Master Thesis

im Rahmen des

Universitätslehrganges „Geographical Information Science & Systems“

(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)

der Paris Lodron-Universität Salzburg

zum Thema

„Classification of multi-spectral and multi-temporal satellite images using deep learning LSTM networks“

- Agricultural crop prediction of Sentinel 2 images -

-

vorgelegt von

Sascha Woditsch

104866, UNIGIS MSc Jahrgang 2017

Betreuer/in:

Dr. Dirk Tiede

Zur Erlangung des Grades

„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Münster, 04.10.2019

Acknowledgements

First of all, I want to thank the “Thüringer Landesverwaltungsamt” (engl: Ministry of Administration), which provided the anonymised agricultural vector data without which this thesis would not have been possible.
For the interesting and insightful GIS modules and the general support during my studies, I want to thank the UNIGIS Team of the University of Salzburg. For the thematic support and valuable hints in regards to my thesis, I want to thank Dr. Dirk Tiede and my colleges at EFTAS. I also want to thank EFTAS for the financial support for my UNIGIS MSc studies.

Furthermore, I’m very grateful for the proofreading done by Gregor Kleine-Limberg and Cathy Bialy. Lastly, I’m very thankful for the support of my wonderful wife Josefa. Thank you for your patience during the last year, especially during all those weekends occupied by me sitting in front of a computer trying to understand GIS, Python and neural networks.

Eidesstattliche Erklärung (Deutsch)

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Münster, the 04.10.2019

Sascha Woditsch

Declaration of Authorship (English)

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I have not used other than the declared sources and I have explicitly marked all material which has been quoted either literally or by content from the used sources. This thesis was not previously presented to another examination board and has not been published.

Münster, the 04.10.2019

Sascha Woditsch

Table of Contents

Table of Figures

Table of Tables

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
ESA	European Space Agency
FE	Feature Engineering
FN	False negatives
FP	False positives
CAP	Common Agricultural Policy
GPU	Graphics Processing Unit
LSTM	Long Short-Term Memory (Neural Network)
NN	Neural Network
RNN	Recurrent Neural Network
SITS	Satellite image time series
TN	True negatives
TP	True positives
UTM	Universal Transverse Mercator

Notation

This thesis follows the general notation of DL and NNs as is the common standard (c.f. Goodfellow et al., 2016, Chapter Notation, p. xiii-xvi). The most important notations are:

x	Input vector which contain the information to predict y. The x vector can contain multiple feature variables for each sample of the data set. In this thesis input x are the different Sentinel 2 bands.
y	Output vector, which is explained by the input vector x in the form of y = f(x). In this thesis input y or y data are the actual crop types.
ŷ	Prediction vector is predicted by a NN model on the input data x, as an approximation of the function ŷ = y = f(x).
w	The letter w stands for weights and determines how each feature x_i influences the prediction. Thus w_i is used as coefficient which is multiplied by feature x_i to find the best ŷ for the given input x. If a weight for a certain feature is greater in magnitude, it has a stronger influence on the prediction. On the contrary, a feature's weight close to zero has only little effect on the prediction.

1. Introduction

In recent years, the availability of free satellite images has increased largely. Kussul et al. call these years the *“[...] years of Big Free Data in remote sensing.” (Kussul et al., 2017, p. 1)*. This huge amount of free data provides the possibility to analyse a growing number of phenomenons on earth’s surface via remote sensing, especially because the available data has two important characteristics (Pelletier et al., 2019):

1. Multi-spectral: The images contain multiple bands of different electromagnetic wavelengths. This allows the detection of different attributes, which are not evident in the visible spectrum, e.g. the condition of vegetation can be assessed in the near-infrared wavelength.
2. Multi-temporal: Images of the same area are recorded more frequently. This sequential information of the changes within a time series of images offers an additional information layer. This improves the classification quality of any temporal phenomena because the observation is not based on a single image, but additionally one a temporal progression of the pixel values.

However, these two characteristics have a price which is complexity. Complexity arises from the multidimensional nature that results in an enormous size, e.g. the Sentinel 2 satellites produce 1.6 terabytes of data daily (Drusch et al., 2012). This makes it very difficult to extract the relevant and needed information from the images. Thus, an ongoing field of research is the development of intelligent analysis methods of aerial images incorporating the spectral and especially the temporal dimension. In 2012 Petitjean wrote: *„In order to efficiently handle the huge amount of data that will be produced by these new sensors, adapted methods for SITS analysis have to be developed.“(Petitjean 2012, p.1805).* Five years later Ienco et al. writes: *„How efficiently manage and analyze remote sensing time series is still an open challenge in the remote sensing field.“(Ienco et al., 2017, p. 1685).* And still in 2019 Pelletier et al. writes: *“The state-of-the-art classification algorithms used to produce maps are currently Support Vector Machines (SVMs) and Random Forests (Rfs). [...] These algorithms are oblivious to the temporal dimension that structures SITS.”* (Pelletier et al., 2019, p.2). This shows the still ongoing research question of how to effectively include the temporal dimension of SITS.

Multiple studies indicate DL (cf. chapter 1.3) as one of the most promising methods to analyse and classify images. One of the main advantages of DL is that it doesn’t require F. *“Previously, tradi-*

tional approaches for image classification tasks had been based on hand-engineered features, whose performance affected heavily the overall results. FE is a complex, time-consuming process which needs to be altered whenever the problem or the dataset changes.” (Kamilaris and Prenafeta-Boldú, 2018, p. 72). Instead of necessary interactions by experts to detect the feature patterns stretching across the spatial, spectral and temporal dimension, DL is able to learn and then detect these patterns. Different DL models have been developed for different tasks. One of the more recent developments in DL are so-called LSTM models (Kamilaris and Prenafeta-Boldú, 2018) that focus on the analysis of sequential data, such as temporal data.

This thesis contributes to the open question of intelligent image analyses by developing a LSTM network capable of classifying SITS. As a prominent example of aerial image analysis, the goal of this thesis is crop classification. Since different crop sub-classes are often hard to discern as they share very similar features in contrast to larger meta-classes, e.g. water, forest, urban area, the classification of crop can benefit strongly from the temporal dimension. Time provides an additional layer of information and thus the possibility to differentiate crops due to different temporal progress values. The SITS used in this theses as input data for the LSTM are Sentinel 2 images. Sentinel 2 data has three major advantages, first being its free availability. The second benefit is the multi-spectral diversity, with 13 different spectral channels, several of them in the near-infrared spectrum which are well suited to detect differences between plants. The third advantage is the temporal frequency, with different S2 images available every 1-4 days (over Europe). These images, therefore, have a huge potential for temporal analysis. Based on these elements the general research question of this thesis is: Is it possible to classify agricultural crops with a LSTM by using multi-temporal and multi-dimensional Sentinel 2 images?

1.1. Structure of the thesis

The following sections present a framework of this thesis to understand its origins, concepts and goals. Therefore these chapters take a detailed look at the research question, the current scientific research regarding DL in aerial image analysis and the general concept of the developed approach. Depending on the level of knowledge the reader might want to read chapter 4.1 and 4.2.1 first to gain an understanding of the basic concepts of NN.

The following chapters of this thesis, i.e. chapter 2 to 8, are split into two general parts. Chapter 2 to 4 constitute part one which describes the theoretical background. Chapter 2 describes the properties of the vector and raster input data used in this thesis. Chapter 3 presents the used tools, i.e. the used software and hardware. Chapter 4 explains the history and theoretical background of DL and NNs in general and focuses on LSTM networks.

Chapter 5 to 8 constitute part two which describes the development and application of the NN. The structure of part two follows the chronological data flow. Chapter 5 gives a short overview of the development and data flow. Then chapter 6 describes the data preprocessing. The three sections 6.1, 6.2 and 6.3 focus on the preparation of the three data types used, namely: vector, raster and numerical data. Chapter 7 demonstrates the architecture of the actual LSTM network. It shows how the network was trained and gives a first overview of the training accuracies. Chapter 8 evaluates the classification results from a general to a more specific perspective. In chapter 8.1 the general scoring parameters and in chapter 8.2 the individual class scoring metrics are presented. Then chapter 8.3 analyses the reasons for miss classifications. The discussion in chapter 9 summarizes the classification and its evaluation results and connects them to the research questions posed in chapter 1.2. Lastly, chapter 10 presents general conclusions, based on the proposed LSTM model and the evaluation of its classification results, on how to improve and further develop the network.

1.2. Research questions and thematic classification of the thesis

The general research question of this thesis is: Is it possible to classify agricultural crops with LSTM networks by using multi-temporal and multi-dimensional Sentinel 2 images?

Although this general question of feasibility has been approached by others recently (Ienco, D. et al., 2017, Rußwurm and Körner, 2017, Pelletier et al., 2019) it still is an open challenge. The general relevance and topicality of the question is underlined by Kamilaris and Prenafeta-Boldú who write: *"More approaches adopting LSTM or other RNN models are expected in the future, exploiting the time dimension to perform higher performance classification or prediction.* (Kamilaris and Prenafeta-Boldú, 2018, p. 78)."

To further differentiate and focus the main research question a number of secondary questions have been formulated. These originate from the geographic and geo-informatics background of this thesis and focus on the analysis of the used spatial information in relation to NNs. The secondary questions are:

1. How can a network differentiate between classes with very similar features as are to be expected for certain crop types, e.g. wheat and rye.
2. What influence do general characteristics, e.g. quantity, and spatial characteristics, e.g. location and distribution, of the input data have on the classification accuracy of the NN?
3. How can most of the available multi-temporal and multi-spectral information be used effectively in the classification process?
4. What DL approach allows the classification of extensive areas, with available or easy to create labelled y polygon-datasets and consumer-grade hardware within reasonable time?

Given the topic, the research question, the used methods and data the thesis can be classified within the general field of geoscience. Within that field, it touches three distinct scientific disciplines. The general challenge of classification of aerial raster data and its pixel values, as it is used as input to the NN, lies within the discipline of remote sensing. The parts focusing on automatic and intelligent data analysis is part of the machine learning discipline. Lastly, the discipline providing the methods to prepare and analyse the spatial data is geo-informatics.

1.3. Current state of the art of the scientific research

The focus of the current development of aerial image analysis in general and within agricultural applications are intelligent image analysis techniques and methods to classify aerial images automatically. One focus of automatic image classification is to incorporate the vast amount of data and its characteristics which are: multi-temporal and multi-spectral (Kussul et al., 2017 and Kamilaris and Prenafeta-Boldú, 2018). Methods used for automatic aerial image classification are, e.g.: support vector machines, random forest classifiers, linear polarizations, wavelet-based filtering and regression analysis. Besides these the most promising technique to analyse multi-temporal and multi-spectral images is DL. In the task of aerial image classification DL surpasses other machine learning methods in overall and class-specific accuracy as shown by multiple studies, e.g. Ienco, D. et al. (2017), Kussul et al.(2017), Mou et al.(2017), Rußwurm and Körner(2017), Pelletier et al.(2019). In a general overview and comparison study about the state of the art of DL and other machine learning techniques for image analysis in agricultural applications, Kamilaris and Prenafeta-Boldú conclude: "*Our findings indicate that deep learning offers better performance and outperforms other popular image processing techniques.*" (Kamilaris and Prenafeta-Boldú, 2018, p. 78). Within the context of DL different types of NNs are used to analyse aerial images. In the summary of Kamilaris and Prenafeta-Boldú out of 40 analysed studies, 31 papers use a CNN architecture, while 5 papers use LSTM networks and the rest other types of networks *(Kamilaris and Prenafeta-Boldú, 2018)*. Thus, the most used network type for aerial images are CNNs which are specialized in analysing visual images and detecting objects. As Kussul et al. show CNNs can already be used to convolute images in the spatial and the spectral dimension and therefore detect objects using all available spectral bands *(Kussul et al., 2017)*. A disadvantage of pure CNNs is their lack of using temporal information, i.e. information gained by taking an image multiple times at certain time intervals. A huge potential source of information is left out by using only mono-temporal input data. As Rußwurm and Körner point out:

> "[...] *some land cover classes such as, e.g., vegetation and especially crops are difficult to classify by mono-temporal approaches, as vegetation changes its spectral and textural appearance within its species-dependent growth cycle. Especially crops develop these temporal dynamics in a systematic and thus predictable manner, dependent on phenology and the applied crop calendar." (Rußwurm and Körner, 2017, p.551).*

Multi-temporal data is abundantly available for aerial images because earth is monitored constantly from above by various platforms and therefore specific areas are recorded over and over again. Thus, there is an enormous potential of using the temporal information of consecutive aerial images. For that reason, RNNs, or theirs subtype LSTMs, are the second most used method. The capability to analyse sequential information within consecutive aerial images with such networks is an ongoing topic in the scientific community. One of the state-of-the-art method used to analyse sequential data are LSTM networks. The superior classification capabilities of LSTM networks compared to mono-temporal networks is for example shown by Rußwurm and Körner, who compare CNN with LSTM networks. They conclude: *"These experiments have shown that LSTM and RNN networks are able to directly utilize temporal information, namely phenological characteristics of crops, for classification and achieve superior results compared to models which—by design—can not benefit from these features but solely rely only on spectral and textural characteristics."* (Rußwurm and Körner, 2017, p.556)

This thesis builds upon the idea of the articles of Rußwurm and Körner and Ienco, D. et al. who used LSTM networks to classify images (Rußwurm and Körner, 2017, Ienco, D. et al., 2017). However, their approaches are changed and adopted by:

- using only selected field objects as input data and not the total coverage of an aerial image
- using the average pixel values of a field object of each Sentinel 2 channel
- using a higher rate of Sentinel 2 images from in total 143 different dates
- using the best, i.e. most cloud-free, pixel values within a small time window for each field

One common concept used in the above described studies using CNN or LSTM for classification is the approach to classify the totality of an area. This means that the machine learning algorithm determines all pixels or objects of a selected area without any gaps. However, this completeness comes at expenses which are:

- High computational cost requiring sophisticated software, powerful hardware and long computation time. This is especially true if all available bands of an aerial image and all available images of a time series are used.
- An area-wide classification without a focus increases the chance of misclassification, especially for small and very similar classes.

- An area-wide classification requires an area-wide labelled dataset, containing the land cover class of each pixel. This y data is required to provides the NN with the correct solution for the input data x during the training process. Such an area-wide label dataset, if available, is often inaccurate because the label data of individual pixels or pixel areas is outdated, regarding the current Sentinel 2 scene. Depending on the extent of the inaccurate labels this results in more or less bad classification results. If such a label dataset is not available it has to be created, which is an extensive task even for a small area.

There are different ways to overcome these obstacles. This thesis proposes to compress the amount of input data. Specifically to reduce the amount of image input data from a single image and compensate for this by using pre-selected areas, by using all available image bands and by selecting the best fitting pixel values within a time window for each field individually. Three steps are used to achieve this:

1. Selecting the polygons of interest to be classified, e.g. fields.
2. Collecting sequential aerial image data of the areas of interest using all available image bands.
3. Training of a LSTM to classify only these polygons of interest, rather than the whole aerial image. The NNs input data is generated by calculating the average pixel value of each polygon for each image band for multiple consecutive aerial images. While the total number of images is fixed, the actual date of the image can differ within a given time window (e.g. 4 days) from polygon to polygon. This increases the chance of finding the best, i.e. cloud-free, pixel values for each polygon.

This approach has, of course, its own limitation and is not usable for all types of projects, but it is especially applicable for any kind of large-scaled monitoring and classification scenarios. Examples for such projects are (among others): monitoring of certain features of nature reserve areas, classification of plant types in forest or agricultural fields. This thesis uses the classification of crop types in an agricultural setting as an example of the proposed method. In these kinds of projects, the disadvantages of the developed method are less impactful, while the advantages are more impactful. The following list describes the disadvantages of the developed method, in contrast to an area-wide classification:

- Less extensive geoinformation: Only small parts of the total area are classified, because not the whole image, but only selected areas are observed. The impact on monitoring and classification projects is of a lesser degree as they often focus on selected areas and thus mostly don't require an area-wide, but rather a focussed extraction of spatial information.

- Temporal data needed: As monitoring projects focus and observe developments and thus require data from multiple points in time, sequential temporal data is often already available.
- Additional geoinformation needed: Besides the aerial image, additional geoinformation in the form of polygons of the areas of interest is required. Monitoring and classification project usually predefine area(s) to be monitored or classified. Thus, the necessity of additional selected areas is already full-filled. Furthermore, the last decade saw a significant rise in the availability of geoinformation for large areas, provided by official administration or crowd based data collection. Many datasets already contain polygons, e.g. forests, agricultural fields, for vast areas which can be used to avoid computational intensive, area-wide object segregation. Therefore, other project types, which do not require the most up-to-date object segregation extracted from aerial images might benefit from the proposed method as well.

Besides these disadvantages there are benefits when using the proposed approach:

- Differentiation of similar features: The focus on selected areas allows to only use those pixels within the selected areas. A network is then able to learn from those areas (and not the whole image) and thus differentiate between areas with similar features.
- Usage of different dated images for each polygon: By using different dated images for each polygon, within a given time window, the most fitting image can be selected for each polygon individually, rather than selecting one scene (per time step) for all polygons.
- The labelled dataset, containing the land cover classes of the field polygons, is much easier to obtain, than for an area-wide classification, because each polygon contains only one crop type. Such a labelled y polygon-dataset is often already available and is less error-prone. Since the average pixel values of a field are used, individual pixels of the input x not overlapping correctly with the y data, are much less disturbing for the training of the NN.
- Lower computational cost: A comparable low computational cost, which allows a fast training of the NN using only standard consumer-grade CPU and GPU hardware. A short training time enables the user to test more hyperparameters and find an optimal set. Also, the low computational cost enables the trained network to classify tens of thousands of data samples in a matter of minutes. Especially for projects covering large spatial areas and/or time periods, this method allows a much faster training time during development of the NN and a much faster prediction time once the model is implemented.

Part I: Theoretical basis

The practical implementation of DL in image analysis is based on many theoretical foundations. This chapter describes theses theoretical basics which are required to develop the data preprocessing and the LSTM. Part I is split into three parts, describing the used input crop data, the tools used in this thesis and the background of NNs.

2. Input data

This chapter presents a look at the input data. This thesis uses two types of data. As usually classified in geo-informatics these two types are raster data and vector data.

Both types of data will help to produce the final dataset which becomes the input for the NN. The raster data provides the pixel values, while the vector data limits the raster to the relevant areas (i.e. the fields) and provides the actual crops planted on each field.

2.1. Sentinel 2 raster data

The different Sentinel 2 images provide the basis for the input data which the NN uses to detect patterns. The actual structure of the data, the used Sentinel 2 bands and the practical steps taken to preprocess them are described in chapter 6.2. This chapter will provide a short overview of the general characteristics of Sentinel 2 data and why it is suitable to be used in LSTMs to detect crop types.

The research question itself defines the required properties of the input data (cf. chapter 1.2): It has to be multi-temporal and multi-spectral. Sentinel 2 data fulfils both of these criteria. The exact specifications of the Sentinel 2 satellites and its recorded images can be found for example at Drusch et al., 2012 and the Sentinel 2 - Online User Guide (S2-oUG). A recorded Sentinel 2 scene consists of 12 bands ranging from a wavelength of 443nm up to 2 190 nm with a maximum resolution of 10m up to 60m, depending on the band. Especially the different spectral bands in the near-infrared wavelength allow to detect the status and differences of plants. These spectral bands are also used to calculate vegetation indices such as the normalized difference vegetation index (NDVI) or the enhanced vegetation index (EVI) which are commonly used to observe vegetational changes over time. However, these indices will not be used as input because the NN is supposed to detect this relationship between the image bands and the plant types by using the raw pixel values provided as input. Furthermore, other spectral bands, which seem non-relevant at first, are used to support the crop classification. *"Further spectral bands are often discarded, even though that information is perceived by the satellite and may also contribute to the classification procedure." (Rußwurm and Körner, 2017, p.552).* They provide helpful information, such as the aerosols or water vapour in the air which influences the pixel values of other channels. By using most of the available channels the NN can detect basic patterns, such as: if the aerosol value is high, other channels have altered values, without the crops actually changing.

The two identical Sentinel 2 satellites revisit any area over central Europe in roughly 2 days, i.e. in average every 1-3 days there is a new scene of the same area available. This allows the close monitoring of the vegetation growth cycles. *"Vegetation follows specific periodic growth cycles determined by the plant's biology. The study of these cycles is known as phenology and describes characteristic events such as germination, flowering, or senescence."* (Rußwurm and Körner, 2017, p.551-552). The characteristics of these growth cycles become visible in the reflective spectral values, i.e. the pixel values, of the image. Multiple sequential images show typical patterns not only in the spectral values of the different image bands, but also in the temporal dimension within one band. *"Phenological characteristics* [of crops; note from the author] *are assumed to change in a predictive manner and can thus be utilized for identification, as long as farming practices and environmental conditions remain unchanged or are considered in the model.*" (Rußwurm and Körner, 2017, p.551-552).

