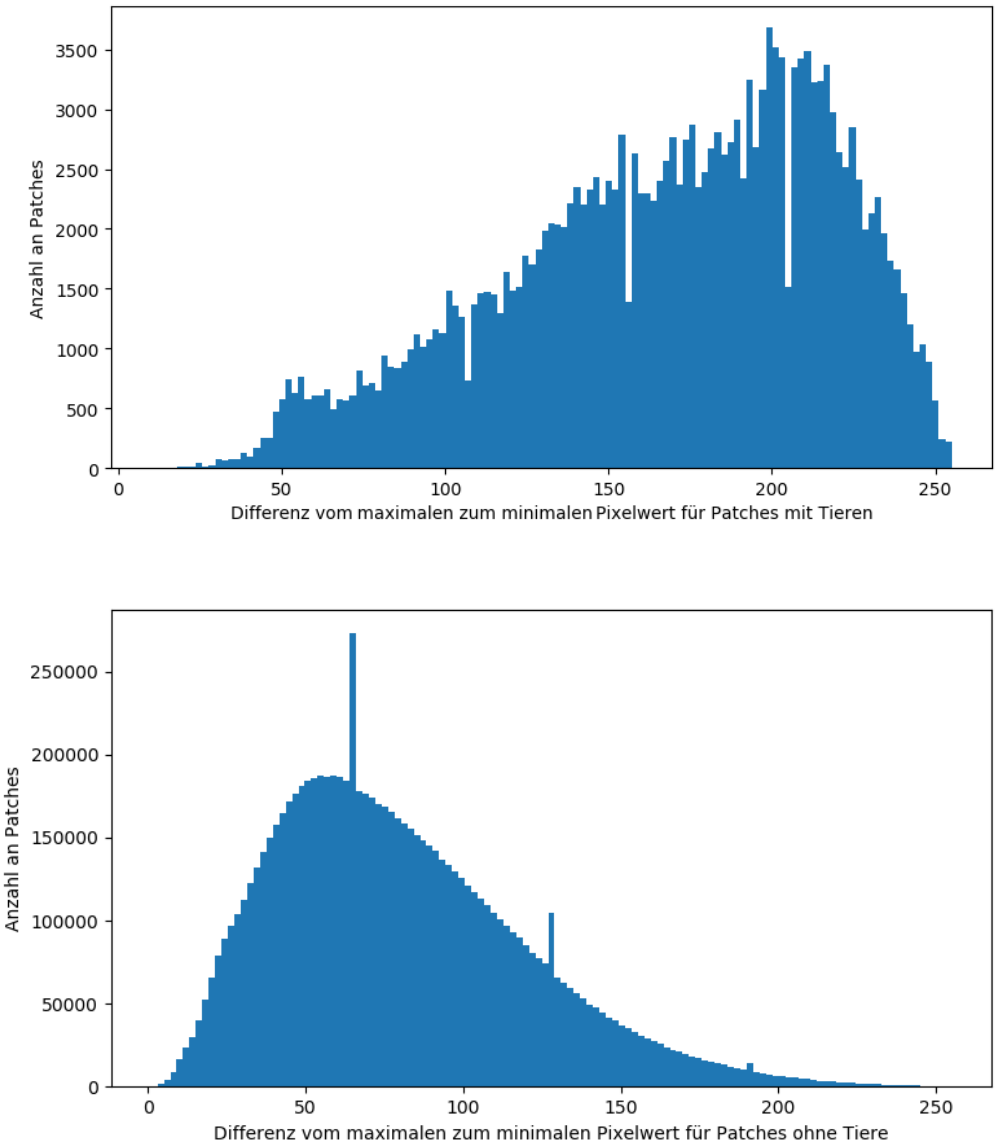


Abbildung A.1.: Differenz aller Datensätze

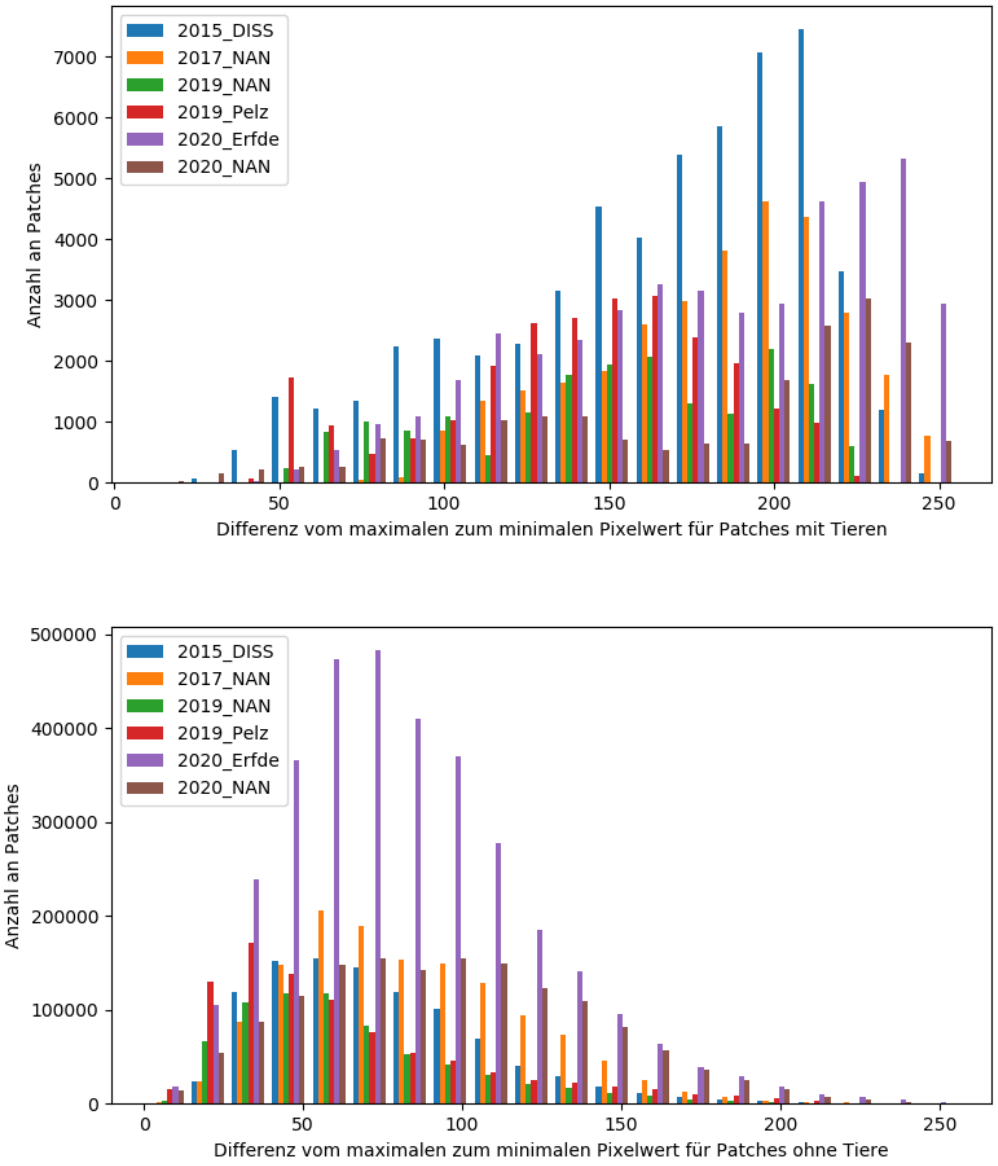


Abbildung A.2.: Differenz getrennt nach Datensätzen

Anhang B.

Netzarchitektur

Listing B.1: Keras Model

```
initializer = tf.keras.initializers.GlorotNormal()
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(16, (5,5), activation='relu', padding='valid',
    strides=(1, 1),
input_shape=input_shape,
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005)
),
tf.keras.layers.MaxPooling2D(strides=(2, 2), pool_size=(2, 2)),
tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu',
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005)),
tf.keras.layers.MaxPooling2D(strides=(2, 2), pool_size=(2, 2)),
tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu',
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005)),
tf.keras.layers.MaxPooling2D(strides=(2, 2), pool_size=(2, 2)),
# tf.keras.layers.Dropout(0.2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(8, activation='relu',
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005)),
tf.keras.layers.Dense(8, activation='relu',
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005)),
tf.keras.layers.Dense(2, activation='softmax',
kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2
    (0.0005))
])
sgd = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['categorical_accuracy',
tf.keras.metrics.Recall(name='recall', class_id=1),
tf.keras.metrics.Precision(name='precision', class_id=1)])
```

Anhang C.

Aufteilung der Datenmenge

C.1. 2015-2017 Datensatz

Flüge für die Testmenge:

- 2015-05-12_18-55_imMoos_nix
- 2015-05-17_07-43_riEismerszell_vorJagdhuette
- 2015-05-18_07-18_pur+_1Kitz_1Hase
- 2015-05-28_07-42_imMoos_1Reh
- 2015-05-28_09-37_Vogelberg_4kitze_1Hase_GeissWeggelaufen
- 2015-06-02_06-57_Laufstrecke_KaltenbergerWald_1Reh_sonst nix
- 2015-06-02_07-49_Wank_oben_1Hase
- 2015-06-02_17-27_Vox-Wiese_2Kitze_vermaeht
- 2015-06-28_13-05_Allee_Hase(8)_rest_Ameisnhfn
- 2015-06-28_15-02_Ott_Hsn_Kitz(3)
- 2015-06-28_15-44_Pur+Ecke_Sau(5)_Kitz(2,3)_Schmalreh(1)
- 2017-05-16_03-39_2Kitze
- 2017-05-16_05-53_fredl_nix
- 2017-05-17_04-36_wolf_1Kitz
- 2017-05-17_05-40_nix
- 2017-05-18_05-01_nix
- 2017-05-21_03-15_nicht_runter_geschaut_nix
- 2017-05-21_03-29_1Hase
- 2017-05-21_04-41_nix
- 2017-05-21_05-28_nix
- 2017-05-21_06-16_1Kitz
- 2017-05-22_06-30_nix
- 2017-05-25_06-07_1Kuhfladen
- 2017-05-26_03-47_nix
- 2017-05-26_05-06_1Hase_1Kitz
- 2017-05-26_05-44_nix
- 2017-05-26_06-03_1Kitz