In conclusion, the properties of Sentinel 2 scenes are appropriate as input for a LSTM model. The Sentinel L2A data includes complete atmospheric correction and a basic scene classification which helps to detect agricultural areas (cf. chapter 6.2 and Level-2A Algorithm Overview). Furthermore, the data is freely available and covers large areas entirely. Besides these advantages, Sentinel 2 scenes consist of multiple spectral channels and follow a dense sequential data acquisition. This generates a dense sequential data pattern which perfectly fits the required input for LSTM models (cf. chapter 4.2.4).

2.2. Vector data of agricultural subsidies

The original vector data is provided by the "Thüringer Landesverwaltungsamt" (engl: Ministry of Administration of Thuringia) which oversees the EU subsides of the CAP in the Free State of Thuringia. The basis of the dataset are the subsidy applications from the year 2018. In the context of the Integrated Administration and Control System, EFTAS Fernerkundung Technologietransfer GmbH was commissioned to manually control the data, in June and July 2018, on the basis of aerial images and local field controls. The dataset was edited based on the rules of the CAP. Among other aspects, the boundaries of the fields and the crop types were corrected, if there were any indications that the application was wrong. It can thus be expected that the data contains (almost) no errors. With permission of the "Thüringer Landesverwaltungsamt" this corrected dataset was then used for this thesis. For privacy protection reasons only the minimal necessary amount of information was provided. The dataset consisted of vector data containing the polygons of all field samples and two attribute columns:

1. Anonymised Field ID
2. Crop type

The dataset contained almost 20 000 polygon samples. Of these a larger portion is not directly used for agricultural purposes, e.g. because the area is protected due to environmental reasons, temporarily unused or contains no crops. The filtering of the data is described in detail in chapter 6.1 - Data preprocessing of the vector data. The final training dataset has 11 687 samples. Table 3 (in chapter 6.1) provides a detailed overview of the different amount of samples per crop class.

3. Methods and Tools

This chapter provides an overview of the most important soft- and hardware used to prepare the data and to develop, train and evaluate the LSTM. As most of these tools are well known only a short introduction is given, not to comprehensively describe them, but to present a complete documentation of the used tools and methods.

3.1. Python

Python is a high level, general purpose programming language distributed by the Python Software Foundation ("Python – SF"). Python was first released in 1991. The current Python 3 version, which is used in this thesis, was released in 2008. It runs through an interpreter, thus python code can be run without the need to compile it. This allows for fast development and testing of code. Due to this and the huge amount of external libraries python is a widely used language in many scientific projects. It is also one of the most used language to develop NNs because it offers many powerful DL libraries, e.g. TensorFlow, Pytorch, Theano, Caffe etc. Furthermore, Python offers an almost limitless amount of other libraries useful for data preprocessing. The following three libraries were used for this purpose:

1. **Numpy**
   Numpy is a module that allows for scientific computing in Python. Its most used feature, in the context of this thesis, is the creation, editing and calculation of N-dimensional array objects ("NumPy").
2. **Pandas**
   Pandas is a software library to structure data and manipulate large numerical tables with high performance ("Pandas"). It is used to preprocess the input data, i.e. structure it, and manipulate the huge tables containing more than 2 000 columns (cf. chapter 6.3.1) and to evaluate the output matrix of the NN.
3. **Scikit-learn**
   Scikit-learn is a software library for machine learning. It offers many methods helpful for preprocessing data and evaluate prediction results ("Scikit-learn").

3.2. Keras

Keras is a NN API running on top of TensorFlow among others. This means it offers much easier access to create models in TensorFlow than pure TensorFlow. It, therefore, allows to focus on fast experimentation and implementation of a NN. In this thesis Keras version 2.2.4, which was published in October 2018, is used. The main features of Keras are: (“KERAS”)

- Easy and fast prototyping and implementation of NNs through user-friendliness, modularity and extensibility.
- Support for ANN, CNN and RNN, as well as a combination of those.
- Can calculate NNs on CPU or GPU.

The LSTM created in this thesis could have been developed with a different Python Machine Learning library. Using the same input data and network parameters, e.g. number of layers and neurons, type of activation function etc., the same or very similar results are to be expected. Keras was selected because it offers a good balance between the needed complexity and accessibility. It is therefore used as the core library to code the general architecture of the LSTM network, its layers, optimisers, callback functions etc. (cf. chapter 7)

3.3. Hardware

All training of the NN was done on a computer with the following specifications:

Type	Model
CPU	AMD Ryzen 7 2700X (3.7GHz, 8 cores)
GPU	8GB Asus GeForce RTX 2070 Dual OC active
RAM	32 GB (DDR4 – 3 000 MHZ)
Hard Drive	500GB Samsung 970 Evo M.2
Operating System	Windows 10 Pro

Table 1: Hardware used to train the NN (own table)

Thus, the given training times for the NN in chapter 7 are referring to this set of hardware. Other hardware configurations will obviously train the same NN with the same dataset faster or slower depending on the performance of the individual parts. The most influential hardware part regarding the training time is the GPU since it handles the most computationally intensive calculations of the NN.

4. Neural Networks and Deep Learning basics

The Terms: “Artificial Intelligence”, “Machine Learning”, “Artificial Neural Networks” and “Deep Learning” are often mixed up to describe the same or very similar concepts and ideas in software development. So it is no surprise that, although these terms come from different times and development periods, they share a common ground, which is: A machine is given not the instructions how to solve a problem, but rather the instructions to learn and to find the best (or a good) solution to a problem. This is in contrast to conventional programming which defines specific subtasks and instructions to solve a bigger problem (Nielsen, M.A., 2015). To enable computers to learn from experience they have to understand the world as a hierarchy of concepts, where each concept is defined through its relation to simpler concepts. This hierarchy of concepts allows the computer to combine simpler concepts to understand more complex ones without the need of a human to formally specify all this information. “*If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning*.” (Goodfellow et al., 2016 p. 1-2)

The following subchapters will take a look at this concept, at its history and development and at special forms of DL, which are ANN and LSTM networks. The latter is the network type developed in part II of this thesis.

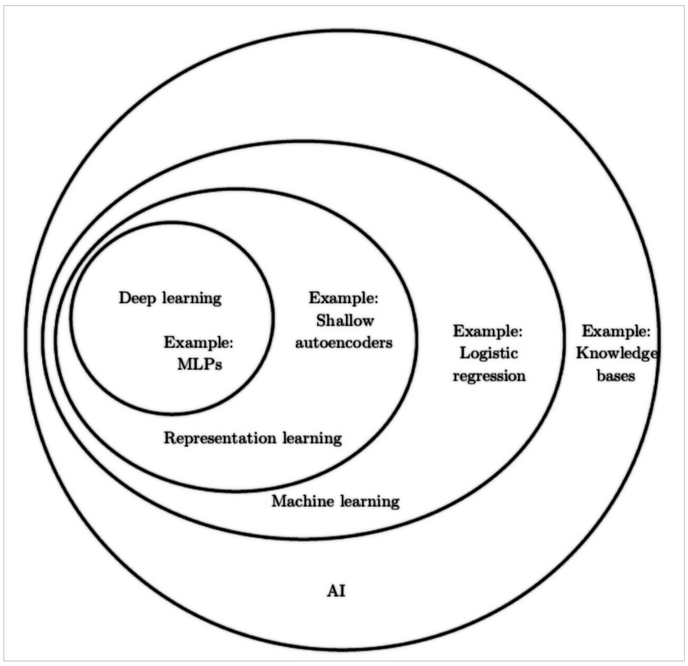


Figure 1: Relationship between AI disciplines (Goodfellow et al., 2016, p. 9)

4.1. Overview and history of machine learning and Deep Learning

DL is a very new development in the wider field of machine learning and AI. These are broad paradigms in which DL is one approach to AI. In short DL “*[...] is a type of machine learning, a technique that enables computer systems to improve with experience and data.*“ (Goodfellow et al., 2016, p. 8). The relationship between the different AI disciplines is shown in figure Figure 1. As this thesis uses only DL methods, they will be the focus of this theoretical chapter thus, other methods of machine learning and AI are left out by intention.

DL methods have a longer history than one might suspect, given the only recent trend to use them widely. The reason for this is the rebrands of the same concept as there were different ups and downs in development and popularity and different perspectives on AI. The three major waves of DL and the term are (Goodfellow et al., 2016).:

1. Cybernetics (1940-1960)
2. Connectionism (1980-1990)
3. Deep Learning (2006-Today)

The first wave of DL, i.e. Cybernetics, was inspired by building computational models of biological learning. The goal was to create machines that would work similarly like brains work. This neural perspective was driven by the idea that brains (from animals or humans) were a working example of a neural network. Thus, as neuron cells in a brain follows a "simple" logic in forwarding synaptic signals, so should simple computational units become "intelligent" when connected to each other. But these first models were only linear models and thus had many limitations, most famously they couldn't learn the XOR operation. The failure to deliver for real-world applications lead to the first so-called AI Winter ("Wiki-AI-winter").

The second wave of DL focussed even more on the importance of the connection of simple units to solve complex problems, hence the name "Connectionism". A number of important concepts were developed during that time which are still in use nowadays, e.g. distributed representation and back-propagation. In the mid-1990s the second wave crashed as AI failed again to deliver in highly ambitious promises (e.g. voice recognition, translation of languages).

The third wave of DL started in 2006 with a breakthrough in the way "deep" networks can be trained. These deep networks were known since the 1980s, but weren't used due to the difficulty in training them efficiently.

Besides this theoretical breakthrough, other important developments contributed massively to the success of the third wave of DL and made earlier developed concepts more and more usable. The most important are (Goodfellow et al., 2016).:

1. Hardware: The steady and significant rise in CPU and GPU calculation power made larger and deeper neural network models possible.
2. Digital Data: The rise of digital sensors (e.g. digital cameras, smartphones, …) and digital storage capacity during the 2000s lead to an abundance of digital data that was easy to ac-

cess and use for DL. Thus, it was possible to provide these algorithms with the amount of data they need to succeed.

3. Software Development: The development of software infrastructure and libraries such as Theano, PyTorch, TensorFlow, Keras etc. made the access to DL easier and in return supported research and commercial projects which helped to further develop DL.

Although neuroscience is still a source of inspiration for modern DL, as it inspired the basic idea of connecting a lot of simple computational units to create intelligence, the goal is no longer to create biological learning, but a working artificial equivalent. Mainly because there was and still is not enough information about the real processes going on inside brains, DL started more and more to use techniques from different fields such as linear algebra, probability theory, information theory and numerical optimization. In the years following the third wave (up until today) deep NNs started to outperform other classical developed software and AI systems based on other machine learning concepts. DL is nowadays used in many diverse domains like: Computer Vision, Speech Recognition, Image and Object Classification (e.g. in Remote Sensing) and many more. (Cresson, 2018, Goodfellow et al., 2016, Raschka and Mirjalili, 2018).

Machine learning and different methods and models of DL, as it is in use today, can be assigned to one of three general types (Raschka and Mirjalili, 2018):

1. "Supervised Learning" uses labelled training data to learn general patterns of data. Based on this the model can make prediction on new unseen data of the same type. This can be used for classification problems, i.e. the computer has to predict to which category an input belongs. Another example of "Supervised Learning" are regression problems, i.e. the computer has to predict a value as close as possible to a future value of a numerical input value(s), e.g. stock price of the next day based on different input values.
2. "Reinforcement Learning" tries to improve a model or agent based on interaction with its environment. The performance of this interaction is usually reported back to the model via a so-called reward signal. This feedback can be seen as a label, i.e. information about the data, and thus reinforcement learning is related to supervised learning.
3. "Unsupervised Learning" is different from the above described types because it explores unlabelled data with an unknown structure. The goal is not to label the data within a context but to find a structure within the data itself, e.g. clusters, and learn the probability distribution of the data or other properties of the data distribution.

As there are far too many different types of DL models to cover in this thesis the theoretical overview will only cover "Supervised Learning", more specifically classification problems with ANNs, which are the basis for the later described LSTM.

4.2. Artificial Neural Networks

4.2.1. The concept

As described in the last chapter DL algorithms are not specifically instructed to solve a problem, but rather algorithms that learn from data how to solve a problem by detecting patterns in the training data. To "learn", in the context of software, is defined by Mitchell as: "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*" (Mitchell, 1997). Following this definition learning itself is not the task, but rather the means to get the ability to solve the task. Tasks in DL are in often easy to solve for humans, but hard to describe formally, e.g. speech or image recognition. For a human, it is quite easy to detect certain objects, e.g. a car on an image. Considering all the different types of cars, the colours and the different angles an image can show a car, it seems almost impossible to formalize specific rules for a computer to detect all possibilities of images showing a car. In this situation DL can provide a solution by learning to detect any car on any image.

The specific problem to solve in this thesis is a classification task. "*Classification is a subcategory of supervised learning where the goal is to predict the categorical class labels of new instances, based on past observations.*" (Raschka and Mirjalili, 2018 p.3). A DL model is given a training set with an input vector x for each sample of the training set and the associated correct category y. The vector x contains multiple features which identify each sample. The task for a NN is to detect feature patterns which define a category of data samples. Then when an unknown feature is given into the NN, it can categorise these with a certain probability to one of the trained classes. Thus, the training of a NN can be understood as the search for a function which fits closest to $y = f(x)$. A trained NN then uses this function to predict new inputs.

The power and usability of ANNs is expressed in the "universal approximation theorem" (Goodfellow et al., 2016). Nielsen describes it as: "*One of the most striking facts about neural networks is that they can compute any function at all.*" (Nielsen, 2015, chapter 4). "Compute" in this definition means to give a very good approximation of any function. This is one key aspect of ANNs: Their algorithms "[...] *solve mathematical problems by methods that update estimates of the solution via an iterative process, rather than analytically deriving a formula to provide a symbolic expression for the correct solution."(Goodfellow et al., 2016 p.78).* To give an exact solution would be very time-consuming due to the multidimensional nature of the functions, but most importantly it is impossible from a logical position, as Goodfellow points out:

> "*Learning theory claims that a machine learning algorithm can generalize well from a finite training set of examples. This seems to contradict some basic principles of logic. Inductive reasoning, or inferring general rules from a limited set of examples, is not logically valid. To logically infer a rule describing every member of a set, one must have information about every member of that set. In part, machine learning avoids this problem by offering only probabilistic rules, rather than the entirely certain rules used in purely logical reasoning. Machine learning promises to find rules that are probably correct about most members of the set they concern."* (Goodfellow et al., 2016 p.114)

Since these function can't be solved analytically a NN "only" finds a (very) good solution with a guided try and error method. To guide the network and give it feedback, it needs a way to evaluate its performance *P*. For classification tasks two performance indicators are commonly used: accuracy and error rate. Accuracy is the proportion of the input the model predicted correct, whereas the error rate is the proportion of the input the model predicted incorrect (Raschka and Mirjalili, 2018).

The experience *E*, mentioned in the above definition of learning, is the totality of the data set provided to the NN. In classification tasks, *E* consists of the input samples *x* with their associated features and their category labels *y*. Given the same quality of data, the larger *E* the better the result. Goodfellow gives a rough rule of thumb about the size of *E* for supervised learning: With 5.000 examples per category DL achieve acceptable performance and with 10 million examples it will match or exceed human performance. (Goodfellow et al., 2016, p.20).

Goodfellow et al. points out that only a few general elements are needed to achieve the learning abilities described above: *"Nearly all deep learning algorithms can be described as particular instances of a fairly simple recipe: combine a specification of a dataset, a cost function, an optimization procedure and a model."* (Goodfellow et al., 2016 p. 151). However, the information and data flow taking place within the model is not simple. The following subchapter takes a deeper look at the processes within a NN and within the neurons itself.

4.2.2. The neuron

The name ANN derives from the idea of an artificial computational model that is structured similar to how a brain is structured (cf. Chapter 4.1). In 1943 Warren McCulloch and Walter Pitts published the first concept of an artificial neuron, called a perceptron (Raschka and Mirjalili, 2018). Their idea was based on a simplified brain neuron. Neurons are interconnected brain nerve cells that process and transmit chemical and electrical signals. A schema of a biological neuron is shown in figure 2.

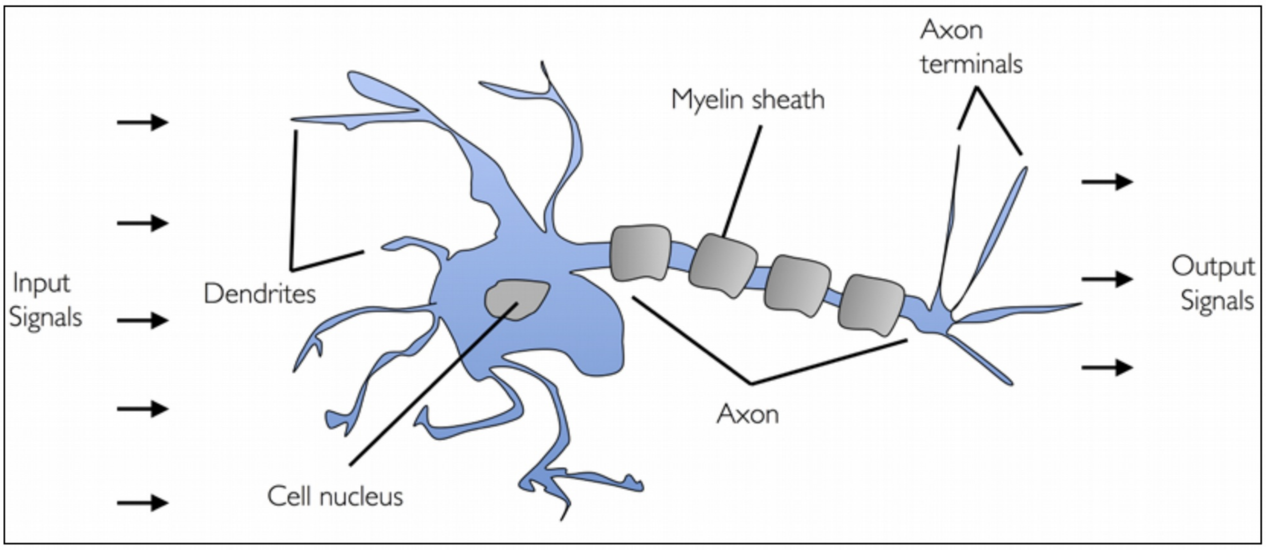


Figure 2: Schema of neuron cell (Raschka and Mirjalili, 2018 p.18)

The brain consists of neurons which receive an input signal, process this signal and produce an output signal. In an analogous manner an ANN consist of artificial neurons which receive and process an input to an output signal (cf. figure 2 and 5). As the "computational" power of the brain comes from the interconnection of many neurons, so does an artificial network gain computational power with the connection of many artificial neurons. Using this concept Frank Rosenblatt developed the first algorithm that could learn to detect the simple form of a square and a circle (Rosenblatt, 1957). Even though this system could only solve linear problems the basic concept is still used today. The smallest unit of Rosenblatt's system is the perceptron. Perceptrons receive as input a vector x containing several features per sample (x_1, x_2, x_3...) which are multiplied by individual weights (w_1, w_2, w_3). The weights are unknown at first and randomly set. The main goal of DL is to learn the general weights w_i for each feature x_i which help to explain the samples class. The higher a weight, the more important is the associated feature to determine the class of x. If, for example, we try to predict the cost of a house, the location in the centre of a city might be more important than the size, although both factors contribute to the price.

Inside the perceptron all features of one sample *x* are multiplied by the weight *w*, then summed up and put into a so-called activation function. (Nielsen, M.A., 2015) This activation function outputs a result based on the value of the summed up input. The very simple activation function used in Rosenblatt's system simply returns 1 if the input is above a certain threshold or 0 if not. It is graphically defined in figure 3 and the following formula:

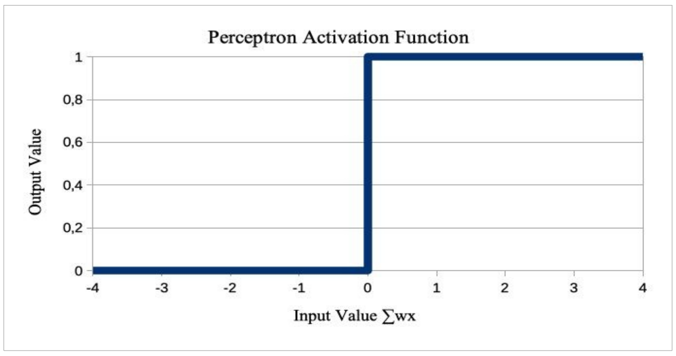
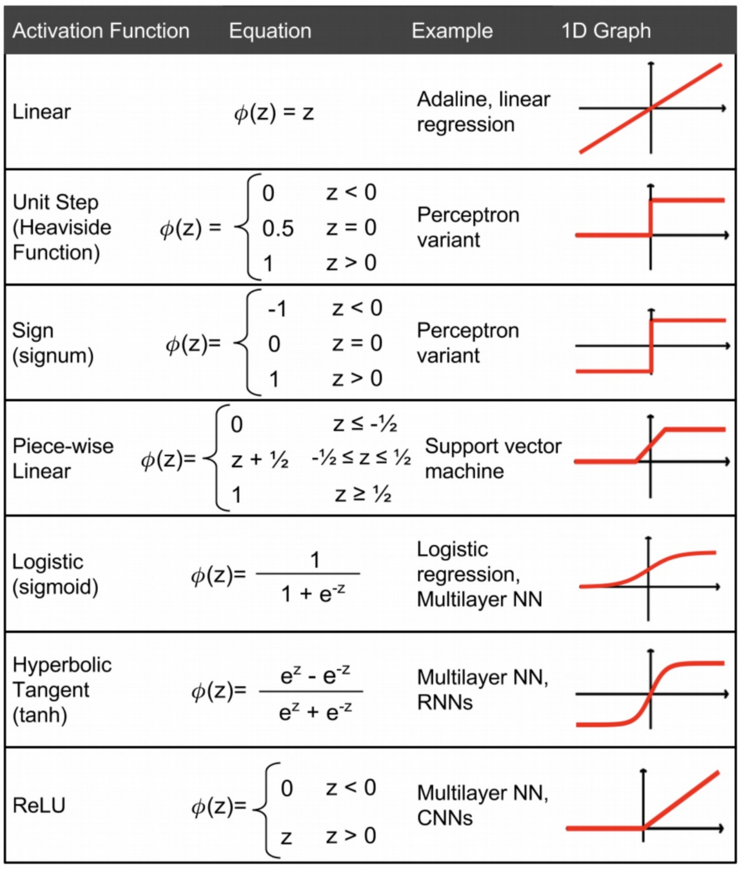


Figure 3: Step activation function of a perceptron with a threshold of zero (own figure)

$$output = \begin{Bmatrix} 0 \, if \ \Sigma w_i x_i \leq threshold \\ 1 \, if \ \Sigma w_i x_i > threshold \end{Bmatrix}$$

A perceptron is basically a small device that weighs different pieces of evidence to make a decision. But the true power of a perceptron only becomes visible when multiple of them are joined together to a network, so that the output of one becomes the input of others. Figure 6 shows such a network.