Flüge für die Trainingsmenge:

- 2015-05-12_15-24_imMoos_grosserSuedhang_hinten
- 2015-05-12_18-22_imMoos_amAbend

- 2015-05-12_18-40_VoxWiese_1Reh
- 2015-05-12_19-09_Wal_Hauserbach_Geiss+2Kitze_2Hasen
- 2015-05-17_08-15_riEismerszell_nachJagdhuette_4Kitze
- 2015-05-17_09-13_riEismerszell_Strasse_1Reh
- 2015-05-18_06-33_pur+_2Kitze_1Geiss
- 2015-05-28_06-37_Hoefer_Suedhang_2Kitze
- 2015-05-28_08-26_Wankwiese_erdhaufen
- 2015-05-28_13-13_Hoefer_Suedhang_1Kitz_1Geiss_Shutterstoerung
- 2015-05-28_13-59_Walleshausen_Sauerampfer_1KitzNurMitHundGefunden_Shutter
- 2015-05-31_08-04_Modellflugplatz_Hase_Erdloch_Geissvoll
- 2015-06-01_15-15_Vogelberg_3Kitze,davon_1 vermaeht
- 2015-06-01_16-57_Wal_Moos_Fried_Kitz(2,3)_Reh(1)
- 2015-06-01_18-35_Eismerszell_Wank_Wdh
- 2015-06-01_19-05_Pestn_Ott_2kitze
- 2015-06-01_20-33_Pestn_unten_1Reh_1Kitz
- 2015-06-02_07-27_Wank_oben_Kitz (2,3,4)
- 2015-06-02_08-39_Petz_Dachslochhang_2Kitze
- 2015-06-02_09-08_Huberdreieck_Kitz(2)_Entengelege(5)_Hase(6)
- 2015-06-02_10-51_Wal_Klotz_Fahrradfahrer_nix
- 2015-06-02_15-35_Hsn_Suedhang_west_Reh(1)_Kitz(2)_Lager(3)
- 2015-06-28_07-38_Eching_Hase(8,9)
- 2015-06-28_07-50_Eching_restGrWiese_Pfosten(1)_Erde(2,5,7)_Kitz(3) _Geiss(4)_nebenPferden_Hase(8,5)_Reh(7)_Sasse(6)
- 2015-06-28_11-59_Alleeende_27deg_nix
- 2015-06-28_12-28_Moos_Kitze(2,4)
- 2015-06-28_16-13_Ott_Wal_Hase(3)_Kitze(4,5)
- 2015-05-17_08-57_riEismerszell_vorne_1Hase_Shutterstoerung
- 2017-05-15_10-28_
- 2017-05-15_14-21_klaer_1Kitz
- 2017-05-16_04-43_liedel_4Kitze
- 2017-05-16_07-11_au_1Kitz
- 2017-05-17_03-43_lidl_nix
- 2017-05-17_05-03_wolfg_wiederholung_1Kitz
- 2017-05-17_05-25_nix
- 2017-05-18_05-18_nix
- 2017-05-21_04-08_wiederholung_1Hase
- 2017-05-21_04-27_Wiederholung_nix
- 2017-05-21_04-57_nix
- 2017-05-21_06-38_4Kitze
- 2017-05-21_07-21_komischeFlecken_2Kitze

- 2017-05-21_07-39_wiederholung_2Kitze
- 2017-05-21_09-14_1Kitz_fragen!
- 2017-05-22_06-14_klaeranlage_1Hase_sonst_nix
- 2017-05-23_05-28_nahbeieinander_2Kitze
- 2017-05-23_06-08_nix
- 2017-05-23_06-31_Zaun_1Kitz
- 2017-05-23_07-02_1Hase
- 2017-05-25_04-53_2Kitze_fragen!
- 2017-05-25_05-12_1Reh_2Kitze
- 2017-05-25_05-44_2aeltereKitze
- 2017-05-25_06-36_1Geiss_1Kitz
- 2017-05-25_07-11_2Kitze
- 2017-05-25_07-25_wiederholung_2Kitze
- 2017-05-25_08-09_2Kuhhaufen_sonst_nix
- 2017-05-26_04-05_nix
- 2017-05-26_04-26_nix
- 2017-05-26_04-44_nix
- 2017-05-26_06-44_2Kitze
- 2017-05-27_04-33_nix
- 2017-05-27_04-58_3Kitze
- 2017-06-10_04-13_3Kitze
- 2017-06-10_04-44_1Kitz
- 2017-06-10_05-30_2Kitze

C.2. gesamter Datensatz

Trainingsmenge 1:

- 2015-05-17_09-13_riEismerszell_Strasse_1Reh
- 2015-05-18_06-33_pur+_2Kitze_1Geiss
- 2015-05-28_06-37_Hoefer_Suedhang_2Kitze
- 2015-05-28_13-13_Hoefer_Suedhang_1Kitz_1Geiss_Shutterstoerung
- 2015-06-02_10-51_Wal_Klotz_Fahrradfahrer_nix
- 2015-06-28_07-50_Eching_restGrWiese_Pfosten(1)_Erde(2,5,7)_Kitz(3)_Geiss(4)_nebenPferden_Hase(8,5)_Reh(7)_Sasse(6)
- 2015-06-28_11-59_Alleeende_27deg_nix
- 2015-06-28_15-44_Pur+Ecke_Sau(5)_Kitz(2,3)_Schmalreh(1)
- 2015-06-28_16-13_Ott_Wal_Hase(3)_Kitze(4,5)
- 2015-06-01_16-57_Wal_Moos_Fried_Kitz(2,3)_Reh(1)

- 2015-06-28_15-02_Ott_Hsn_Kitz(3)
- 2017-05-18_05-18_nix
- 2017-05-21_06-38_4Kitze
- 2017-05-21_07-39_wiederholung_2Kitze
- 2017-05-23_05-28_nahbeieinander_2Kitze
- 2017-05-23_07-02_1Hase
- 2017-05-25_05-12_1Reh_2Kitze
- 2017-05-25_08-09_2Kuhhaufen_sonst_nix
- 2017-05-26_04-26_nix
- 2017-05-26_06-03_1Kitz
- 2017-06-10_04-13_3Kitze
- 2017-06-10_05-30_2Kitze
- 20190509_085534
- 20190510_113918
- 20190516_115738
- 20190516_121605
- 20190516_130747
- 20190524_050104
- 20190524_122744
- 20190607_052801
- 20190612_050320
- 20190612_052554
- 20190613_051048
- 20191123_101217
- 20190331_110113
- 20190414_084403
- 20190421_120308
- 20190502_104709
- 20190507_094642
- 20190516_124144
- 20190521_090352
- 20190524_053552
- 20190524_060818
- 20190531_040746
- 20190531_041619
- 20190531_050853
- 20190601_050856
- 20190602_051317
- 20190603_044000
- 20190604_075513

- 20190604_082014
- 20190608_044128
- 200519_0638_Bargen02
- 200519_0709_Bargen04
- 200519_1411_Bargen23
- 200520_1044_DT02
- 200520_1105_DT03
- 200521_0856_DH22
- 200525_0429_SG09
- 200525_0502_SG11
- 200525_0817_SG4+5
- 200525_0847_SG6
- 200525_0909_SG02
- 200525_1227_MS03
- 200526_0445_JS10
- 200526_0608_JS06
- 200526_0656_KEK09
- 200526_0937_KEK5-6
- 200527_0538_hb5
- 200527_0743_Frensen8
- 200527_1201_Frensen8
- 200528_0551_IH06
- 200528_0604_IH04
- 200528_0917_AW01b
- 200529_0451_KV04-06
- 200529_0518_KV08
- 200529_1238_AW04b
- 200531_0443_AB08
- 200531_0537_Frensen_01
- 200531_0558_Frensen_12
- 200601_0406_Müller01
- 200602_0411_Menzer05
- 200602_0434_Menzer07
- 200602_0819_Menzer09
- 200610_0420_B11+12
- 200610_0547_B06+08
- 200610_0749_B01+02
- 200614_0728_AR01+02
- 200614_0836_AR03
- 200614_1512_tw02

- 200616_0609_OS01
- 200621_0438_zufall02
- 200621_0805_Borstel03
- 200621_0849_Borstel04
- 200622_0605_Heyn04
- 19800110_122434
- 200514_134624
- 200518_102832
- 200520_074527
- 200520_110719
- 200612_080954
- 200623_064053
- 200623_070652
- 20200416_134601
- 20200424_070445

Trainingsmenge 2:

- 2015-05-12_15-24_imMoos_grosserSuedhang_hinten
- 2015-05-12_18-22_imMoos_amAbend
- 2015-05-12_19-09_Wal_Hauserbach_Geiss+2Kitze_2Hasen
- 2015-05-17_08-15_riEismerszell_nachJagdhuette_4Kitze
- 2015-05-28_08-26_Wankwiese_erdhaufen
- 2015-05-28_13-59_Walleshausen_Sauerampfer_1KitzNurMitHundGefunden_Shutter
- 2015-06-01_15-15_Vogelberg_3Kitze,davon_1 vermaeht
- 2015-06-01_19-05_Pestn_Ott_2kitze
- 2015-06-02_07-49_Wank_oben_1Hase
- 2015-06-28_07-38_Eching_Hase(8,9)
- 2017-05-17_03-43_lidl_nix
- 2017-05-17_05-03_wolfg_wiederholung_1Kitz'
- 2017-05-17_05-40_nix
- 2017-05-22_06-30_nix
- 2017-05-25_04-53_2Kitze_fragen!
- 2017-05-25_05-44_2aeltereKitze
- 2017-05-25_06-07_1Kuhfladen
- 2017-05-25_06-36_1Geiss_1Kitz
- 2017-05-26_04-05_nix
- 2017-05-26_05-06_1Hase_1Kitz
- 2017-05-26_05-44_nix
- 2017-05-16_04-43_liedel_4Kitze
- 20190509_090000