The main problem with a perceptron is its limited activation function. *"In fact, a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1. That flip may then cause the behaviour of the rest of the network to completely change in some very complicated way."* ((Nielsen, M.A., 2015, chapter 1). To solve this issue multiple activation functions have been developed so that a small change in any weight causes only a small change in the output. This new form of a perceptron is called an artificial neuron (or simply neuron) and constitutes the smallest unit of any modern NN. With these new types of activation functions a small change in input causes only a small change of a neuron's output and thus only a small change in the final output of a NN. Widely used activation functions are e.g. Logistic Sigmoid Function, ReLU Function, Hyperbolic Tangent Function. Depending on the type of network different activation functions work best. For the LSTM network developed in part II the Hyperbolic Tangent Function will be used. An overview of different activation functions, the mathematical formula, the models to use them in and a graphical representation is given in figure 4.



Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Figure 4: Overview of activation functions (Raschka and Mirjalili, 2018, p. 450)

The structure of an artificial neuron is presented in figure 5. It summarises the concept presented in this chapter: An artificial neuron receives multiple inputs which are each individually weighted and then summed up in the transfer function. The net input is inserted into a specific type of activation function which returns an output.

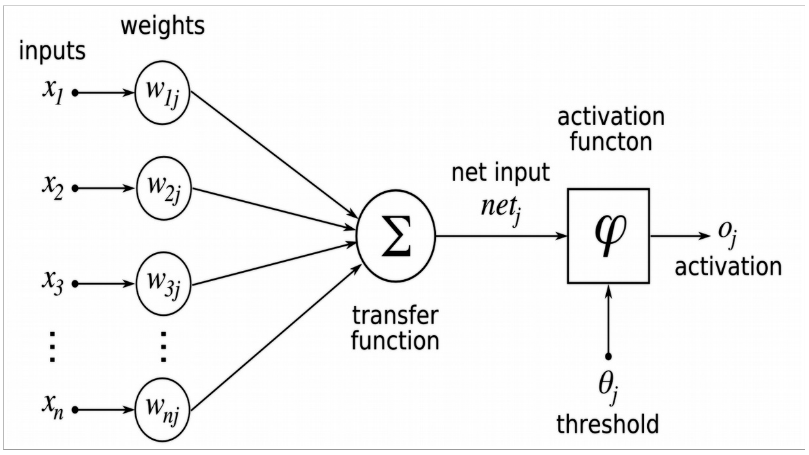


Figure 5: Schema of an artificial neuron ("Wikimedia-ANN-Model")

4.2.3. The network

This section will focus on the general data flow and its manipulation through the network that enables an ANN to learn, i.e. improve its performance on a given task. The network architecture always has 3 general elements (cf. figure 6):

1. Input layer: The input vector containing all the information ($x_1 \ldots x_i$) of one data feature.
2. Hidden layer(s): The connected neurons processing and forwarding the input data.
3. Output layer: The output neuron processing the last hidden layer and then outputting a final result.

Figure 6 shows an example of an ANN with an input vector of size five, three fully connected hidden layers and three categories to predict. These simple network parameters were chosen as an example. A network could have different types and numbers of hidden layers, connections and number of neurons in each layer. However, this topic is too extensive to be discussed in great detail in this thesis because these type of parameters depend heavily on the type of network and the task that should be learned. A deeper look at the actual structure of the NN used in this thesis, a LSTM Network, will be given in the next chapter (cf. chapter 4.2.4).

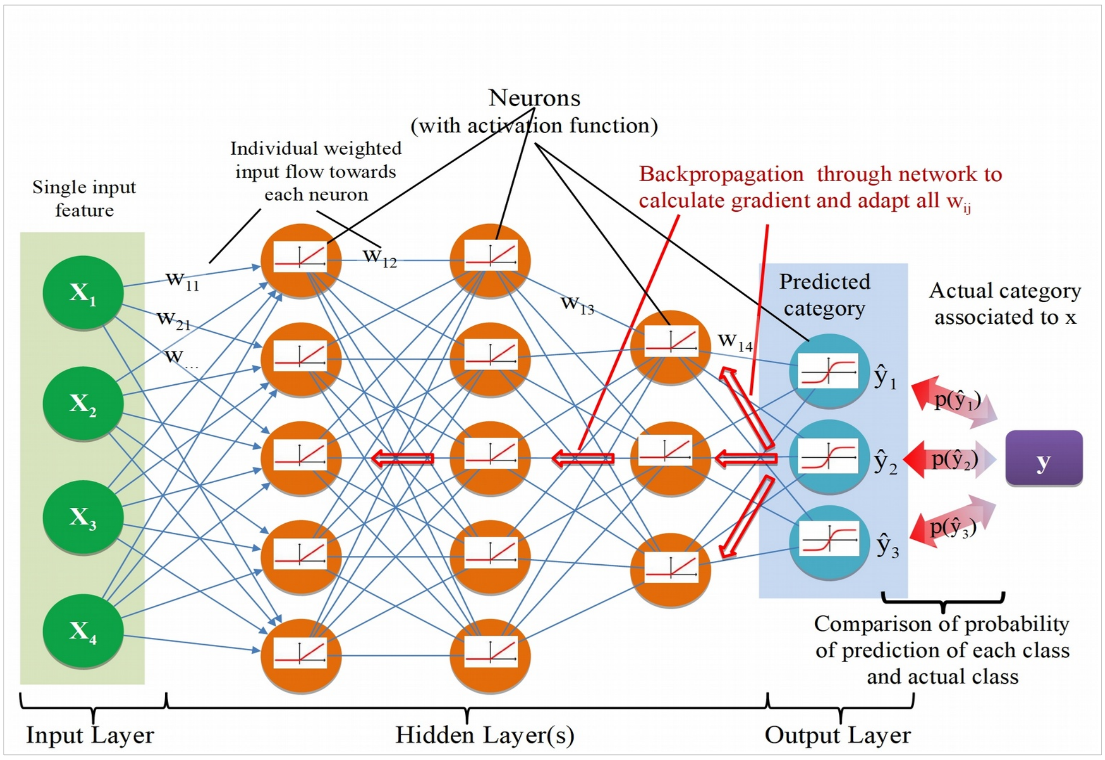

Neurons
(with activation function)
Individual weighted input flow towards each neuron
Single input feature
Backpropagation through network to calculate gradient and adapt all w_{ij}
Predicted category
Actual category associated to x
X_1
X_2
X_3
X_4
w_{11}
w_{21}
$w_{\ldots}$
w_{12}
w_{13}
w_{14}
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$p(\hat{y}_1)$
$p(\hat{y}_2)$
$p(\hat{y}_3)$
y
Comparison of probability of prediction of each class and actual class
Input Layer
Hidden Layer(s)
Output Layer

Figure 6: ANN and the information flow within (own figure)

The network example presented here is a standard deep feedforward network. "*These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define, and finally to the output y.*" (Goodfellow et al., 2016, p.164). A dataset with multiple samples, each containing multiple features, is put into the NN through the input layer one sample at a time. The data put into the input layer has to be preprocessed (i.e. interpolated, normalized etc.) beforehand.

As described in the last chapter, each input to a neuron (blue arrows in figure 6) is individually weighted. At the first run, these weights are initially set with small, randomized values close to 0. Then all inputs are summed up inside each neuron and put into an activation function which generates an output value. In classification tasks with standard feedforward usually the ReLU activation function is used (cf. figure 4). As figure 6 shows, the output of one layer becomes the input of the next. This first phase of information flow through the network is called "forward propagation".

As the input x propagates fully through the network into the output layer, the category of each input sample is determined. The output layer transforms the values to match the categories y associated with the input x. To do this in classification tasks usually a softmax activation function is used which is strongly related to a sigmoid activation function (cf. figure 4). A softmax function returns a value between 0 and 1 for any input while at the same time all components add up to 1. Through the softmax function the prediction output can be interpreted as confidence percentage for each category, i.e. how certain the network is that the given input x falls into a certain category y, in the case of figure 6 (y_1,y_2,y_3,...) The predicted output $\hat{y}$ is then compared to the actual correct category y. Based on the type of network and task to solve, different cost functions are used to calculate the cost between the actual class and the confidence value of the predicted class. To get the influence of the whole dataset all samples of the input x are propagated through the network. The total sum of the difference between the correct and wrongly assigned categories is applied to the cost function. Different types of cost functions like the mean square error function or the cross-entropy function, for categorical problems, are available. The latter cost function *"[...] has the benefit that, unlike the quadratic cost, it avoids the problem of learning slowing down."* (Nielsen, 2015, chapter 3) Thus, it descent faster toward a minimum, while not increasing the chance of overshooting (see below). Obviously, the more samples are predicted correctly, the lower the output of any cost function and visa versa. As described above (cf. Chapter 4.2.1), the general NN function can not be solved analytically with respect to the inputs and weights of all neurons, thus to find a low point a gradient-based optimiser is used.

To calculate the gradient of the cost function an algorithm called back-propagation is used. "*The back-propagation algorithm [...] allows the information from the cost to then flow backward through the network in order to compute the gradient."* (Goodfellow et al., 2016, p.200). It is important to note that in a feedforward network only the information from the cost function, not the information from the input x, flows backward. Once the gradient of the cost function is established through back-propagation, the direction downwards the cost function is evaluated. Then the weights of each neuron can be updated, in a direction that moves along the cost function downwards (cf. figure 7).

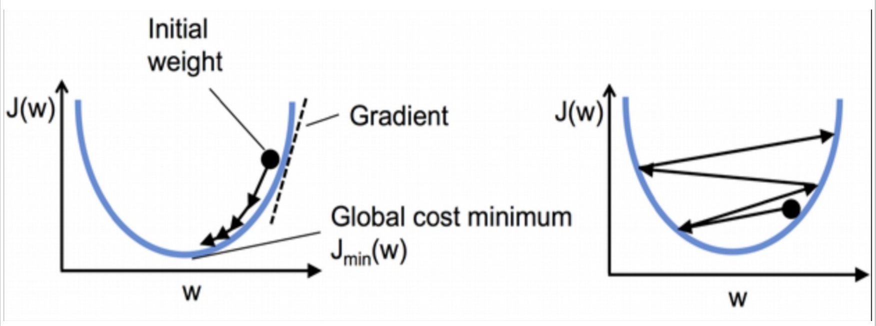


Figure 7: (Left) Example of a gradient descent along cost function J, based on the set weights w. (Right) Gradient Descent with overshooting (Raschka and Mirjalili, 2018, p.36)

The above described process concludes one so-called "epoch", i.e. the data flow of the whole training dataset through the network, the back-propagation, the calculation of the cost function gradient and the updating of the weights. This process is repeated in an iterative process and each time the opposite direction of the gradient, thus the direction downwards the cost function, is evaluated. Thus, the next step descents along the cost function downwards. The set learning rate defines the size of the steps downwards. If the steps are too small, the calculation might take a very long time, if they are too large, an overshooting can occur so that the minimum is missed (cf. figure 7). One solution to this is a learning rate that gets smaller the higher the epoch number. If the global minimum or at least a very low point on the cost function is found, the NNs and its currently applied weights created a good approximation of the function *y = f(x)*. Most cost functions don't have an as easy recognizable form and minimum as the examples presented in figure 7. Goodfellow et al. points out these challenges and the solutions within DL as follows: *"In the context of deep learning, we optimize functions that may have many local minima that are not optimal and many saddle points surrounded by very flat regions. All of this makes optimization difficult, especially when the input to the function is multidimensional. We therefore usually settle for finding a value of f that is very low but not necessarily minimal in any formal sense" (Goodfellow et al., 2016,* p. 82)

To successfully implement gradient descent, despite the complex and multidimensional nature of the functions, most networks use a slightly different, but more resource effective and faster version of the gradient descent. They use a so-called: "Mini-Batch Gradient Descent". In this method, the gradient is not only updated after one epoch (i.e. after each data example has propagated through the network), but after a certain set number of training samples, called a batch, have propagated through the network. Although this usually leads to a more erratic descent, because the descent takes only a fraction of the total input into account, the *"[...] advantage over batch gradient descent is that convergence is reached faster via mini-batches because of the more frequent weight updates.*" (Raschka and Mirjalili, 2018 p.46).

The main challenge when training a NN is that the model must perform well, not only on the training input *x*, but even more importantly on previously unseen inputs. General indicators for the performance of a NN and its ability to generalise well, are:

1. A small training error, i.e. the prediction error of the training input *x*.
2. A small test error, i.e. the prediction error of the unseen test input data.
3. A small gap between the training and the test error.

To achieve this two conditions have to be avoided: "Underfitting" and "Overfitting" (Goodfellow et al., 2016). Underfitting occurs when the model is unable to find a reasonable low training error with its given architecture and input data x. Overfitting occurs when the training error is low, but the test error is significantly higher. Thus, the model can only predict the training data well but fails to predict new unseen data. Figure 8 shows examples for all three conditions on a simple dataset.

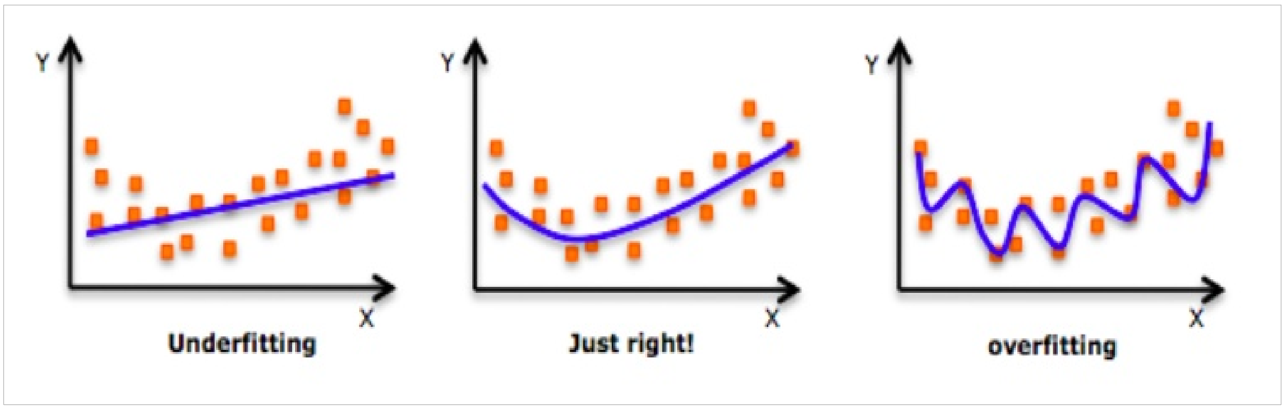


Figure 8: Example of underfitting, generalisation and overfitting. (Gondaliya, A, 2014.)

The example on the left: "Underfitting" shows the prediction line does not fit the training points very well, due to its linearity. The example on the right: “Overfitting” shows the output prediction sticks closely to the sample points and follows every curvature of the training data. The central example: “Just right (generalisation)” shows the output prediction following a line that has the best potential to predict new data well. Of course, to determine the quality of a model in a real world example, we would need to have not only the training results but also prediction results from an independent test sample and then compare the error rates.

To achieve a good generalisation DL algorithms use regularisation techniques by changing different hyperparameters that control the training behaviour. The values of hyperparameters are not learned by the NN but are set and tweaked by the user in advance of the training. These parameters change the training and test error. Example of hyperparameters are:

- Number of Neurons/Nodes in a layer.
- The type of activation function used by the hidden layers and output layer.
- The number of layers within a network.
- The number of epochs a DL algorithm trains.
- The size of the batch, i.e. number of samples, after which the gradient of the cost function is evaluated to update the weights.
- The type of cost function used within the network.

Depending on the type of problem and the specific network type there are numerous other hyperparameters that can be tweaked to improve the performance of an ANN. The different types of hyperparameters and how they affect an ANN can be visually experienced on the website: http://playground.tensorflow.org/ (accessed at 24.07.2019) Though there are some general rules on how to set these hyperparameters, there are no exact rules on which parameters to use for which situations. *“The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error.” (*Goodfellow et al., 2016).

4.2.4. RNN and LSTM networks

The type of ANN presented in the last chapters was a feedforward network. As was shown, these standard NNs learn from a given dataset, whereby the order of the dataset does not matter. In fact, these networks are not capable to process the order of a dataset. Thus, the order in which the samples are passed through the network doesn't change the output result at all. "*Intuitively, one can say that such models do not have a memory of the past seen samples. [...] RNNs, by contrast, are designed for modeling sequences and are capable of remembering past information and processing new events accordingly.*" (Raschka and Mirjalili, 2018, p. 539) Although ANNs and RNNs share many properties as described above, like neurons, cost functions, back-propagation etc., one substantial difference between them is that RNNs use sequences as inputs. Such datasets contain information not only within the samples and their features but also within the order of the samples themselves. One prominent example is language. Characters, word and sentences carry information themselves, but information is also contained within the position of characters, words and sentences within a text. Only the correct position of characters, word and sentences gives a text meaning. Without these sequential information only looking at single characters, words or even sentences makes it impossible to understand a whole text dataset. Another example of sequential information, which is used in this thesis, are sequential images. Such images e.g. satellite images, show the same object but at different points in time. This has two major advantages over a single image: First, a sequence allows to observe the development of an object. Second, through the observation of the development of an object one gets more information about the object itself. RNNs provide a way to do this by not only looking at one set of features of a sample, but also at the previous and following set of features of the same sample. In the following chapter the terms steps, time steps etc. refer to the described concept of sequential information.

Within RNNs information does not only flow forward, like in feedforward networks (cf. chapter 4.2.3) but in cycles. "*These cycles represent the influence of the present value of a variable on its own value at a future time step.*" (Goodfellow et al., 2016, p.368). They are applied within the hidden layer of a network, as shown in the general structure of RNNs in figure 9. The blue box (x) shows the input layer, the orange ellipse (h) shows the hidden layers and the green box (y) the output layer. Each of these boxes represents a full vector and thus contains all the features of one sample of a dataset (the input layer) or multiple neurons (the hidden and output layer). The arrows display the connection of the layers and the flow of information within the network. The individual variables of the vectors are weighted and then summed up and put into an activation function within the neurons, as shown in figure 6. The curved arrow, in the left part of the diagram, represents the cycles within the hidden layers. Without this curved arrow, the illustration would represent a standard ANN. To expose the true structure of an RNN, the right side of figure 9 shows the RNN network unfolded into the time dimension.