- 20190509_100455
- 20190516_125401
- 20190516_132523
- 20190516_143942
- 20190517_052949
- 20190524_070000
- 20190524_113320
- 20190607_041301
- 20190607_043859
- 20190607_050507
- 20190607_060545
- 20190612_050909
- 20190613_054731
- 20190516_120000
- 20190405_084007
- 20190405_084007_Klaerweiher
- 20190421_110000
- 20190507_075216
- 20190507_080902
- 20190524_041717
- 20190531_094536
- 20190531_101825
- 20190531_104359
- 20190601_041700
- 20190601_061501
- 20190601_063826
- 20190603_032940
- 20190604_080001
- 20190607_051858
- 20190613_042435
- 20190613_044916
- 20190414_070942
- 20190525_050000
- 200520_0813_AH04
- 200520_0921_DT01
- 200521_0645_DH12-2
- 200524_1547_DH6b
- 200525_0655_MS09-10
- 200525_0720_MS07
- 200525_0800_SG03

- 200526_0508_KEK10
- 200526_0527_KEK11
- 200526_0943_KEK7-8
- 200527_0929_Kühl1
- 200527_1035_Frensen6
- 200528_0431_AB01
- 200528_0645_IH03
- 200528_0747_IH7-8
- 200528_0903_AW02
- 200529_0550_KV09
- 200529_1110_AW09
- 200531_0417_AB04
- 200531_0508_AB11
- 200531_0636_Frensen05-07
- 200601_0425_Müller02-04
- 200601_0443_Müller11
- 200601_0505_Müller05-07
- 200602_0524_Menzer02+03
- 200602_0649_Menzer01
- 200610_0629_B05
- 200610_0716_B13-15
- 200614_1428_tw01
- 200615_0419_KS06
- 200615_0553_KS02
- 200620_0448_GH04-06
- 200620_0541_GH03
- 200620_0717_chris03
- 200620_0817_chris02
- 200621_0413_zufall01
- 200621_0610_Lütje06-07
- 200621_0742_Borstel02
- 200621_0903_Borstel05
- 190214_111334
- 190214_111418
- 200514_153443
- 200518_052145
- 200518_062708
- 200518_083441
- 200519_072529
- 200522_065819

- 200527_075351
- 200530_075208
- 200623_073143

Trainingsmenge 3:

- 2015-05-12_18-40_VoxWiese_1Reh
- 2015-05-17_07-43_riEismerszell_vorJagdhuette
- 2015-05-28_07-42_imMoos_1Reh
- 2015-05-28_09-37_Vogelberg_4kitze_1Hase_GeissWeggelaufen
- 2015-06-02_07-27_Wank_oben_Kitz (2,3,4)
- 2015-06-28_13-05_Allee_Hase(8)_rest_Ameisnhfn
- 2015-05-31_08-04_Modellflugplatz_Hase_Erdloch_Geissvoll
- 2015-06-02_06-57_Laufstrecke_KaltenbergerWald_1Reh_sonst nix
- 2017-05-15_10-28_
- 2017-05-15_14-21_klaer_1Kitz
- 2017-05-18_05-01_nix
- 2017-05-21_03-29_1Hase
- 2017-05-21_04-08_wiederholung_1Hase
- 2017-05-21_04-57_nix
- 2017-05-21_06-16_1Kitz
- 2017-05-21_09-14_1Kitz_fragen!
- 2017-05-22_06-14_klaeranlage_1Hase_sonst_nix
- 2017-05-23_06-31_Zaun_1Kitz
- 2017-05-26_03-47_nix
- 2017-05-27_04-33_nix
- 2017-05-27_04-58_3Kitze
- 2017-05-16_07-11_au_1Kitz
- 2017-05-21_03-15_nicht_runter_geschaut_nix
- 20190325_130259
- 20190502_092222
- 20190502_095516
- 20190502_100000
- 20190502_120001
- 20190509_142434
- 20190510_114728
- 20190510_122412
- 20190516_140305l
- 20190517_050455
- 20190524_043953
- 20190524_061238