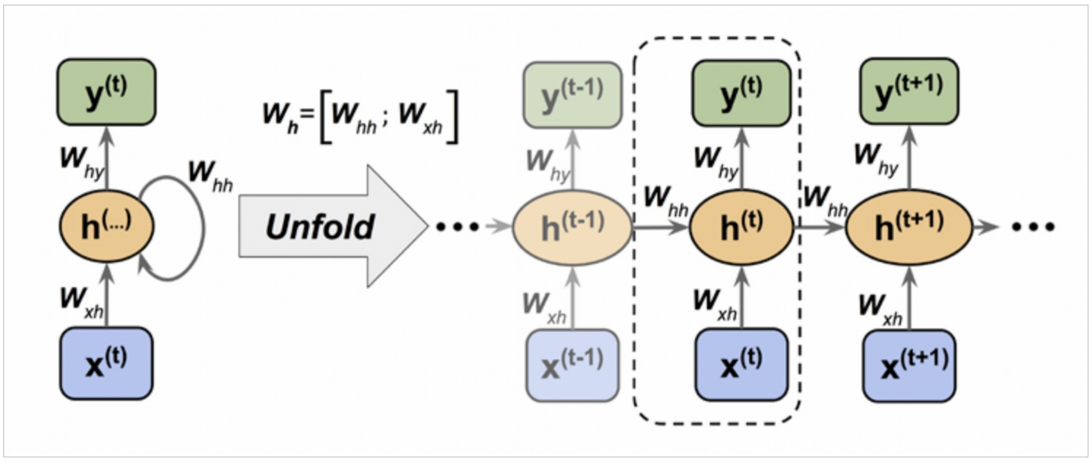


Figure 9: Structure of a RNN (Raschka and Mirjalili, 2018, p. 544)

The full structure of a RNN becomes now visible. While in feedforward ANNs hidden neurons only have one input source (the input vector x), the hidden neurons of RNNs receive two inputs: First from the input layer $x^{(t)}$ (as in a standard ANN), but also from the hidden layer from the previous time step $h^{(t-1)}$. Considering the time or sequential dimension a RNN is like multiple, interconnected ANNs, which recur to themselves. Thus, while a feedforward ANNs can approximate any function (cf. chapter 4.2.1), a RNN can approximate any function involving recurrence, such as $h^t = f(h^{(t-1)};x)$. This is possible due to the concept of "Parameter sharing" (Goodfellow et al., 2016). The idea, developed in the 1980s to improve machine learning and statistical models, is to learn a single model that uses the same parameters on all time steps and sequences, rather than learning a separate model for each time step. "*Learning a single shared model allows generalization to sequence lengths that did not appear in the training set, and enables the model to be estimated with far fewer training examples than would be required without parameter sharing*." (Goodfellow et al., 2016, p. 371).
Depending on the task, the possible input and the needed output different RNN architectures are available. Figure 10 shows different types in contrast to an ANN. Each rectangle represents a vector, containing multiple features or neurons. The colours show the type of the vector or respectively the layer: Input vectors are red, hidden layers are green and the output layers blue. The type of network are from left to right as follows:

- “one to one”, represents a standard ANN, with a fixed-size input and a fixed-size output.
- “one-to-many” shows a RNN which takes a single vector input and produces a sequential output. An example task is a model that takes a single image and captions it with multiple words.
- “many-to-one” displays a RNN which takes a sequential input (of multiple vectors) to process a single output. An example is a sequential series of satellite images which is used to classify plants visible on the time series. Thus, this is the type of architecture that is developed and used in part II.
- “many-to-many” shows a RNN which takes sequential inputs and then computes sequential outputs. An example task for this architecture is language translation. Multiple words in language A are translated into multiple words in language B.
- “many-to-many” (synced) represents a RNN which takes sequential inputs and calculates synchronised a sequential output. An example is a video classification that labels individual frames of a video.

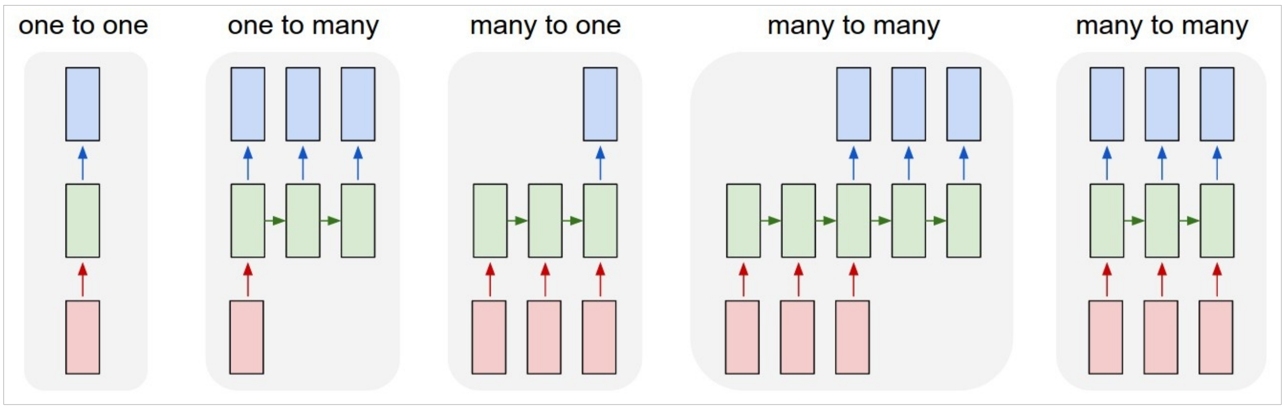


Figure 10: Standard feedforward ANN (left) and different types of RNNs (Karpathy, Andrej, 2015)

By using sequential data, which contains additional information within the order and progress of the data, RNNs need far fewer examples to achieve good accuracy than standard ANNs. However, the processing of sequential data comes at an expense called the vanishing or exploding gradient problem. The main challenge of processing sequential information and learning patterns with backpropagation through time is the effect of recurrence on the weights w over many time steps. As described in chapter 4.2.3 the weights are manipulated to calculate a better output $\hat{y}$ with the given input x. In a RNN weights are multiplied many times with lessened or potentiated versions of itself. Hochreiter and Schmidhuber described the problem as follows: “*With conventional "Back-Propaga-*

tion Through Time" or "Real-Time Recurrent Learning", error signals "flowing backwards in time" tend to either (1) blow up or (2) vanish: the temporal evolution of the backpropagated error exponentially depends on the size of the weights. Case (1) may lead to oscillating weights, while in case (2) learning to bridge long time lags takes a prohibitive amount of time, or does not work at all." (Hochreiter and Schmidhuber, 1997, p.1). This, in turn, makes the optimisation of the cost function through approximation impossible. To overcome this problem different solutions have been proposed, but one of the most successful is the LSTM model, a special kind of RNN, designed in 1997 by Hochreiter and Schmidhuber (Hochreiter and Schmidhuber, 1997). LSTM networks were specially designed to learn long-term dependencies and avoid the vanishing gradient problem. The central idea is to have a recurrent edge within each cell that has a weight w = 1. This eliminates the problem of the vanishing gradient problem, as recurrent multiplication by 1 does neither diverge nor converge to zero. To still enable a LSTM to learn a process called *constant error carrousel* controls the cell state from one sequential step to the next without any weights being multiplied directly. Information flows on two levels from one step to the next while the update of the weights is controlled by so-called gates. Gates use different activation functions to forward or stop the information flow. "*Each memory cell's internal architecture guarantees constant error flow within its constant error carrousel CEC, provided that truncated backprop cuts off error flow trying to leak out of memory cells. This represents the basis for bridging very long time lags.*" (Hochreiter and Schmidhuber, 1997, p.23).

Summarising, the central change within a LSTM model in contrast to a RNN or feedforward model is a more complex type of hidden neuron. "*Instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but also has more parameters and a system of gating units that controls the flow of information.*" (Goodfellow et al., 2016, p.406).

Figure 11 shows a LSTM model with a LSTM cell. Identical to the previous figures each element represents a vector with multiple features which is put through the network. In this illustration three time steps and a dataset with three sequential data sample (x_{t-1},x_t,x_{t+1}) are visible. The central LSTM cell is shown in detail to reveal the processes within it. The network is only shown up to the output of the hidden layers (h_{t-1},h_t,h_{t+1}), the output $\hat{y}$ is not represented here.

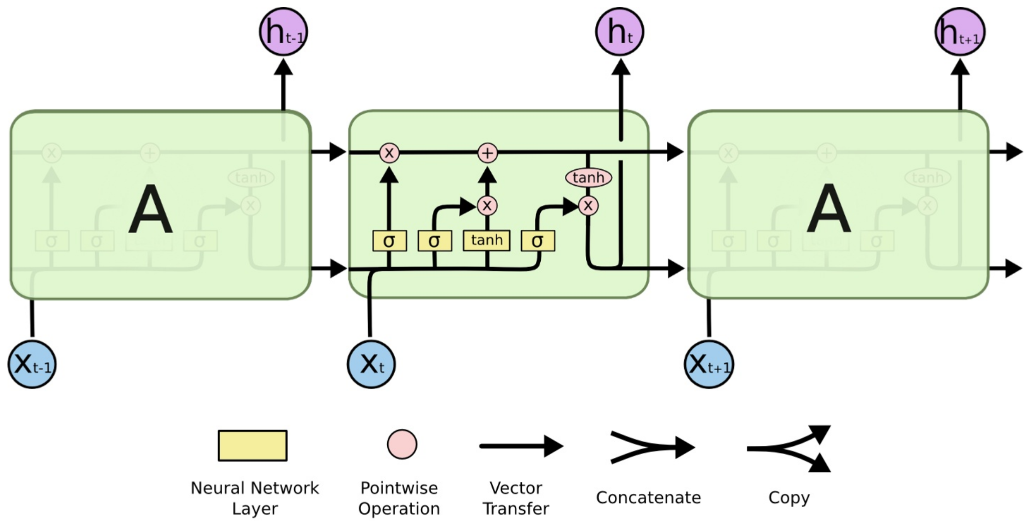
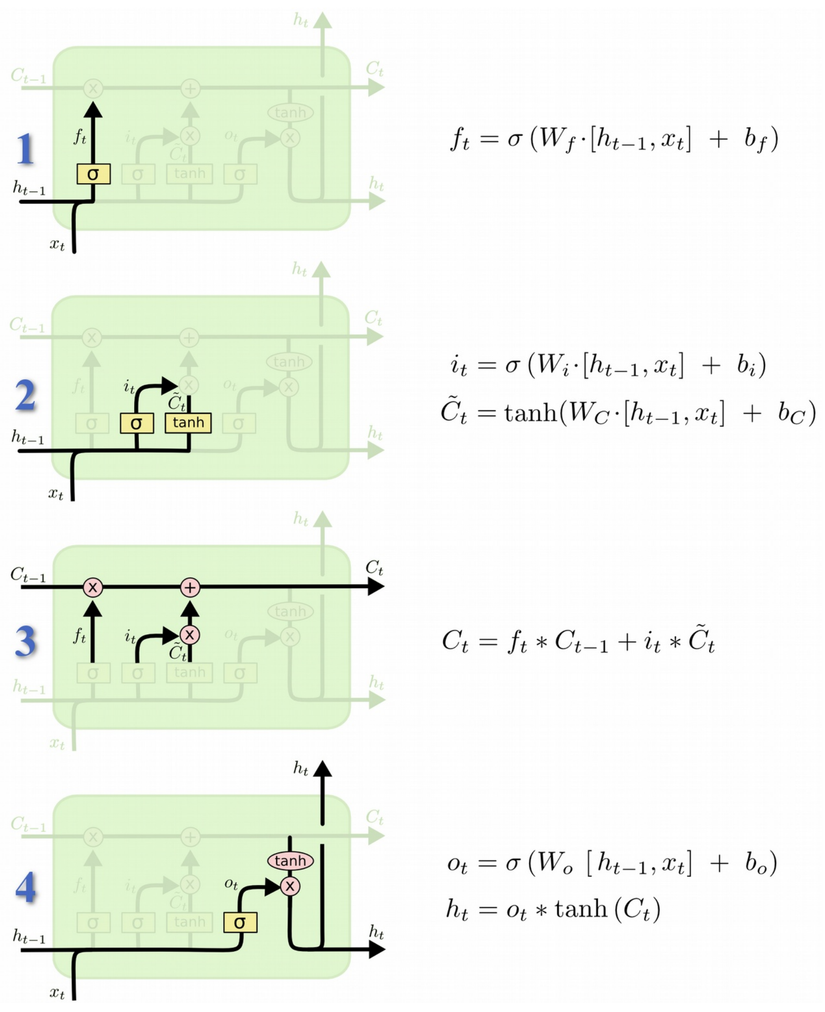


Figure 11: LSTM hidden neuron (Olah, 2015)

Information flows through an individual LSTM cell from the previous through the current to the next sequential step in two lanes. The top horizontal, straight arrow is the memory lane from C_{t-1} to C_t (cf. figure 12-3) and the more twisting is the hidden state from h_{t-1} to h_t (cf. figure 12-4). These two cell states compose the connection from current and previous sequential inputs and from the 'memory'. Both are connected and have their values changed at various points through gates. There are three gates that control the flow and change of information: forget (f_t), input (i_t) and output (o_t) gate. "*All the three gates combine the current input x_t with the hidden state h_{t-1} coming from the previous timestamp. The gates have also two important functions: 1) they regulate how much information has to be forgotten/remembered during the process and 2) they deal with the problem of vanishing/exploding gradients.*" (Ienco, D. et al., 2017, p. 1686)

Figure 12-1 shows the forget gate which decides how much of the past information should be kept, regarding the current input. To do this it combines the information of h_{t-1} and x_t in a sigmoid layer (σ). The output is of the interval [0,1], whereby 0 corresponds to all information is discarded and 1 to all information is kept. The inputs from h_{t-1} and x_t are further put through the input gate (cf. figure 12-2) and combined via a pointwise multiplication operation with the tanh layer $\tilde{C}_t$. This step regulates how much of the current input, regarding the past information, should be stored and forwarded to C. The memory lane (C_{t-1} to C_t) shows the transports of the main information from the past, through the current to the future time step, with only two minor interactions (cf. figure 12-3), controlled by the described gates. The forget gate uses a pointwise multiplication operation and the input gate addition or subtraction to change the value of the cell state C. Last, the output gate decides what output h_t the cell delivers to the next level and to the next sequential step. This output is calculated by a sigmoid layer which combines the current input x_t and the previous hidden state h_{t-1} with an added bias b_o. This value is combined via pointwise multiplication operation with the already by the forget and input gate manipulated memory lane C, which is run through a tanh layer (cf. figure 12-4). Finally, the memory C_t and the hidden state h_t are forwarded to the next step.



1
2
3
4
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$
$h_t = o_t * \tanh(C_t)$

Figure 12: Information flow through a LSTM Cell (own compilation of images from: Olah, 2015)

In the formulas in figure 12 the "*[...] different W∗∗ matrices and bias coefficients b∗ are the parameters learned during the training of the model.*" (Ienco, D. et al., 2017 p. 1686).
This chapter showed the concept of RNN and their ability to analyse sequential data. A special type of RNN, the LSTM network was presented as one solution to the vanishing gradient problem. Because of this, a LSTM network seems to be a good choice to extract the most information out of sequential satellite images: First, the direct information from the image's channels and secondly the temporal correlations within the sequential development of different objects on these images. Therefore, in part II a LSTM model will be developed to classify time series of Sentinel 2 images.

Part II: Development of the LSTM Model

In this part the development, training and evaluation of a LSTM is described, to classify Sentinel 2 time series images with the methods and tools presented in part I. The first chapter provides an overview of the whole concept and process, from data preprocessing to the LSTM architecture and its evaluation. The following chapters present detailed descriptions of the individual steps.

5. Overview

This thesis follows the classical steps to build a NN for a classification task which are (Raschka and Mirjalili, 2018):

1. Data preprocessing
2. Training of the model
3. Evaluation of the model with an independent dataset.
4. Implementation of the model on new data

In this thesis steps one to three are described. Step 4 would be the actual implementation within a project to classify SITS.

The individual steps have some iterations between them, especially during the training and validation of the model. Regarding the task at hand, the classification of SITS, this thesis uses the workflow displayed in figure 13. The three coloured boxes (blue, red and green) frame the three major steps, i.e. data preprocessing, development and training of the LSTM and evaluation of the model. An overview of these three steps is given below in this chapter.

The following chapters 6, 7 and 8 will describe these three steps in detail focussing on the general data- and workflow. All processes described there were implemented with Python. An overview of the developed Python methods and the structure of the files is given in "Annex I: Overview of the developed python scripts and their structure". In addition to this, the actual Python files include detailed comments on the code. Some general data manipulation processes, such as the manipulation of pandas data frames or the merging of csv files are not described here as they are not relevant for this thesis.

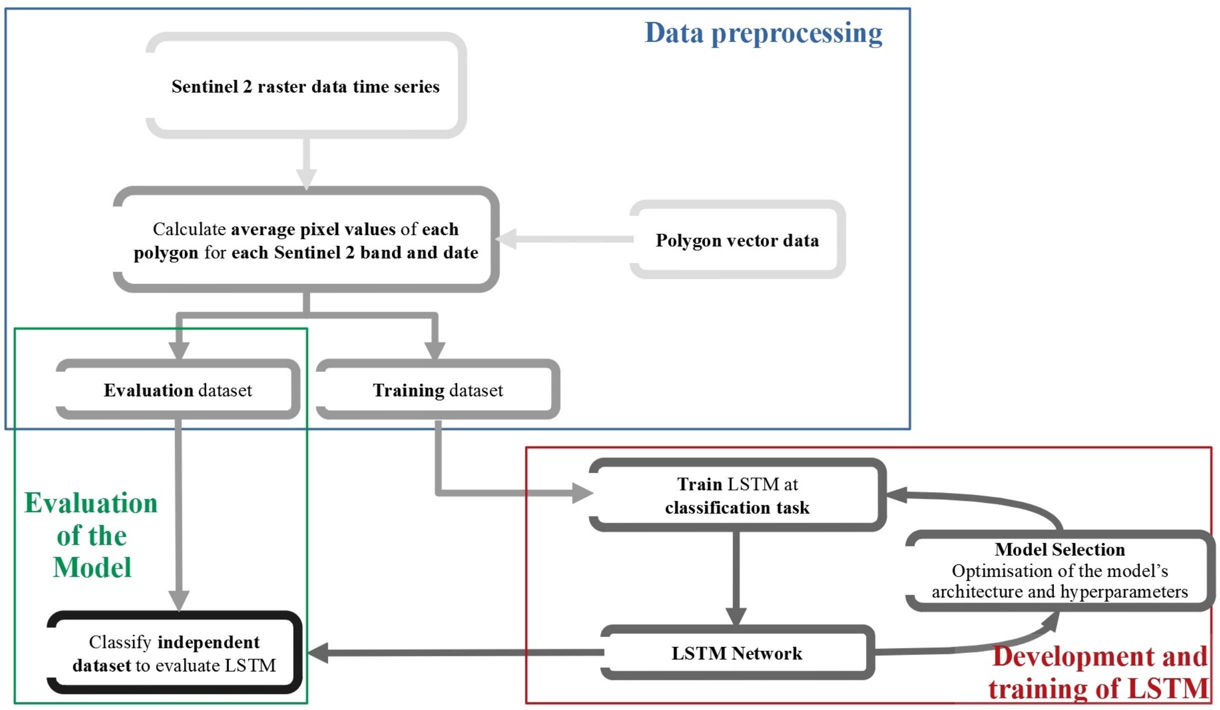

Data preprocessing
Sentinel 2 raster data time series
Calculate average pixel values of each polygon for each Sentinel 2 band and date
Polygon vector data
Evaluation dataset
Training dataset
Evaluation of the Model
Train LSTM at classification task
Model Selection Optimisation of the model's architecture and hyperparameters
LSTM Network
Classify independent dataset to evaluate LSTM
Development and training of LSTM

Figure 13: Workflow to classify SITS with a LSTM (source: own figure)

1. Data preprocessing

The vector data, containing the fields borders and cultivated crops, has to be separated in fitting and non-fitting data. Unusable are data samples that can't be used for the training of the NN due to the low information level of these samples, e.g. due to small field size, low number of cases or certain CAP Rules. Some classes are also summarised because their difference is more due to the system of the CAP rules than any real visible difference.

Based on the extent of the vector data the Sentinel images are downloaded and all images of the same date are combined in a virtual layer. Then, the average value for each vector polygon for each available day and each image channel is calculated. This data is put into a 3d vector with the three corresponding axes: field number, time step and Sentinel 2 channel. The data is split into a training and evaluation dataset. The training data is further split via a k-fold method into 10 different sets of training and validation samples which are used to train the NN. The evaluation dataset is used to evaluate the NN prediction abilities on new unseen data. Due to hardware limitations a function was developed to remove unnecessary, interfering and redundant time samples to speed up the training process considerably. Then the dataset is interpolated and at last normalized. After these steps, the data is ready to be put into the LSTM network. The data preprocessing is described in-depth in chapter 6.

2. <u>Development and training of the LSTM</u>

The LSTM is built using Python and within that programming language the library: Keras (cf. chapter 3.2). This model can then be trained and evaluated based on its accuracy, i.e. the percentage where the prediction fits the actual labels. The LSTM model is created within a python function, so that hyperparameters, such as the neurons, dropout rate etc., can easily be changed by calling the function with different inputs. The optimal hyperparameters can be found by systematically adjusting the hyperparameters and comparing the accuracy of the derived models. This process, called model selection, is very time consuming because each model has to be trained for many epochs to obtain reliable information of its accuracy. The development and training of the LSTM is described in detail in chapter 7.

3. <u>Evaluation of the Model</u>

The data is split into a training and an evaluation dataset before the training of the LSTM network. The evaluation dataset is never used to train or validate the NN. It is thus an independent sample. This sample is used to finally test the accuracy of the network and its classification performance across different classes. Different metrics (e.g. accuracy, precision, recall, AUC-Score) are calculated to evaluate the model and allow quantitative comparison to other models. Apart from these metrics other analysis methods of the predicted results, e.g. via a confusion matrix, help to identify the performance of individual classes. The evaluation of the developed LSTM model is described in detail in chapter 8.

6. Data preprocessing with python

„*Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm. Thus, the preprocessing of the data is one of the most crucial steps in any machine learning application.*"(Raschka and Mirjalili, 2018, p.12) For this reason, this chapter takes a detailed look at the three different data preprocessing steps:

1. The first part shows how the vector data, i.e. the data containing the field polygons, is separated into fitting and non-fitting data and technically prepared.
2. The second part demonstrates how the Sentinel 2 raster images are prepared to extract the values of the individual channels for each day and field.
3. The third part displays the extraction of the numerical pixel values from the Sentinel 2 images, the creation of a 3d array and the reduction, interpolation and normalisation of its values.

6.1. Data preprocessing of the vector data

The original vector dataset provided by the "Thüringer Landesverwaltungsamt" (engl: Ministry of Administration) contained roughly 20 000 polygons (cf. chapter 2.2). These were filtered in fitting and non-fitting samples based on different criteria. Non-fitting samples were removed from the dataset. The following list provides an overview of the criteria which are described in detail thereafter:

1. Plants types: Since not all polygons contain crops, those containing non-crop like plants are obviously non-fitting to train a NN to detect crops.
2. Field size: A minimum size of each field is required to avoid mixed and non-pure pixels which would corrupt the training data.
3. Frequency of crop type: A minimum number of training samples are required to enable a NN to learn.
4. Summarising and removing of certain crop types: The data is received from the CAP campaign 2018 in Thuringia. The labelling of the data was therefore subject to certain CAP rules which is not or hardly visible in the real world even when standing next to the field, e.g. dif-

ference between different form of meadows, pastures and grassland. For this reason, some classes had to be excluded while others were summarised into a single class.

The first criterion is the plants associated with the polygon. As the network should be trained on crops all polygons containing no or other plants are removed. A large number of polygons consists of structural important ecological elements (in german: “Landschaftselemente” and “Ökologische Vorrangflächen”). These elements contain no crops but hedges, small woods, tree lines, small ponds, grass and flowering field borders and others. In order to receive the CAP subsidies, farmers are required to keep some of these elements for nature protection reasons. Another group of polygons without crops are areas that are temporarily removed from the subsidies program, e.g. due to infrastructure projects in that area. After removing all the polygons not containing any crops, there were roughly 13 200 polygons left.

The second criterion sets a lower threshold for the size of the fields. As Sentinel 2 images have a minimum resolution of 10m and 20m, depending on the band, a minimum field size is required to recognise the crop type definitely. If the fields are too small their pixel values might be blended with surrounding objects, e.g. roads, hedges, trees, etc. This would have significant negative impacts on the training because it increases the difficulty to clearly define a crop class. The following steps were taken to prevent this negative impact:

a) All polygons with an area of smaller than 500m^2 were deleted, i.e. polygons consisting of 5 or less Sentinel 2 image pixels. So few pixels could ruin the training data, because only the average pixel values of each Sentinel 2 band and field are used.

b) A negative buffer of 5m was applied on all fields so that the polygons do not contain pixels which could be blended with other objects at the border of the fields.

This second step removed ca. 500 polygons, so that 12 700 polygons are left in the dataset.

The third criterion focusses on the class frequency. All crop classes with a frequency below 30 data samples were removed. NNs need a certain minimum sample size to be able to detect a pattern so that they can recognize features in unknown data and classify it (cf. chapter 4). After some initial tests a frequency of 30 was selected. This frequency is still very low and bares a high risk to produce classes with a bad detection rate. However, this low number was selected on purpose to test the lower limit of inputs needed for the LSTM. This step removed 329 polygons, so 12 371 fields are left in the total dataset.

As a fourth step individual classes were evaluated based on their labelling and their real world features. As described above, the polygons are based on CAP subsidies applications and rules. These rules define some crop types that are differentiated less by their cultivation and visible features and more by their economic usage. The following table presents an overview of the classes that are summarised for this reason.

Summarised Class	Class I	Class II	Class III	Class IV
Maize	Corn-Cob-Mix Maize	Maize	Maize undersown with grass	
Clover - Grass - Bur clover	Clover-Grass	Clover – Bur clover	Grass – Bur Clover	
Grassland	Meadows (german: "Wiese")	Mow pastures (german: "Mähweiden")	Pastures (german: "Weiden")	Orchard meadow (german: "Streuobstwiese")

Table 2: Summarised crop classes (own table)

As a last step three classes were completely removed because of the unique CAP rules defining these classes not by the actual crop type, but by the way the field is managed. These classes have such unique features that it proved impossible to classify them with reasonable accuracy in the context of this thesis. These three classes are:

1. Field grass (German: "Ackergrass")
2. Field removed from production temporarily (German: "Ackerland aus der Erzeugung genommen")
3. Field removed from production permanently (German: "stillgelegte Ackerflächen")

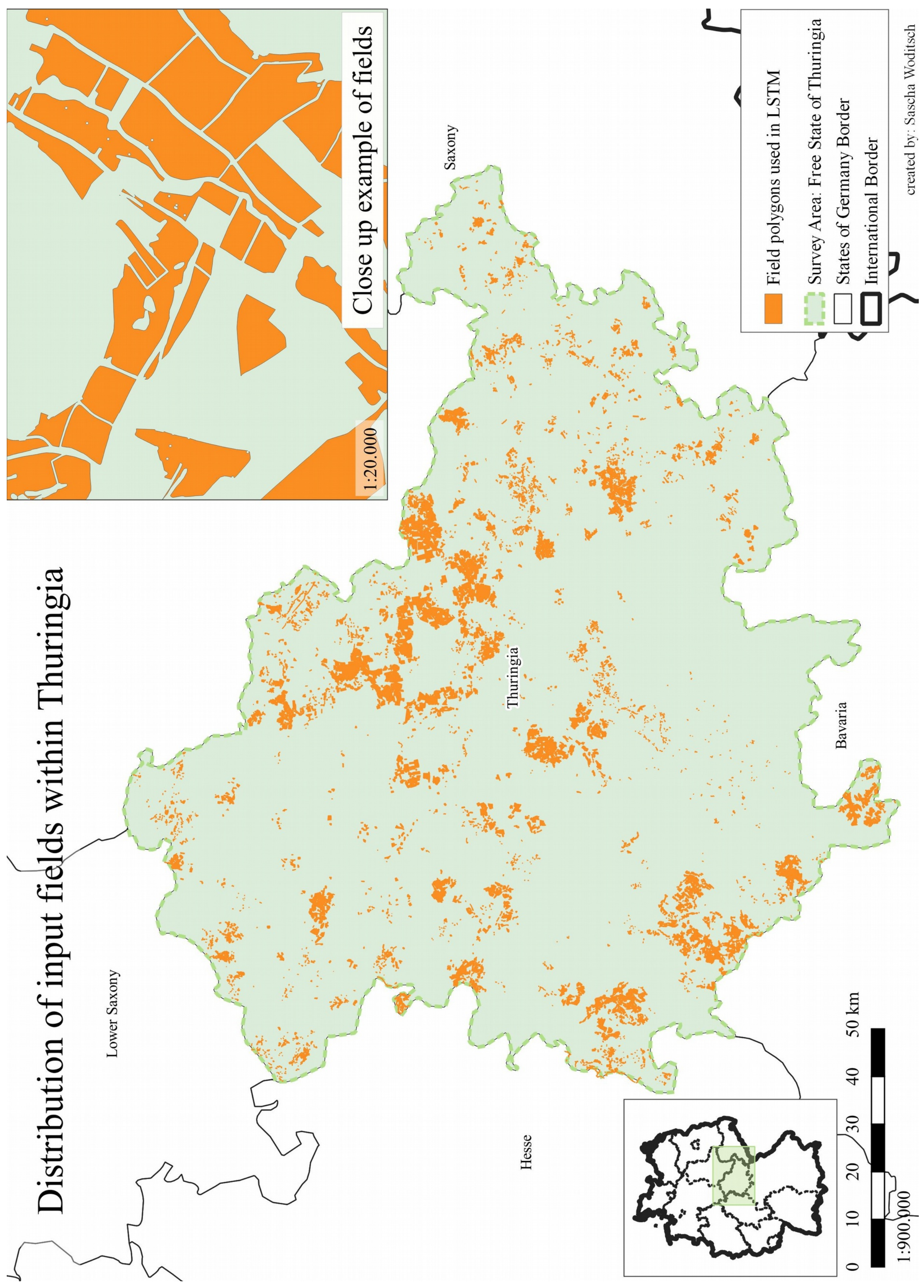
After initial tests the LSTMs accuracy results for all three classes, listed above, was always very low. The fields removed from production can be in very different states depending on how long they have been out of production. This makes them hard to differentiate from certain grassland types. The same applies to the field grass class. Field grass is a specially managed type of grassland that has to be ploughed at least once every 5 years, but it might be ploughed more often. This is in contrast to other types of grassland which can not be ploughed ever, according to CAP rules. Thus, a parcel containing field grass that grew already 3-5 years is practically identical, regarding its visible features, to other grasslands. In tests the LSTM Network recognized only 17% of the class field grass correctly while 73% were misclassified as grasslands. The classification heavily depends on the field management practice in 2018, i.e. if the field was ploughed in the season 2018 or not.

Thus, field grass cannot be added to the grassland class and it cannot be used as a class of its own. For this reason, the class was removed completely. It still would be possible to detect these three classes defined by management practices that span over several years by simply using input data that also covers several years. Sentinel 2 scenes covering multiple years would make it possible to detect management practice (e.g. ploughing) from previous years and thus classify the fields accordingly. However, this is beyond the scope of this thesis which focusses on crop detection with the data from one year. Therefore these three classes were removed from the dataset. This step removed 684 polygons, so 11 687 fields were left the dataset.

In total from 13 200 fields containing actual crops, 500 have been removed due to size, 329 due to low frequency and 684 due to special CAP rules. The final dataset contains 11 687 field polygons with 16 different crop types. Table 3 lists all of them and their respective frequency. The following map (cf. figure 14) displays the distribution of the fields in Thuringia.

Crop Type	Number of Samples
Grassland	6 263
Winter Wheat	1 815
Rape	987
Winter Barley	561
Maize	520
Summer Barley	474
Grass - Clover - Bur clover	355
Summer Wheat	137
Triticale	114
Winter Rye	114
Pea	84
Summer Oats	80
Sugar Beet	71
Other Seeds and Herbs	43
Field Bean	37
Potato	32
Total	**11 687**

Table 3: Crop types and number of samples (own table)



Distribution of input fields within Thuringia
Close up example of fields
1:20.000
Saxony
Thuringia
Bavaria
Lower Saxony
Hesse
Field polygons used in LSTM
Survey Area: Free State of Thuringia
States of Germany Border
International Border
created by: Sascha Woditsch
0 10 20 30 40 50 km
1:900.000

Figure 14: Map of used fields within the Free State of Thuringia (own figure).

6.2. Data preprocessing of the raster data

The information to classify the fields is extracted from the pixel values of the raster data. This data consists of Sentinel 2 scenes which were downloaded from ESAs “Copernicus Open Access Hub” (“Copernicus-Open-Access-Hub”). The images were preprocessed by EFTAS Fernerkundung Technologietransfer GmbH with the Sen2Cor software so that they are of Level-2A product quality. Sentinel Level-2A images are preprocessed to enhance image quality and accuracy of the pixel values. “*The Level-2A prototype product is an orthorectified product providing Bottom-Of-Atmosphere (BOA) reflectances, and basic pixel classification (including classes for different types of cloud)*” (ESA, 2015, p.60).

Table 4 displays the Sentinel 2 channels. The average pixel values were calculated for the green marked channels.

Band name	Resolution (m)	Central wavelength (nm)	Band width (nm)	Purpose
B01	60	443	20	Aerosol detection
B02	10	490	65	Blue
B03	10	560	35	Green
B04	10	665	30	Red
B05	20	705	15	Vegetation classification
B06	20	740	15	Vegetation classification
B07	20	783	20	Vegetation classification
B08	10	842	115	Near infrared
B08A	20	865	20	Vegetation classification
B09	60	945	20	Water vapour
B10	60	1 375	30	Cirrus
B11	20	1 610	90	Snow / ice / cloud discrimination
B12	20	2 190	180	Snow / ice / cloud discrimination

Table 4: Sentinel 2 Channels (own table with information from ESA, 2015, p.51-54 and “GDAL-S2”)

The channels B01, B09 and B10 were not directly used because:

1. They have very low resolution of 60m and thus a high blur radius, i.e. pixels of smaller fields or any border pixels will contain impurities from other neighbouring crops and/or other objects.

2. They are mostly used for atmospheric image correction (ESA, 2015). Since the used Sentinel 2 data is already of Level-2A the atmospheric correction is already complete.

Apart from the directly recorded Sentinel 2 bands, the following computed bands, which are provided for all Level-2A scenes, were used.

Computed Band Name	**Description**
Aerosol Optical Thickness (AOT)	AOT map at 550nm
Visibility Index (VIS)	Visibility Index corresponding to the AOT
Water Vapour (WVP)	Scene-average Water Vapour map
Scene Classification (SCL)	Provides a basic pixel classification map of 12 classes, e.g. cloud, cloud shadows, vegetation, water, snow, etc.

Table 5: Level-2A computed bands (own table with information from ESA,2015 p.48-49, Mueller-Wilm, 2018, p.22-25 and S2-L2A Overview)

In order to analyse the complete agricultural season 2018 a total of 143 images taken between 01.03.2018 – 31.10.2018 have been preprocessed. The images are provided in tiles or granules with a size of 100*100km^2 (“S2-oUG – Definitions”). These granules cover the survey area more or less depending on the path of the satellite and the swath of the taken images (cf. figure 15). The number of tiles overlapping with the survey area at a given date varies greatly from 1 to 11 tiles. For each of these tiles an individual JPEG-2000 file exists for each channel and computed band. For this thesis, 10 channels (cf. table 4) and 4 computed bands (cf. table 5) adding up to 14 individual layers for each tile and date were used. Table 6 displays the total number of image tiles, the respective number of days with an image and the number of individual image files. For example: The survey area was covered by 3 tiles on 48 days, i.e. 3 tiles multiplied by 48 days and 14 channels results in 2016 image files. All in all 10 794 JPEG2000 files had to be processed.

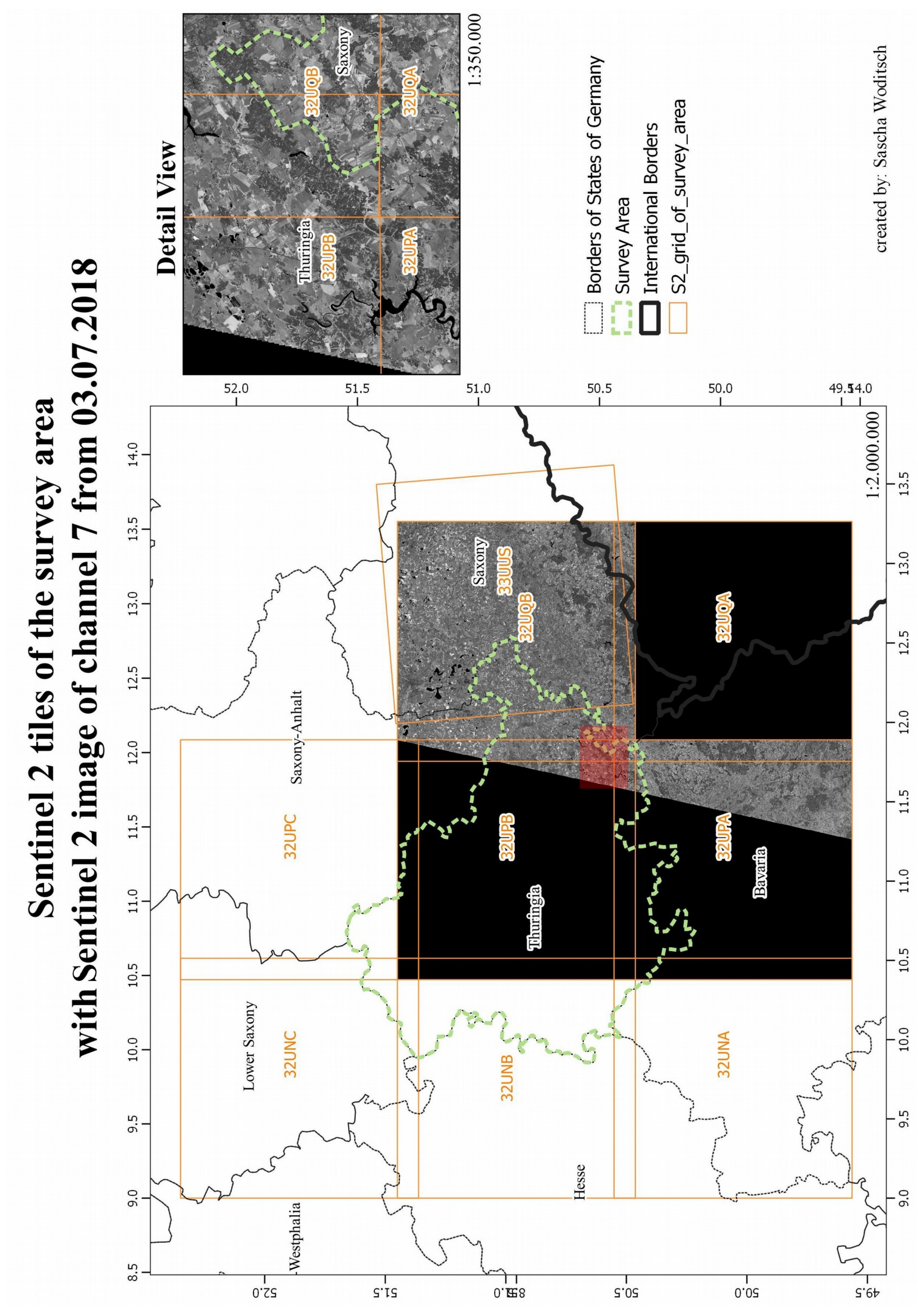
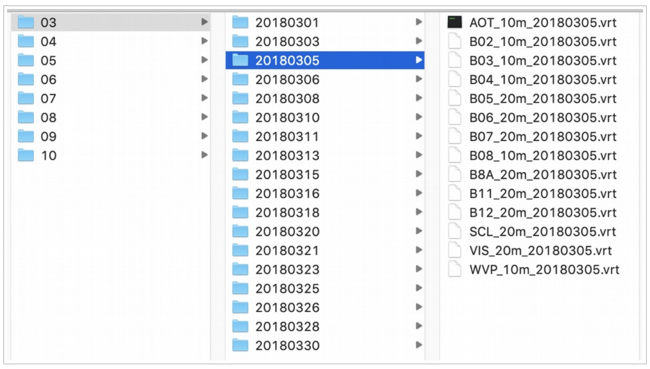


Figure 15: Map of Sentinel 2 tiles over the survey area (own figure)

Sentinel 2A and 2B granules overlapping with survey area	Available dates of Sentinel 2 scenes of survey area	Number of individual image files (of 14 channels and bands)
3	48	2 016
6	46	3 864
7	46	4 508
9	2	252
11	1	154
Total:	**143**	**10 794**

Table 6: Overview of Sentinel 2 scenes, tiles and files (own table)

Figure 15 shows the Sentinel 2 tiles touching the survey area and one example scene from the 03.07.2018 displaying channel B07 consisting of 4 tiles that overlap partially with the survey area. The amount of image files covering different parts of the survey area and spanning different dates makes it obvious that the first challenge was to organise these images temporally and geographically in a way that the sequential data needed for the LSTM Network could be extracted.



```
03    20180301    AOT_10m_20180305.vrt
04    20180303    B02_10m_20180305.vrt
05    20180305    B03_10m_20180305.vrt
06    20180306    B04_10m_20180305.vrt
07    20180308    B05_20m_20180305.vrt
08    20180310    B06_20m_20180305.vrt
09    20180311    B07_20m_20180305.vrt
10    20180313    B08_10m_20180305.vrt
      20180315    B8A_20m_20180305.vrt
      20180316    B11_20m_20180305.vrt
      20180318    B12_20m_20180305.vrt
      20180320    SCL_20m_20180305.vrt
      20180321    VIS_20m_20180305.vrt
      20180323    WVP_10m_20180305.vrt
      20180325
      20180326
      20180328
      20180330
```

Figure 16: VRT File Structure (own figure)

To achieve this a script was developed to produce GDAL virtual rasters combining all images of one channel and one date. Due to the strict naming convention of the Sentinel 2 SAFE file format the images date and tile number can be easily identified ("S2-oUG - Product Naming Convention"). The script "BuildVRT.py" (cf. Annex I) uses the folder names to identify all unique dates. It then searches within the S2 SAFE format for the *.jp2 – files of each Sentinel 2 band and combines them based on their unique date and the unique date list. The result is a python dictionary which stores the Sentinel 2 band name and unique date as key and all corresponding tiles within a list as values. Based on this dictionary multiple virtual rasters are built via the "gdal.BuildVRT()" method (cf. "GDAL – API"). These virtual rasters combine the selected raster tiles only via a link and therefore have a very small file size, but can be queried like a normal raster file. The result is written in a folder based on month and dates (cf. figure 16). The final results are 2 002 virtual rasters, i.e. 14 channels multiplied by 143 unique dates of the survey area.