- 20190524_110718
- 20190531_044714
- 20190617_050756
- 20191123_092228
- 20190517_042907
- 20190524_065550
- 20190414_065026
- 20190414_073454
- 20190507_084951
- 20190525_045638
- 20190531_033324
- 20190531_034820
- 20190601_045135
- 20190603_034435
- 20190608_040846
- 20190613_035058
- 20190602_055656
- 20190607_050001
- 200518_0925_Bargen_Langeckern
- 200519_0509_Erfde03
- 200519_0754_Baltz01
- 200519_0927_Bargen06
- 200519_1058_Bargen16_17
- 200519_1203_Bargen19
- 200520_0705_AH02
- 200521_0520_DH12
- 200525_0442_SG10
- 200525_0619_MS05
- 200525_0637_MS08
- 200525_0940_SG01
- 200525_1256_Sg07
- 200526_0424_JS11-12
- 200526_0545_JS07
- 200526_0751_JS02
- 200526_0816_JS4-5
- 200527_0507_hb3-4
- 200527_0556_hb1-2
- 200527_0625_Frensen1-3
- 200527_0858_Kühl3
- 200528_0517_AB03

- 200528_0627_IH05
- 200529_0431_KV03
- 200529_0526_KV07
- 200529_0606_KV03b
- 200529_0737_AW08
- 200529_0802_AW04
- 200530_0708_DK01
- 200531_0643_Frensen04
- 200531_0749_Frensen02
- 200601_0531_Müller08-10
- 200601_0612_Müller13
- 200610_0605_B07++
- 200610_0643_B03+04
- 200614_0928_AR01+02
- 200615_0455_KS05
- 200620_0646_chris02
- 200620_0923_chris04
- 200620_1037_chris05
- 200621_0642_Lütje04
- 190214_111317
- 190214_111404
- 190214_111424
- 190214_111444
- 190214_111513
- 200518_060552
- 200518_085753
- 200520_064603
- 200529_065407
- 200529_073418
- 200623_082356
- 20200423_082320

Testdatenmenge:

- 2015-05-12_18-55_imMoos_nix
- 2015-05-17_08-57_riEismerszell_vorne_1Hase_Shutterstoerung
- 2015-05-18_07-18_pur+_1Kitz_1Hase
- 2015-06-01_18-35_Eismerszell_Wank_Wdh
- 2015-06-01_20-33_Pestn_unten_1Reh_1Kitz
- 2015-06-02_08-39_Petz_Dachslochhang_2Kitze
- 2015-06-02_09-08_Huberdreieck_Kitz(2)_Entengelege(5)_Hase(6)

- 2015-06-02_15-35_Hsn_Suedhang_west_Reh(1)_Kitz(2)_Lager(3)
- 2015-06-02_17-27_Vox-Wiese_2Kitze_vermaeht
- 2015-06-28_12-28_Moos_Kitze(2,4)
- 2017-05-16_05-53_fredl_nix
- 2017-05-17_04-36_wolf_1Kitz
- 2017-05-17_05-25_nix
- 2017-05-21_04-27_Wiederholung_nix
- 2017-05-21_04-41_nix
- 2017-05-21_05-28_nix
- 2017-05-21_07-21_komischeFlecken_2Kitze
- 2017-05-23_06-08_nix
- 2017-05-25_07-11_2Kitze
- 2017-05-25_07-25_wiederholung_2Kitze
- 2017-05-26_04-44_nix
- 2017-06-10_04-44_1Kitz
- 2017-05-16_03-39_2Kitze
- 2017-05-26_06-44_2Kitze
- 20190516_140305
- 20190516_143306
- 20190517_034829
- 20190524_052511
- 20190531_052841
- 20191112_103743
- 20191123_082916
- 20191123_110531
- 20190331_100223
- 20190421_112452
- 20190523_071828
- 20190525_043253
- 20190525_052837
- 20190525_061358
- 20190531_043903
- 20190531_100338
- 20190531_110911
- 20190601_032651
- 20190601_034717
- 20190601_042710
- 20190601_054753
- 20190602_043506
- 20190602_045118

- 20190603_040759
- 20190607_045449
- 20190607_054327
- 20190608_042301
- 20190613_032808
- 200519_1003_Bargen10_11
- 200520_0632_AH01
- 200520_0736_AH03
- 200521_0429_DH14
- 200521_0617_DH10-11
- 200521_0732_DH16-18b
- 200521_0812_DH15
- 200521_0923_DH20-23
- 200521_0953_DH6a
- 200525_0554_MS06
- 200525_1039_Schacht
- 200525_1150_MS01
- 200525_1206_MS02
- 200526_0717_JS01
- 200526_1137_KEK3
- 200527_0650_Frensen4-5
- 200527_0806_Kühl9
- 200527_0910_Kühl2
- 200528_0445_AB02
- 200528_0823_AW01
- 200528_0838_AW03
- 200529_0416_KV01-02
- 200529_0645_AW07
- 200531_0704_Frensen10-11
- 200531_0728_Frensen03
- 200601_0558_Müller14-15
- 200601_0658_Müller12
- 200602_0519_Menzer07a
- 200602_0558_Menzer04
- 200602_0743_Menzer08
- 200610_0444_B09+10
- 200614_1248_gh01+02
- 200615_0406_KS03
- 200620_0621_chris01
- 200621_0532_Lütje05

- 190214_111257
- 190214_111320
- 190214_111440
- 190214_111459
- 190214_111611_2kitz
- 200514_145340
- 200518_055057
- 200518_071150
- 200518_073619
- 200518_081203
- 200518_091911
- 200519_080257
- 200519_082553
- 200520_072236
- 200527_071640
- 200529_054712
- 200623_075759

Anhang D.