6.3. Data preprocessing of the numerical data

The numerical data is the actual data that is put into the LSTM. This part of the data preprocessing includes multiple steps. The milestones of this process are:

1. Creation of a numerical table from the input vector and raster data and adding the date
2. Splitting of the data in a training, validation and evaluation sample
3. Reshaping of 2d table into a 3d array
4. Reduction of low information data within the time dimension to increase the performance of the model
5. Interpolation of missing data
6. Normalization of the data

6.3.1. Creation of a numerical table from the input vector and raster data

The numerical data is created by extracting the average pixel values of each field for each date and channel raster. The mean pixel value of one field for each channel and date was calculated in QGIS using the QgsZonalStatistics method ("GDAL–API"). A small script "Calc_Zonal_Statistics.py" (cf. Annex I) was used to automate this process. The result of this process is a large table with 11.687 rows and 2.004 columns. Each row holds one field. The first column holds a unique field id, the second the crop type. The following 2.002 columns hold the extracted mean values for that field, for each date and channel.

As Rußwurm and Körner proposed in their paper "*Multi-temporal Land Cover Classification with Long Short-term Memory Neural Networks*" (Rußwurm and Körner, 2017, p.554) the day the Sentinel image was taken should be included in the LSTM input. This way the time information can be connected not only sequentially, like before or after, but located exactly on a timeline in connection to all other inputs. This is especially important as the Sentinel scenes are not taken regularly and some time steps of each field will be removed due to cloud coverage and to enhance performance. To insert the dates in a way the LSTM network can use them the time delta between the Sentinel 2 observation date and the 01. January 2018 was calculated. This was done within Python directly, using the python datetime module ("Python – Datetime"). This resulted in a new column for each observation date containing the day of the observation (counted from the 01.January 2018). This number was normalised by dividing it through 365, for a linear representation of the date between 0 and

1. A circular date representation would be required, if dates from all over the year would be used and thus the 31. December and 01. January should be very close. To keep the calculation and the 3d array more simple and because only parts of the year, i.e. one growing season from March to October is observed, time was calculated linear and not in a circular way. To add the day of observation to the data 143 new columns, one for each observation date were inserted to the data table. The data table has now 11 687 rows and 2 147 columns.

6.3.2. <u>Splitting of the data</u>

The NN will use the training and validation sample multiple times to test and find the optimal set of hyperparameters. During this process, called model selection, the hyperparameters are changed and adapted to get the best results. To still ensure the independence of the hyperparameters from the dataset, an evaluation dataset was split from the data before any training. Because "*[...] if we reuse the same test dataset over and over again during model selection, it will become part of our training data and thus the model will be more likely to overfit.*" (Raschka and Mirjalili, 2018, p.190). Therefore, a random stratified evaluation or test sample of 15% of the data was removed from the main dataset and saved to a separate csv file. This creates a large enough test dataset, while still leaving enough samples to train the NN. Furthermore, this split ratio is within the range mostly used, as Kamilaris and Prenafeta-Boldú point out: *"Most of the studies divided their dataset between training and testing/verification data using a ratio of 80–20 or 90–10 respectively." (Kamilaris and Prenafeta-Boldú, 2018, p. 74).* The total dataset is thus now split in an 85% training and validation set and a 15% evaluation or test dataset. Table 7 displays the exact numbers of samples in each dataset, while figure 17 provides a graphical overview of the data split and usage procedures within the training process.

CropType	**kFold training and validation sample**		**Evaluation sample**	
	Number of Samples	**in %**	**Number of Samples**	**in %**
Grassland	5 323	53.59	940	53.59
Winter Wheat	1 543	15.53	272	15.51
Rape	839	8.45	148	8.44
Winter Barley	477	4.8	84	4.79
Maize	442	4.45	78	4.45
Summer Barley	403	4.06	71	4.05
Grass - Clover - Bur clover	302	3.04	53	3.02
Summer Wheat	116	1.17	21	1.2
Winter Rye	97	0.98	17	0.97
Triticale	97	0.98	17	0.97
Pea	71	0.71	13	0.74
Summer Oats	68	0.68	12	0.68
Sugar Beet	60	0.6	11	0.63
Other Seeds and Herbs	37	0.37	6	0.34
Field Bean	31	0.31	6	0.34
Potato	27	0.27	5	0.29
Total	**9 933**	**84.99**	**1 754**	**15.01**

Table 7: Number of samples in training-validation and evaluation data set (own table)

The training and validation dataset (the 85% of the total sample) is further split into different parts used for training and validation. Goodfellow et al. point out that a random split in training and validation dataset can be problematic, as it implies a statistical uncertainty around the average test error (Goodfellow et al. 2016). One solution to reduce the statistical uncertainty is to use a cross validation procedure. *"The most common of these is the k-fold cross-validation procedure [...], in which a partition of the dataset is formed by splitting it into k nonoverlapping subsets. The test error may then be estimated by taking the average test error across k trials. On trial i, the i-th subset of the data is used as the test set, and the rest of the data is used as the training set.*"(Goodfellow et al., 2016, p.120). Figure 18 shows the procedure of a k-fold cross validation. Whereby k represents the number of folds the data is split into. *"A good standard value for k in k-fold cross-validation is 10, as empirical evidence shows."* (Raschka and Mirjalili, 2018, p.192). This thesis follows this recommendation and uses a 10 fold cross validation. Since the frequency of the different classes are very uneven, the largest class covers around 53% of the sample (cf. table 7), the split into folds has to be stratified to make sure all classes are represented in each training and validation fold.

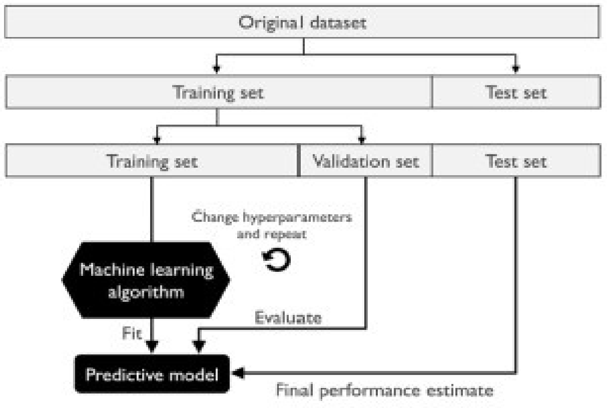


Figure 17: Data split procedure (Raschka and Mirjalili, 2018, p.191)

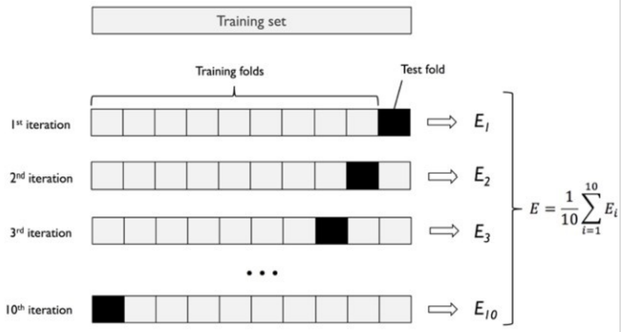


Figure 18: K-fold cross validation (Raschka and Mirjalili, 2018, p.192)

To generate 10 folds, the "StratifiedKFold" method from the sklearn library was used ("SCIKIT-LEARN-API-StratifiedKFold"*).* After the total x and y data was loaded from a csv file into memory, the following Python code was executed to produce stratified folds, separated in 10 folds of x and y data and separated each in a training and validation dataset. The stratification is done based on the input of "dataY" in the ".split()" method.

```
k = 10 # define number of splits
skf = StratifiedKFold(n_splits=k, shuffle=True) # define stratified kfold
kNr = 0
# split dataset in k train and validation sets stratified by values of y
for train_index, valid_index in skf.split(dataX, dataY):
   kNr += 1
   # create X and y train and test samples based on current k fold
   xTrain, xValid = dataX[train_index], dataX[valid_index]
   yTrain, yValid = dataY[train_index], dataY[valid_index]
```

After this code segment, still within the “for loop”, the output variables: “xTrain”, “xValid”, “yTrain” and “yValid” are saved to individual csv files. The following figures provide an overview of the final data split. The data manipulations described in the following chapters are calculated for each fold of the training-validation dataset right before the start of the training and for the evaluation dataset to predict its inputs. Figure 19 shows the frequency of the individual classes in the training, evaluation and test sample. Figure 20 displays the random stratified distribution of the training and validation data in each fold for each class. The light grey area represents data used as training input, whereas the dark grey areas represent data used as validation data during the training process. The colours of the class labelling of the bottom bar are the same as in figure 19.

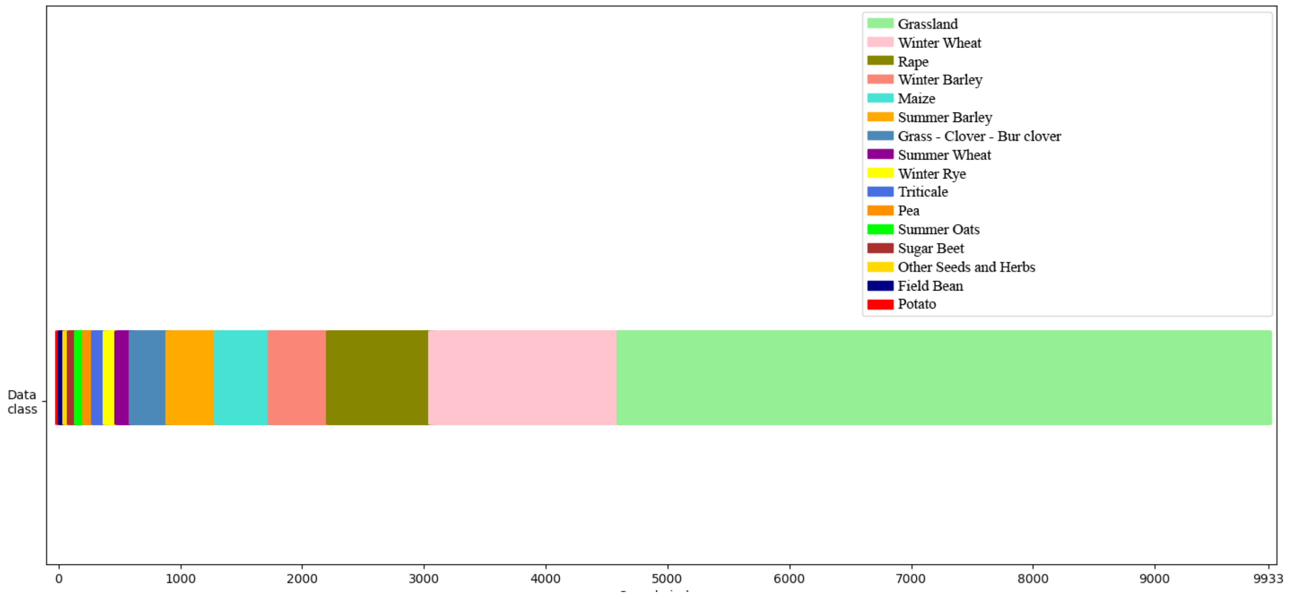


Figure 19: Frequency of data class in total sample (own figure)

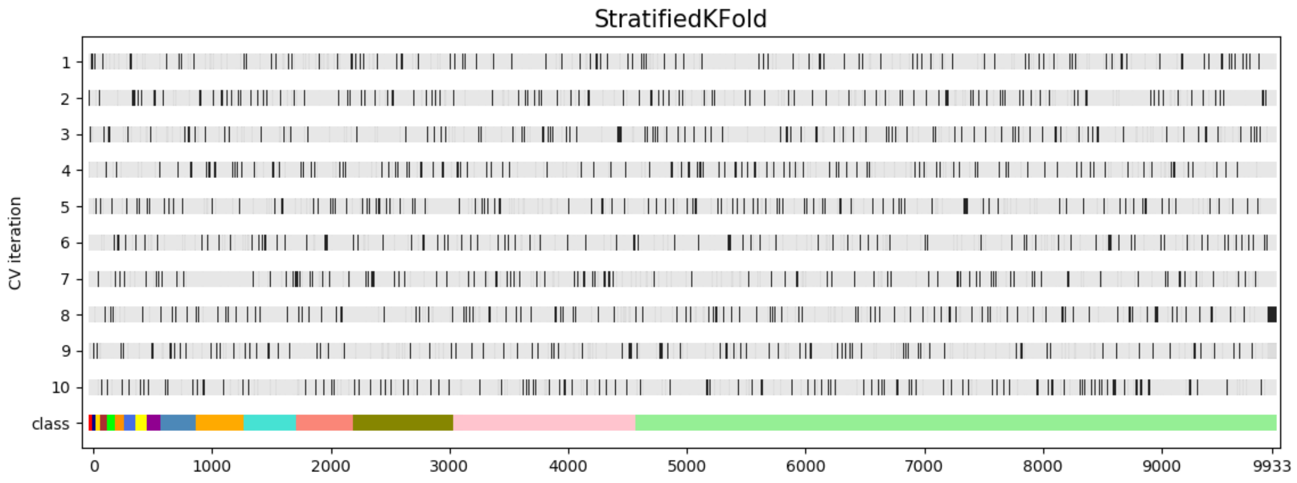
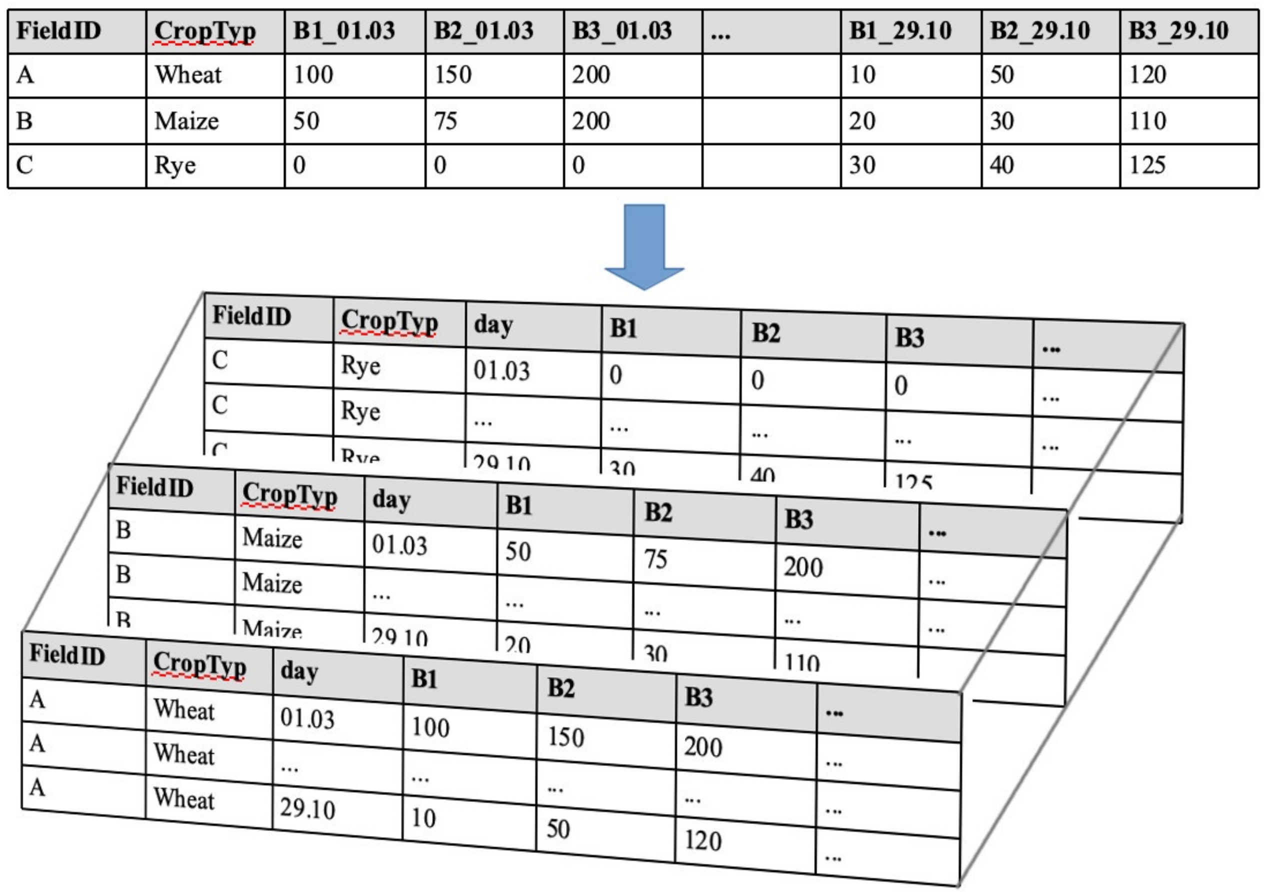


Figure 20: Stratified k-fold cross validation: Distribution of training and validation data within each fold (own figure)

6.3.3. <u>Reshaping of 2d table into a 3d array</u>

So far the input x datasets containing the mean pixel values are large 2-dimensional tables. To "activate" the time-dimension within the data, it is necessary to reshape it into a 3 dimensional array. This 3d – array has the required sequential data format to be used as LSTM input. The transformation from 2d to 3d array is done within the method "shape_data_to_3d_array()" (cf. Annex I). The method transforms the dataset into a 3d array. The dataset columns containing the pixel values are extracted and then reshaped via the "numpy.reshape()" function ("NP-API-Res"). The data is restructured in a way that each field polygon is presented in its own 2d table. The columns display the 14 Sentinel band and the column for the observation day. The rows of that table display the 143 different time steps for each field. A step into the third dimension shows the average pixel values of the next field polygon. Figure 21 shows this process exemplary for three fields with random values. In contrast, to figure 21 the actual 3d array contains only the dates and pixel values and not the field id or crop type. The field id is preserved by the order of the array and is used to connect the input x data to the y data, i.e. the crop type solutions. Since, the third dimension of the 3d array created here has the same order as the list of extracted crop types, i.e. yTrain (cf. chapter 6.3.2), both arrays can be connected. This way the NN knows the y label for each input x.



FieldID	CropTyp	B1_01.03	B2_01.03	B3_01.03	...	B1_29.10	B2_29.10	B3_29.10
A	Wheat	100	150	200		10	50	120
B	Maize	50	75	200		20	30	110
C	Rye	0	0	0		30	40	125

FieldID	CropTyp	day	B1	B2	B3	...
C	Rye	01.03	0	0	0	...
C	Rye	...	...	...	...	...
C	Rye	29.10	30	40	125	

FieldID	CropTyp	day	B1	B2	B3	...
B	Maize	01.03	50	75	200	...
B	Maize	...	...	...	...	...
B	Maize	29.10	20	30	110	

FieldID	CropTyp	day	B1	B2	B3	...
A	Wheat	01.03	100	150	200	...
A	Wheat	...	...	...	...	...
A	Wheat	29.10	10	50	120	...

Figure 21: Transformation of the data from 2d to 3d array (own figure)

6.3.4. <u>Reduction of low information data within the time dimension</u>

So far the data still consist of 143 individual time steps. Within these 143 time steps are multiple fields that contain no or only bad data values. No data pixel values contain 0 or “NaN” (“Not a Number”), while bad data values are altered by cloud coverage.

No data values exist because the sequential input data has to be of equal length for all samples. The mean pixel values of all field polygons were extracted from all 143 observation dates for each of the 14 Sentinel 2 channels (cf. chapter 6.2). Since the Sentinel 2 tiles rarely cover all of the survey area in one day there are two cases which produce no data values (cf. figure 15 and table 6),:

1. A polygon falls outside all available tiles, i.e. the mean value returned by the QgsZonalStatistics method for all channels of that observation day is “NaN”.
2. A polygon falls on the black border pixels of a tile, i.e. the mean value returned by the QgsZonalStatistics method for all channels of that day is “0”.

Bad data values exist due to cloud coverage that blocks the satellite sensor from recording the actual ground reflection, so it records the pixel values of the clouds instead. These values could mostly be filtered by preselecting Sentinel 2 scenes with general low cloud coverage. The disadvantage of this method would be that the whole image is discarded, although there are some areas without cloud coverage. Furthermore, there might be cloud coverage over certain areas even on images with low cloud coverage. Therefore, all images regardless of cloud coverage are used and the actual cloud coverage is determined for each individuals field plot within a certain time frame. This has the advantage of reducing the input size substantially while using the optimal pixel values for each polygon individually within the set time window.

The exact crop grow status (and thus the pixel reflection value) differs slightly from field to field for a given point in time due to different local factors, e.g. different date of sowing, status of soil, rainfall in the local area etc. Furthermore, the growth status of the plants and therefore the actual pixel values change only very little within a small time period of a few days. In conclusion within a certain time period of a few days, pixel values from the same crop type will produce the same or a very similar pixel values. Therefore, by selecting only the best pixel values within a narrow time window for each field, results in no information loss, but rather an increase in information quality.

This approach is implemented by filtering the whole dataset based on the values of the SCL and AOT band. The SCL band contains already a rudimentary classification of 11 classes ("S2-L2A Overview") which can be used to track the most promising pixel values. However, the SCL classes are no longer clearly defined to one SCL class because the SCL values were averaged for each field. To deal with this problem the filters do not search for the exact SCL classes, but for values ranging close to those classes.