Konfusionsmatrizen von jedem Flug im Testdatensatz

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	9	12951

(D.1.1) `2015-05-12_18-55_imMoos_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	4729	0
	kein-Tier	266	26421

(D.1.3) `2015-05-18_07-18_pur+_1Kitz_1Hase`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	5462	1109
	kein-Tier	237	27287

(D.1.5) `2015-06-01_20-33_Pestn_unten_1Reh_1Kitz`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1402	19
	kein-Tier	243	36901

(D.1.7) `2015-06-02_09-08_Huberdreieck_Kitz(2)_Entengelege(5)_Hase(6)`

	vorhergesagt	
	Tier	kein-Tier
Tier	1773	278
kein-Tier	1	21118

(D.1.2) `2015-05-17_08-57_riEismerszell_vorne_1Hase_Shutterstoerung`

	vorhergesagt	
	Tier	kein-Tier
Tier	735	0
kein-Tier	45	17528

(D.1.4) `2015-06-01_18-35_Eismerszell_Wank_Wdh`

	vorhergesagt	
	Tier	kein-Tier
Tier	47	0
kein-Tier	78	29756

(D.1.6) `2015-06-02_08-39_Petz_Dachslochhang_2Kitze`

	vorhergesagt	
	Tier	kein-Tier
Tier	896	0
kein-Tier	318	38597

(D.1.8) `2015-06-02_15-35_Hsn_Suedhang_west_Reh(1)_Kitz(2)_Lager(3)`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	568	384
	kein-Tier	46	21275

(D.1.9) `2015-06-02_17-27_Vox-Wiese_2Kitze_vermaeht`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	468	1
	kein-Tier	105	12841

(D.1.10) `2015-06-28_12-28_Moos_Kitze(2,4)`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	48	14992

(D.1.11) `2017-05-16_05-53_fredl_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	182	0
	kein-Tier	41	27628

(D.1.12) `2017-05-17_04-36_wolf_1Kitz`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	91	13349

(D.1.13) `2017-05-17_05-25_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	91	13669

(D.1.14) `2017-05-21_04-27_Wiederholung_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	117	18043

(D.1.15) `2017-05-21_04-41_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	1	39759

(D.1.16) `2017-05-21_05-28_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	784	1
	kein-Tier	248	19312

(D.1.17) `2017-05-21_07-21_komischeFlecken_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	93	31427

(D.1.18) `2017-05-23_06-08_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1547	0
	kein-Tier	30	38790

(D.1.19) `2017-05-25_07-11_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1540	0
	kein-Tier	23	28638

(D.1.20) `2017-05-25_07-25_wiederholung_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	125	25955

(D.1.21) `2017-05-26_04-44_nix`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	714	0
	kein-Tier	33	32818

(D.1.22) `2017-06-10_04-44_1Kitz`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1109	4
	kein-Tier	29	13681

(D.1.23) `2017-05-16_03-39_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	660	0
	kein-Tier	208	22801

(D.1.24) `2017-05-26_06-44_2Kitze`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	337	9743

(D.1.25) `20190516_140305`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	4	2956

(D.1.26) `20190516_143306`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	20	21340

(D.1.27) `20190517_034829`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	925	2
	kein-Tier	117	20110

(D.1.28) `20190524_052511`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	137	31
	kein-Tier	23	12772

(D.1.29) `20190531_052841`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	331	12
	kein-Tier	80	23428

(D.1.30) `20191112_103743`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	302	13
	kein-Tier	71	6640

(D.1.31) `20191123_082916`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	248	25
	kein-Tier	38	6430

(D.1.32) `20191123_110531`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	139	16981

(D.1.33) `20190331_100223`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	630	0
	kein-Tier	1077	15454

(D.1.34) `20190421_112452`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1009	279
	kein-Tier	14	10420

(D.1.35) `20190523_071828`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	791	0
	kein-Tier	13	12600

(D.1.36) `20190525_043253`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	228	3
	kein-Tier	6	6785

(D.1.37) `20190525_052837`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	301	0
	kein-Tier	196	14832

(D.1.38) `20190525_061358`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	497	62
	kein-Tier	14	11644

(D.1.39) `20190531_043903`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	110	15090

(D.1.40) `20190531_100338`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	94	7426

(D.1.41) `20190531_110911`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	322	0
	kein-Tier	69	8398

(D.1.42) `20190601_032651`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	19	12461

(D.1.43) `20190601_034717`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1027	9
	kein-Tier	3	8686

(D.1.44) `20190601_042710`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	315	0
	kein-Tier	126	12102

(D.1.45) `20190601_054753`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	420	0
	kein-Tier	75	18549

(D.1.46) `20190602_043506`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	186	0
	kein-Tier	93	14137