The developed "filterCloudFree()" function (cf. Annex I) searches for the optimal pixel values within a time window of four consecutive Sentinel 2 scenes for each field individually. Different time periods were tested (cf. chapter 7) and a time window of four consecutive Sentinel 2 images yielded the best results, regarding speed and accuracy of the NN. Within the input x 3d array, the method searches, via the "numpy.where()" method ("NP-API-where"), for the observation row with the best chance of actually containing true ground pixel values. The rows are filtered based on the following hierarchical filters, whereas values returned from a higher filter are preferred to a lower filter.

1. Return row with valid SCL value or, in other words, return all rows where the SCL column is not ‘NaN’ or not ‘0’.
2. Return row with the lowest AOT value, as a lower AOT value means less atmospheric disturbance within the image.
3. Return values with an SCL Value close to 4 ($3 < x < 4.5$). A SCL value of 4 represents a vegetation area and thus indicates that the pixel values used to identify the crop type are truly ground pixel reflections.
4. Return values with an SCL Value close to 5 ($4.5 <= x < 6$). A SCL value of 5 represents a bare soil area and thus indicates that the pixel values used to identify the crop are truly ground pixel reflections. Furthermore, bare soil is returned in the highest (i.e. most important) filter because a change to bare soil is more sudden in contrast to a longer and slower period of vegetation growth. Thus to capture sudden changes on a field, e.g. due to harvesting of the crop, this filter has the highest priority.

The result of this method is a new 3d array containing only 143 divided by the number of time steps. In case of 4 consecutive Sentinel 2 scenes the resulting array has a size of 36 time steps, i.e. 143 divided by 4 and rounded up.

It has to be noted that this type of feature engineering was not implemented out of theoretical but practical necessity, due to the given hardware limitations and to increase the model’s performance. As described in chapter 4, the core concept of NNs is to learn the patterns within datasets. Thus, the developed NN can learn the connections which are filtered by the above described method itself, e.g. data samples with a lower AOT value or a certain SCL value have more weight because they carry more important information. But the computational cost of using the whole dataset with all the fields that contain no or only low information values is many magnitudes larger.

When using the 3d Array with all 143 time steps, and thus not using the “filterCloudFree” method, the NN still achieved a good accuracy of 85% in several test runs. This accuracy could have been improved if hyperparameters had been fine-tuned further. This shows that the LSTM detects the patterns within the dataset despite the cloud data. However, it took roughly 120 seconds for one epoch and 150-200 epochs to achieve a stable validation accuracy. In contrast, using the “filterCloudFree” method and reducing the time steps to 36, reduces the training time of one epoch to only ca. 30 seconds, while a stable validation accuracy is achieved at roughly 50-100 epochs (cf.

figure 24). The total time needed to train one model for 250 epochs, as done in chapter 7, without reducing the time steps would be 8.3 hours, while the training time with reduced time steps is 2 hours. This calculation does not even consider the fact that without the filtering the network is achieving a stable accuracy much later and thus even more epochs might be needed to finalise the training.

In conclusion, the method “filterCloudFree()” is implemented to reduce training and prediction time significantly by a factor of 4. This time reduction allows a more extensive model selection, i.e. grid search for an optimal set of hyperparameters, with the given hardware limitations.

6.3.5. Interpolation of missing data

Even after the “filterCloudFree()” method has been applied, as described in the last chapter, the dataset contains “NaN” and “0” values. These values can’t be used as input to the NN, because a “NaN” quite literally is not a numerical value and the “0” value doesn’t represent the true pixel value of that field of that day. Given the circumstance that no data is available, the easiest way to generate valid and reasonable input values is via interpolation and extrapolation of existing data across time. Extrapolation of data is only necessary if invalid data occur at the first or last position of all time steps.

The method “interpolate_columns()” within the file “AgriNeuroUtils.py” (cf. Annex I) loops through the individual columns of each individual field. These 1d arrays contain the values of one Sentinel 2 band for all used time steps (cf. figure 21). Within each column all valid values are extracted. These valid values are then used to linearly interpolate and extrapolated the “NaN” and “0” values. The core function used in this process is the “scipy.interpolate.interp1d()” method (“SCIPY-API-Interpolate”). The result of this method is a 3d array which contains no “NaN” or “0” values.

6.3.6. Normalisation of the data

As a final step of the data preprocessing all input values have to be normalised. As Raschka and Mirjalili point out “*[...] the majority of machine learning and optimization algorithms behave much better if features are on the same scale [...]*” (Raschka and Mirjalili, 2018, p. 120). So far different bands have very different value ranges, e.g. while the SCL band has values ranging only from 0 to 11, other bands have values ranging up to 7 000. If left unchanged, the cost error function within the NN would be dominated by the larger errors of the larger values. To avoid this the different features of the dataset have to be brought onto the same scale via normalisation.

To keep the individual characteristic of each feature and sample the normalisation is only conducted within each Sentinel 2 band. The method “normalize_data()” within the file “AgriNeuroUtils.py” (cf. Annex I) normalises each Sentinel 2 band across all time steps and samples. The function “MinMaxScale()” ("SCIKIT-LEARN-API-MinMaxScaler") is used to scale the lowest value of each Sentinel 2 band to 0, the highest to 1 and all other values appropriately between. The result of this method is a 3d array with all of its features on a scale between 0 and 1.

6.3.7. Encode Y

To allow the NN to connect the input x data samples to the correct classification class, the y data has to be encoded. This means the y data has to be transformed from a text value, e.g. Winter Wheat, to a numerical value the NN can use. The encoding is done by creating a matrix with a column for each possible output class. For each input sample a row is created. In the rows a ‘1’ value is set in the one column with the correct solution, whereas all other columns of that row are set to ‘0’. The NN can then use this solution matrix to compare the predicted crop types with the actual crop types and consequently update the weights during backpropagation (cf. chapter 4.2.3). The method “encode_train_Y” encodes the y data in the above described way. It uses the “LabelEncoder” function from the scikit-library to achieve this (“SCIKIT-LEARN-API-LabelEncoder”).

7. Development and training of the LSTM model

7.1. Model selection

The development and training of the LSTM was conducted in python with the Keras library. (cf chapter 3.2). Keras offers two ways to create, train and use NN models: the Sequential model and the model class used with the functional API. (“KERAS – API – AboutModels”). The LSTM in this thesis was developed using the sequential model. This way the neural network is coded as a stack of layers using the following process (“KERAS – API – SequentialGuide”):

1. A Sequential object is created.
2. The “.add()” method of the sequential object is used to add a layer of a certain type e.g. LSTM layer, Dense layer, Dropout Layer etc.) with a set of hyperparameters (e.g. hidden nodes, input shape of data, dropout rate, recurrent dropout rate etc.). This step is repeated until all required layers are created.
3. The whole model is compiled with the “.compile()” method of the sequential object.
4. The model is trained with the “.fit” method of the sequential object. The training parameters, such as input x, y-true, number of epochs and batch size are set here. After each epoch of the training, the input x-validation samples are classified to return a validation accuracy for the data not used for training so far. This value gives a better, because less overfitted, esti mation of the NNs current capability to predict unseen data. When using kfold cross validation, the x-train and x-validation samples are swapped until each fold (here: 10) was used as x-validation dataset once (cf. chapter 6.3.2).

After the training, the “.predict” method of the sequential object can be used to predict new unseen data, like the evaluation dataset.

Setup and training of the NN, as well as the data preprocessing (as described in chapter 6.3.1 to 6.3.6), are organised within the file “AgriNeuro_Main.py” (cf. Annex I). This file loads the x- and y-data of the kfold splits, runs all the processes to prepare the data, sets the hyperparameters and runs the function: “*def lstm_model()*” in the file “AgriNeuroModel.py”, which returns the compiled LSTM model. The compiled model is then fitted to the provided train and validation data of the current folds. During the training process a callback function is initiated after each epoch to save the training log of each epoch and more importantly to save the trained model if the validation accuracy has improved compared to the best validation accuracy so far. Lastly, after all epochs are finished the currently trained model is evaluated and its history and parameters are stored in a csv file.

To find the optimal (or a very good) set of hyperparameters fitting the developed NN and dataset structure, multiple sets of parameters had to be tested and evaluated via a grid search. With the above described program structure it is possible to do this by providing lists of hyperparameters (as a Python list within the file "AgriNeuro_Main.py") and calculate and evaluate the NN for each combination of elements of these lists. After some initial tests which provided a rough orientation for good hyperparameters values, these values were used for a more intensive grid search.

The time frame window hyperparameter was set during data preprocessing, because it manipulates the input data, as described in chapter 6.3.4.

- Time Frame Window: Time frame of Sentinel 2 images to look for the best possible input data. Tested values are 1,4 and 7 Sentinel 2 scenes apart.

The following enumeration provides an overview of the optimised hyperparameters set when compiling the model:

- Number of LSTM Layers: Number of stacked layers in the NN. The tested values were 1 to 4 LSTM layers.
- Hidden nodes: Number of hidden nodes (or neurons) per LSTM layer. The tested values were 256, 512, 768 and 1024.
- Dropout rate: The dropout rate represents the percentage of connections between the stacked LSTM layers that are randomly dropped during each batch of learning. By randomly blocking or not using certain connections the network is forced to evaluate all possible input features and neuron connections. This way the network gets a broader view at the dataset and overfitting is reduced. The dropout rate is a so-called regularisation technique. These methods regulate the learning of the model, whereby the main goal is to reduce overfitting (Goodfellow et al., 2016). Following initial tests to find a general applicable range of the dropout rate, the tested values were 0.385 to 0.425 in 0.01 steps.
- Recurrent dropout rate: The recurrent dropout rate is a regularisation technique only used in RNN networks. It blocks random connections between time steps. Thus, it is a regularisation not between the LSTM layers (as the dropout rate) but between the sequential values, as

they propagate through the network. Following initial tests to find a general applicable range of the recurrent dropout rate the tested values were 0.19 to 0.21 in 0.01 steps.

- Optimiser: The optimiser is the function the NN is trying to minimise. This function evaluates the difference (or "distance") between ŷ and the y. Different optimiser functions use different methods to calculate this distance and are thus differently suitable, depending on the dataset and type of NN. The following four different optimisers were tested: "Adam", "SGD", "Adagrad", "RMSprop" ("KERAS – API – Optimizers").

The following hyperparameters were evaluated when calling the ".fit()" method during the training process:

- Batch Size: Number of data samples trained at the same time. The tested values were 64, 128 and 256.
- Epochs: Number of times the whole dataset is evaluated. After initial testing the value was fixed to 250 epochs.

This results in a total number of 2 304 different possible combinations of hyperparameters: 4 layers * 4 neurons * 4 dropout rates * 3 recurrent dropout rates * 4 optimisers * 3 batch sizes * 1 epoch. As one complete training and validation of 10 kfolds took roughly 20 hours, it would have taken 46 080 hours or 1 920 days to calculate all of these as 10-fold validation with the given hardware limitation. Obviously, this was not feasible. To reduce the training time three strategies were used to detect a range of promising parameter values:

1. The majority of parameters combinations were only calculated for two folds. This quickly returned a general range for each parameter and also a set of parameters that yielded good results while reducing the training time significantly.
2. Only a reduced epoch number of 150 was calculated which further reduced the training time.
3. Good values that seem to be valid for all combination of parameters were detected, e.g. after several tests it became clear that a batch size of 64 always provided a superior result, regardless of the other values. By finding parameter values that were working very well generally, the combinations of possible numbers of sets of parameters could be reduced.

Using these three steps it was possible to train and evaluate hundreds of models automatically within a reasonable time. This resulted in a set of 16 different most promising sets of the following hyperparameters:

Time Frame Window (4, 7), LSTM Layers (2, 3), Hidden nodes/neurons (512, 786), Dropout rate (0.4, 0.425), Recurrent dropout rate (0.2), Optimiser ("Adam"), Batch size (64), Epochs (250).

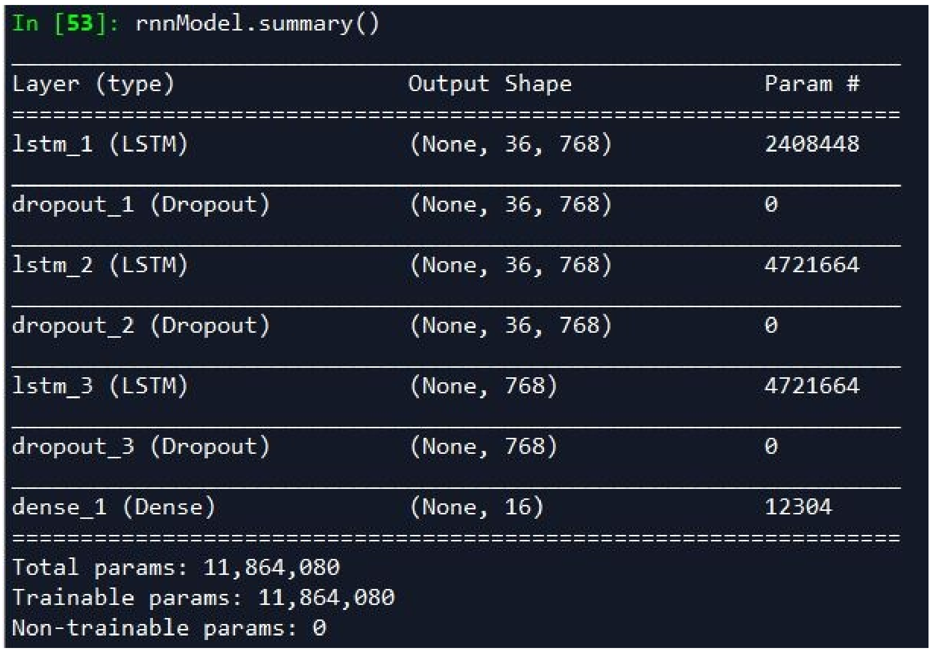
For all 16 combinations of these hyperparameters a k-fold cross validation of 10 folds was calculated. The best average validation accuracy was achieved by the following parameters:

Time Frame Window (4), LSTM Layers (3), Hidden nodes/neurons (786), Dropout rate (0.425), Recurrent dropout rate (0.2), Optimiser ("Adam"), Batch size (64), Epochs (250).

This set of hyperparameters is the result of the model selection. The following descriptions of the NNs training, its architecture and its evaluation refers to a model trained via k-fold cross validation with these parameters.

7.2. Training and architecture of the LSTM

A brief summary of the model can be created in Python with the "model.summary()" method, as shown in figure 22. The developed LSTM model has 11 864 080 parameters that are trained on the given input data. All of these parameters are used to find the lowest or a very low point of the cost function. The detailed structure of the model and the final hyperparameters values are displayed in figure 23. The NN consists of three stacked LSTM layers with the above described parameters. The three layers are used recurrently for t times, whereby t represents the sequential time steps of the input data. When using a time frame window of 4, t equals 36. The last layer is a dense fully connected layer, i.e. each neuron receives input from all neurons of the previous layer. This layer contains as much hidden nodes as y output classes. The "softmax" activation function of that layer attributes a percentage of confidence for each class for each sample that sums up to 100%. In this way, the layer provides an output which can be attributed to the available classes. Depending on the required accuracy a threshold can be applied here, so that only if a class is predicted with a confidence above the threshold, e.g. 90%, it is attributed as actual prediction, otherwise, i.e. if the confidence for all classes of that data sample is below the threshold, the sample is attributed as unknown (cf. chapter 8.1).



```
In [53]: rnnModel.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 36, 768)           2408448
_________________________________________________________________
dropout_1 (Dropout)          (None, 36, 768)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 36, 768)           4721664
_________________________________________________________________
dropout_2 (Dropout)          (None, 36, 768)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 768)               4721664
_________________________________________________________________
dropout_3 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                12304
=================================================================
Total params: 11,864,080
Trainable params: 11,864,080
Non-trainable params: 0
```

Figure 22: LSTM model summary in python console (own figure).

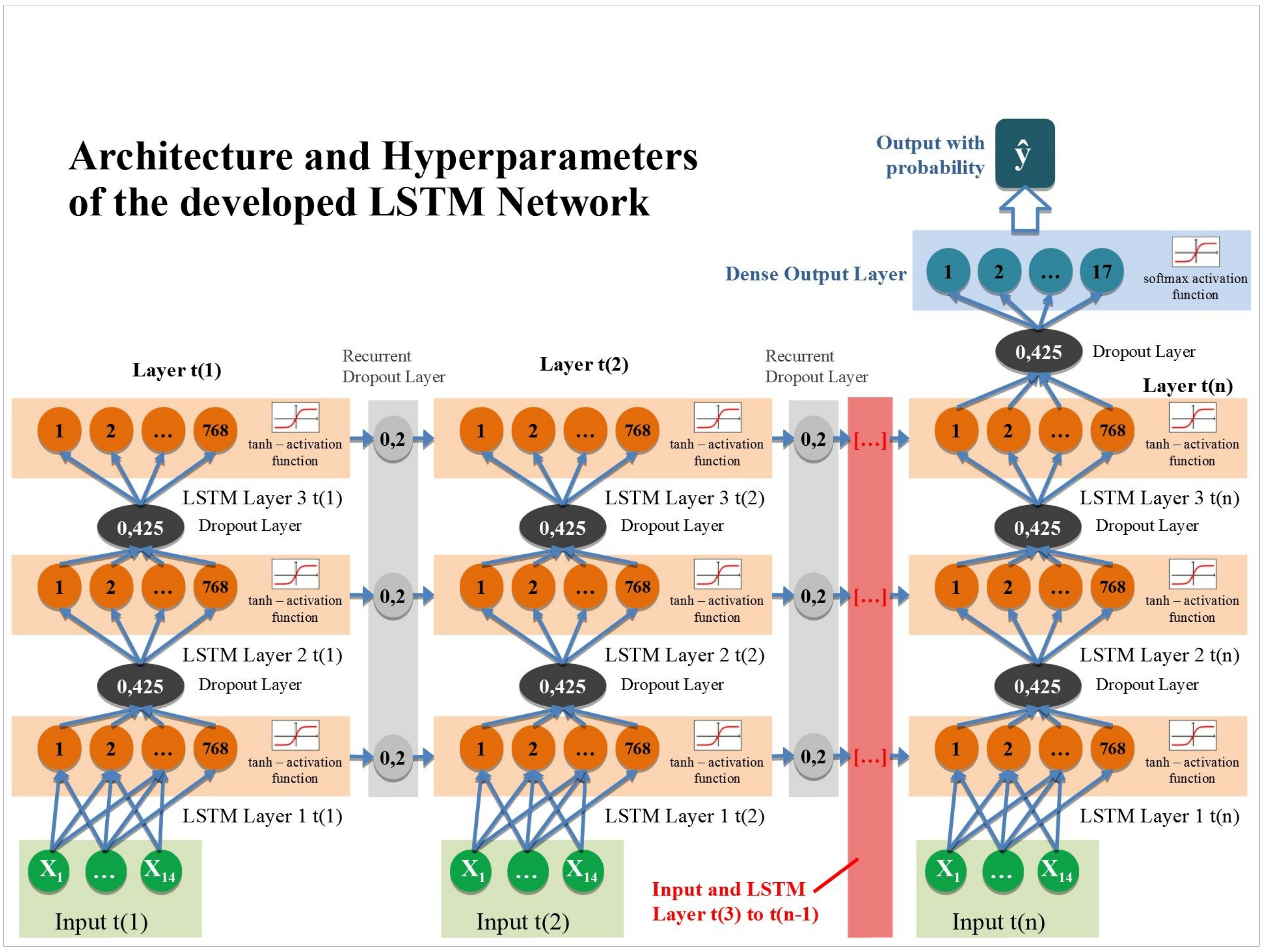


Figure 23: Architecture and Hyperparameters of the developed LSTM Network (own figure)

The actual training of the NN with the described structure and hyperparameters is visualized in figure 24. It does not display an individual training progress but rather the aggregated training progress of all 10 cross validation trainings. The light blue filled area displays the minimum and the maximum validation accuracy over all training runs at the current epoch. The dark blue line represents the mean validation accuracy for each epoch. As described above, a callback function was implemented to save the so far trained weights of the model in a ".hdf5" file after each epoch if the validation accuracy increased. This way for each of the 10 training runs the best model over all epochs was saved. The validation accuracy ranges from 93.16% up to 94.16%, while the mean validation accuracy is 93.56% (cf. table 4). The very low standard deviation shows that the model delivered nearly identical quality for all 10 training folds.

kfold	Max. Validation Accuracy	Min. Validation Loss Function
1	93.31	0.2997
2	93.39	0.2955
3	93.98	0.2594
4	93.26	0.2802
5	93.16	0.2678
6	93.76	0.3014
7	94.16	0.2708
8	93.62	0.2560
9	93.72	0.2747
10	93.31	0.3154
Mean	93.57 (sd) +/- 0.0034	0.2821 (sd) +/- 0.1989

Table 8: Max. validation accuracy and min. validation loss for each fold (own table).

Figure 24 shows the higher the epoch, the slower the increase in accuracy. At around 50 epochs most folds achieve a good accuracy of ca. 90%. The mean accuracy is then increasing only very slowly. During the training of the individual folds the peak validation accuracy occurs around epoch 150 to 225. As figure 24 shows, the mean validation accuracy is rising less and less and is tending to a limit between 0.9 and 0.95. Therefore no substantial increase can be expected by training the model for even more epochs.