(D.1.47) `20190602_045118`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	256	3
	kein-Tier	75	23513

(D.1.48) `20190603_040759`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	113	12367

(D.1.49) `20190607_045449`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	949	122
	kein-Tier	91	18027

(D.1.50) `20190607_054327`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	9	18151

(D.1.51) `20190608_042301`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	38	17882

(D.1.52) `20190613_032808`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	91	19989

(D.1.53) `200519_1003_Bargen10_11`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	105	0
	kein-Tier	58	17376

(D.1.54) `200520_0632_AH01`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	227	18813

(D.1.55) `200520_0736_AH03`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	496	3
	kein-Tier	38	20017

(D.1.56) `200521_0429_DH14`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	50	25790

(D.1.57) `200521_0617_DH10-11`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	170	29910

(D.1.58) `200521_0732_DH16-18b`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	128	18272

(D.1.59) `200521_0812_DH15`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	147	0
	kein-Tier	554	37038

(D.1.60) `200521_0923_DH20-23`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	531	26669

(D.1.61) `200521_0953_DH6a`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	20	27260

(D.1.62) `200525_0554_MS06`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	152	20728

(D.1.63) `200525_1039_Schacht`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	336	0
	kein-Tier	177	18767

(D.1.64) `200525_1150_MS01`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	1507	26573

(D.1.65) `200525_1206_MS02`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	56	18344

(D.1.67) `200526_1137_KEK3`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	18	13022

(D.1.69) `200527_0806_Kühl9`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	207	18
	kein-Tier	90	16376

(D.1.71) `200528_0445_AB02`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	613	73
	kein-Tier	260	12107

(D.1.73) `200528_0838_AW03`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	147	0
	kein-Tier	10	7026

(D.1.75) `200529_0645_AW07`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	500	2
	kein-Tier	78	26139

(D.1.77) `200531_0728_Frensen03`

	vorhergesagt	
	Tier	kein-Tier
Tier	392	0
kein-Tier	23	14764

(D.1.66) `200526_0717_JS01`

	vorhergesagt	
	Tier	kein-Tier
Tier	0	0
kein-Tier	167	18473

(D.1.68) `200527_0650_Frensen4-5`

	vorhergesagt	
	Tier	kein-Tier
Tier	0	0
kein-Tier	142	13618

(D.1.70) `200527_0910_Kühl2`

	vorhergesagt	
	Tier	kein-Tier
Tier	392	0
kein-Tier	472	26395

(D.1.72) `200528_0823_AW01`

	vorhergesagt	
	Tier	kein-Tier
Tier	0	0
kein-Tier	50	16910

(D.1.74) `200529_0416_KV01-02`

	vorhergesagt	
	Tier	kein-Tier
Tier	119	0
kein-Tier	22	20691

(D.1.76) `200531_0704_Frensen10-11`

	vorhergesagt	
	Tier	kein-Tier
Tier	21	112
kein-Tier	66	29208

(D.1.78) `200601_0558_Müller14-15`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	81	164
	kein-Tier	99	25895

(D.1.79) `200601_0658_Müller12`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	33	9167

(D.1.80) `200602_0519_Menzer07a`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	172	0
	kein-Tier	167	15985

(D.1.81) `200602_0558_Menzer04`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	155	12325

(D.1.82) `200602_0743_Menzer08`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	100	19980

(D.1.83) `200610_0444_B09+10`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	257	9
	kein-Tier	562	21745

(D.1.84) `200614_1248_gh01+02`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	209	29
	kein-Tier	9	10058

(D.1.85) `200615_0406_KS03`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	119	27321

(D.1.86) `200620_0621_chris01`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	33	25327

(D.1.87) `200621_0532_Lütje05`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	915	39165

(D.1.88) `190214_111257`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	494
	kein-Tier	133	8565

(D.1.89) `190214_111320`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	406
	kein-Tier	864	21678

(D.1.90) `190214_111440`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	478	624
	kein-Tier	555	45867

(D.1.91) `190214_111459`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	634	349
	kein-Tier	718	49149

(D.1.92) `190214_111611_2kitz`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	2087	13
	kein-Tier	300	26534

(D.1.93) `200514_145340`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	98	0
	kein-Tier	157	13200

(D.1.94) `200518_055057`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	364	0
	kein-Tier	247	43820

(D.1.95) `200518_071150`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	265	36
	kein-Tier	896	46444

(D.1.96) `200518_073619`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	174	26226

(D.1.97) `200518_081203`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	187	9
	kein-Tier	237	34476

(D.1.98) `200518_091911`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	1246	0
	kein-Tier	402	31864

(D.1.99) `200519_080257`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	297	24343

(D.1.100) `200519_082553`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	141	34979

(D.1.101) `200520_072236`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	191	33569

(D.1.102) `200527_071640`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	0	0
	kein-Tier	166	34794

(D.1.103) `200529_054712`

		vorhergesagt	
		Tier	kein-Tier
tatsächlich	Tier	623	0
	kein-Tier	308	35751

(D.1.104) `200623_075759`