During the training the probabilistic and approximation approach of NNs in finding a low point on the cost function became apparent in two occasions when the weights of the network were not converging. In these cases, the accuracy stayed around 50% even after 250 or more epochs of training.

This accuracy value corresponds to the frequency of the largest class: “grassland” (cf. table 7). So the network was only learning: “Predicting always grassland will result in a general accuracy of at least ca. 50%”.

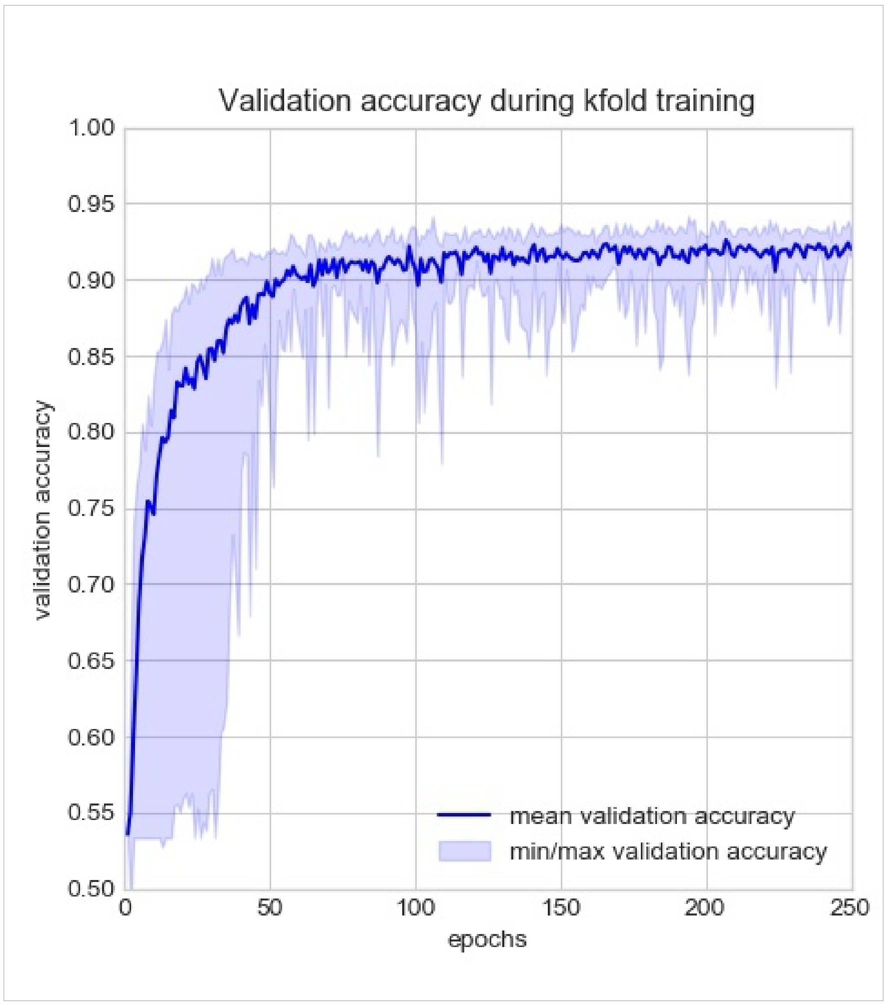


Figure 24: Validation accuracy during kfold training (own figure).

This problem was certainly largely increased by the very uneven distribution of the data, but also shows the influence of the random initial set of the neuron’s weights. As pointed out in chapter 4.2.3 the multi-dimensional cost function has many local minima, many saddle points and large very flat regions, all of which make optimisation difficult. Figure 25 shows a greatly simplified example of a function with only one input parameter x and multiple local minima. When trying to find a low point by only following the descend in incremental steps, like NNs do (cf. chapter 4.2.3), the outcome of training is influenced by the starting point of the initial weights. Selecting a starting point further to the right, in figure 25, will result in a less optimal minimum compared to starting points further left. Although NNs can overcome local minima, e.g. by changing the learning rate, they can also be trapped within, as it happened in the described instances. The initial starting point is so far off to any better local minima that the best solution to the cost function is for the NN to always predict the most frequent class (i.e. grassland).

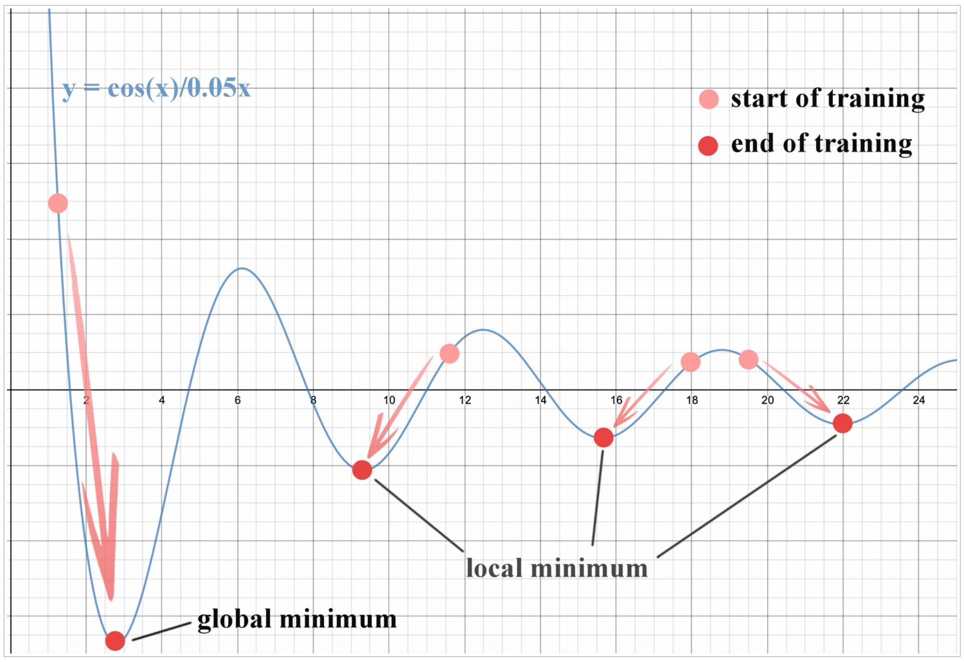


Figure 25: Simple cost function with multiple local minima (own figure)

In these cases the LSTM, together with the initial neuron's weights, was reset and trained again with the same dataset, i.e. the current training and validation fold, which resulted in greatly enhanced accuracy. This shows that for a more precise assessment of the accuracy and to find the best accuracy, i.e. a lower point on the cost function, with the given hyperparameters the neural model should be retrained with the same k-fold splits multiple times. It also has to be noted that this problem doesn't influence or occur on the already successfully trained model because the weights are already set and fit to the input data and there is no random initialisation of the weights.

8. Evaluation of the developed LSTM neural network

The results of the cross validation described in the last chapter already presented a good estimation of the prediction abilities of the network, as these results were already taken from 10 different validation folds. However, due to the grid search repeatedly using the same training and validation datasets, there is the danger of influencing even the validation dataset. Therefore, an evaluation dataset, containing 15% of the samples, was created and split from the total dataset before any training occurred (cf. chapter 6.3.2). Rußwurm and Körner justify this procedure of creating a third dataset as follows: *"To ensure that these parameters are chosen independently, training of network weights and evaluation of hyper-parameters was performed on training and validation datasets, respectively. A third evaluation dataset is used for to calculate accuracy measures of neural network independently from network weights and parameters."* (Rußwurm and Körner, 2017 p.554). For this reason, all results presented in this chapter are calculated using the independent dataset split off before the training. Following this procedure allows an evaluation of the prediction power of the NN on unseen data.

The evaluation of the LSTM can be done with the methods developed in the "EvaluateLSTMModel.py" file (cf. Annex I). To predict data with the LSTM, the data has to be prepared the same way the training data was prepared. This can be easily done with the already developed methods as described inc chapter 6.3.1 to 6.3.6. The test dataset is manipulated in the same way as the training data, so that the final output is a normalised evaluation dataset. The next step is to load the .hdf5" file containing the NN with its trained weights into python. Then the preprocessed evaluation dataset can be used as input for the "prediction()" method of the loaded model with the following line of code:

```
prediction_matrix = lstmModel.predict(normalised_evaluationDataX)
```

The result of the "prediction()" method is a prediction matrix containing for each input sample the "softmax" confidence percentage for each possible output class. The prediction matrix has the same order as the original input sample, thus the data can be reconnected to the actual field and its information e.g. location, crop type. In this manner, the prediction ability of the model can be evaluated. in detail. Multiple samples of the actual prediction matrix are displayed in table 9. The first column contains an internal unique number to identify each field. The second column contains the actual crop of the field as provided with the dataset. The third column "y predict" (or ŷ) contains the prediction of the NN. The following 16 columns contain the softmax confidence score for each class. The class with the highest confidence is the y predict result. Table 9 shows that many predictions are given with a high confidence of above 0.95. The last line (id: 1592) gives an example of a wrongly predicted field. The actual field contains “Summer Barly” which was predicted with a confidence of 0.309, but potato was wrongly predicted with a higher confidence of 0.601.

Id	y true	y predict	Field Bean	Pea	Grass - Clover - Bur clover	Potato	Maize	Summer Barley	Summer Oats	Summer Wheat	Other Seeds and Herbs	Grassl and	Winter Barley	Rape	Winter Rye	Triticale	Winter Wheat	Sugar Beet
47	Winter Barley	Winter Barley	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
189	Rape	Rape	0	0	0	0	0	0	0	0	0	0	0.001	0.999	0	0	0	0
461	Winter Barley	Winter Barley	0	0	0	0	0	0	0	0	0	0	0.995	0	0.005	0	0	0
1002	Winter Wheat	Winter Wheat	0	0	0	0	0	0	0	0	0	0	0.002	0	0	0	0.998	0
1066	Rape	Rape	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1183	Wiese	Wiese	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1212	Rape	Rape	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0.098	0
1280	Winter Barley	Winter Barley	0	0	0	0	0	0	0	0	0	0	0.987	0	0	0	0.013	0
1414	Sugar Beet	Sugar Beet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1415	Winter Wheat	Winter Wheat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1457	Summer Barley	Summer Barley	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1592	Summer Barley	Potato	0	0	0	0.601	0.066	0.309	0.002	0.009	0	0.001	0	0	0.007	0.003	0.002	0

Table 9: Samples from the prediction matrix (own table)

Based on the prediction matrix different classification scoring metrics referring to the total dataset and to the individual classes were calculated to evaluate the model’s performance. These were selected based on the overview table of the most important performance metrics in the article “*Deep learning in agriculture: A survey*” (Kamilaris and Prenafeta-Boldú, 2018, table 1 p.75). All these classification metrics were coded in python using the module “metrics” from the “sklearn” library ("SCIKIT-LEARN-API-Metrics").

Since this evaluation is covering a multi-class classification and some of the scoring metrics are designed for binary classification problems, these metrics are calculated via one-versus-all classification. The metrics calculated below are based on the concepts of evaluating the relationship between the actual crop type and the predicted crop type. There are four possible relations between y and ŷ. In a one-versus-all classification these are considered, as the name suggests, from the point of one class versus all other classes. For example, based on the crop type grassland these four categories are:

1. True positives (TP): All grassland instances that are classified as grassland.
2. False positives (FP): All non-grassland instances that are classified as grassland.
3. False negatives (FN): All grassland instances that are not classified as grassland.

4. True negatives (TN): All non-grassland instances that are not classified as grassland.

8.1. General classification scoring metrics

The general classification metrics give a picture of the LSTM networks abilities to predict new unseen data over all classes. As mentioned above, the metric types were selected based on their usefulness to evaluate classification abilities and on most well-known metrics selected by other articles to provide a comparison (Kamilaris and Prenafeta-Boldú, 2018 and Rußwurm and Körner, 2017). When calculating the metrics over all classes, different types of averaging all classes can be selected. Due to the strong class imbalance in the dataset, the best method of averaging is the micro-average method, which weights the scoring metrics of each class by the frequency of that class. This way the number of samples in each class and the correctly predicted samples are respected.

Metric	Performance of evaluation data set
Overall accuracy/ Precision / Recall	92.36
F1-score	92.36
AUC	99.53

Table 10: Weighted macro-average of evaluation data set (own table).

Table 10 displays the core classification evaluation metrics over all samples. The test sample contains 1 754 samples, out of these 1 620 were classified correctly and 134 incorrectly. This results in an overall accuracy of 92.36%. The low interval between the accuracy of the evaluation data and the average accuracy of the training-validation dataset of 93.57% (cf. table 8) proves that no significant overfitting of the network occurred during training. Precision and recall are two different measures of accuracy or ways to set TP, TN, FP and TN in relationship. Precision returns the ratio of the correctly predicted samples to the total predicted samples, while recall returns the ratio of the correctly predicted samples to the total input samples. When determining the predicted crop simply by selecting the crop type which has the highest softmax confidence (cf. table 9), precision and recall and the F1-score, the harmonic mean between those two, are the same, because in total the number of input and output samples are the same. However, when a threshold is implemented to the result of the softmax function the confidence, precision and recall scores can be quite different. In this case, for a prediction to be valid as correct or incorrect the softmax confidence has to be higher than the threshold value. Any prediction with a confidence lower than the threshold is put into the category ‘unknown’. This procedure can be used to increase precision, i.e. reduce the FP values, at the cost of the recall score. Table 11 shows precision and recall for different thresholds. Without any threshold, there are 1 620 samples classified correctly and 134 incorrectly. At a threshold of 0.5 four correct and six incorrect samples are categorised as “Unknown”. This lowers the recall value to 92.13, whereas the precision value is slightly increased to 92.66. Any further increase of threshold lowers the recall value further because more and more samples become unknown and thus less samples from the actual input are classified correctly. At the same time the precision is increased, because of the total predicted samples more and more are predicted correctly. At a threshold of 0.99, 96.33% of the predicted samples are correct, but of the total input samples only 85.35% are classified correctly.

Threshold	**0.99**			**0.90**			**0.7**			**0.5**		
Category	**Nr**	**Recall**	**Prec**	**Nr**	**Recall**	**Prec**	**Nr**	**Recall**	**Prec**	**Nr**	**Recall**	**Prec**
Correct	1 497	85.35	96.33	1 564	89.17	94.67	1 605	91.51	93.42	1 616	92.13	92.66
Incorrect	57	3.25	3.67	88	5.02	5.33	113	6.44	6.58	128	7.3	7.34
Unknown	200	11.4	-	102	5.82	-	36	2.05	-	10	0.57	-

Table 11: Precision and recall at different threshold values (own table)

The different thresholds are also important for the “AUC” score which is 99.52 for the test dataset (cf. table 10). “AUC” stands for “Area under the ROC Curve”. It is the integral of the ROC curve (receiver operating characteristic curve) which expresses the relation of the true positive rate (or recall) and the false positive rate at all possible classifier threshold values (Raschka and Mirjalili 2018 and “Dev-google-machine-learning”). The diagonal of these relations, i.e. if the true positives and the false positives rates are the same for all threshold values, represents a line of random guessing. This implies with an AUC score of 50 a model is no better than random guessing. An AUC score of 0 shows that the model is inverting the results and an AUC score of 100 shows that the network classifies the result perfectly. The AUC score thus tells how capable a model is at distinguishing between classes.

The described metrics show that the model is in general very potent in classifying and distinguishing the input data to the possible output classes. The already satisfying accuracy of 92.36% can be increased even further to a precision of 96.33% with a recall rate of 85.35% if the correct predicted samples are more important than the amount of predicted samples. The high AUC score shows that the model is able to distinguish between the individual classes. To allow a more in depth analysis why the NN predicts some inputs wrong it is important to take a look at the evaluation metrics of the individual classes. The next chapter will analyse those class based metrics.

8.2. Classification metrics per individual class

A confusion matrix offers a first overview of the prediction abilities of the LSTM for the individual output classes. Table 12 shows such a confusion matrix of the evaluation dataset and its total 1 754 samples. The rows represent the actual crop type, while the columns represent the predicted crop types. Thus, the row "ŷ-predict" sums up all the prediction for that crop type, whereas the last column "y-true" sums up the actual crop types. The crossing cell of the same crop type in column and row displays the number of correctly predicted samples. These crossing cells are marked green. The darker the green, the higher the precision of that crop type, i.e. the quantity of correctly predicted samples in relation to all predicted samples of that crop type. All other cells display incorrectly classified samples and are marked red.

The rows of the confusion matrix show for each actual class into what type of crop it has been classified. The columns of the confusion matrix give the information out of which actual crop types each predicted crop type is composed. For example: There are 5 actual potato fields within the evaluation dataset. Out of these only 1 has been correctly classified, while 4 have been misclassified as maize. This results in a recall rate of 0.2 (cf. table 13). At the same time, the network classified a total of 4 fields as potatoes. Again 1 of these predictions is correct, whereas 2 maize fields and one summer oats field were misclassified as potato field. This results in a precision rate of 0.25. Overall the confusion matrix shows how the actual classes and the predicted classes spread out into other crop types.

From the actual crop types, the following 2 crop types spread out the most and are classified into 6 other crops: Summer Barley and Winter Wheat. While the class "Grass - Clover - Bur clover" does not spread out as much into other classes, 25 of its samples are misclassified as grassland, as much as the correctly classified 25 samples. This shows that there are very similar features (within the Sentinel 2 data) from the first to the second class. The other way around, only 3 grassland samples are misclassified into the: "Grass - Clover - Bur clover" class. On the other hand, 5 classes spread out into only one other class type, these are: Pea, Potato, Maize, Other Seeds and Herbs and Sugar Beets. Looking at the columns, the following three predicted crops consist of 6 or more actual crop types: Grass - Clover - Bur clover, Grassland and Winter Wheat. Two crop types are classified with perfect (100%) precision, i.e. all predicted fields consist only of the actual crop type, these are: Field Bean and Other Seeds and Herbs.

Crop Types	Field Bean	Pea	Grass - Clover - Bur clover	Potato	Maize	Summer Barley	Summer Oats	Summer Wheat	Other Seeds and Herbs	Grass-land	Winter Barley	Rape	Winter Rye	Trit-icale	Winter Wheat	Sugar Beet	**y-true**
Field Bean	4	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	6
Pea	0	12	0	0	0	0	1	0	0	0	0	0	0	0	0	0	13
Grass - Clover - Bur clover	0	0	25	0	0	0	1	1	0	25	0	0	0	0	0	1	53
Potato	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0	5
Maize	0	0	0	2	76	0	0	0	0	0	0	0	0	0	0	0	78
Summer Barley	0	1	1	0	0	57	4	3	0	0	0	2	0	0	3	0	71
Summer Oats	0	0	0	1	2	4	1	3	0	1	0	0	0	0	0	0	12
Summer Wheat	0	0	1	0	0	3	1	11	0	1	0	0	0	0	4	0	21
Other Seeds and Herbs	0	0	0	0	0	0	0	0	4	2	0	0	0	0	0	0	6
Grassland	0	0	3	0	1	0	0	0	0	936	0	0	0	0	0	0	940
Winter Barley	0	0	2	0	0	0	0	0	0	4	65	0	0	2	11	0	84
Rape	0	1	0	0	0	0	0	0	0	1	2	142	0	0	2	0	148
Winter Rye	0	0	0	0	0	0	0	0	0	0	0	0	11	1	5	0	17
Triticale	0	0	0	0	0	0	0	0	0	1	1	0	0	10	5	0	17
Winter Wheat	0	0	0	0	0	1	0	1	0	4	7	0	2	2	255	0	272
Sugar Beet	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10	11
ŷ-predict	4	14	34	4	83	66	8	19	4	975	75	144	13	15	285	11	1754

Table 12: Confusion matrix of the test dataset (own table)

Using the numbers from table 12, the precision, recall and F1 score per class can be calculated. This returns a more detailed look at the predicted capabilities for each class. Table 13 displays these scores as well as the samples in the datasets. The table is sorted after the quantity of the samples. This reveals one major influence on the prediction capability of the network: the quantity of the training samples. A clear trend is visible in table 13: the more dataset samples a class has, the higher the precision, recall and F1-score. Classes with more than 400 training samples, i.e. train-validation sample, have an unweighted average F1-score of 0.91, ranging from 0.83 to 0.98. In contrast, crop types with less than 400 (here: 302 or less) training samples have an unweighted average F1-score of 0.62, ranging from 0.10 to 0.91. The summer oats class with only 68 samples has the lowest F1-score of 0.10. On the other hand, some classes like “Sugar Beets” and “Other Seeds and Herbs” have good results with an F1-score of 0.91 and 0.80 respectively despite their low number of training samples. Thus, these classes not only have a lower average F1-score, but also a much higher variance, than classes with more than 400 samples. Therefore, apart from the number of samples others factors seem to influences the quality of the classification. These different factors contributing to incorrect predictions of samples will be discussed in the next chapters.

Crop Type	Train/Validation Samples	Test Data Samples	Precision	Recall	F1-score
Grassland	5 323	940	0.96	1.00	0.98
Winter Wheat	1 543	272	0.89	0.94	0.92
Rape	839	148	0.99	0.96	0.97
Winter Barley	477	84	0.87	0.77	0.82
Maize	442	78	0.92	0.97	0.94
Summer Barley	403	71	0.86	0.80	0.83
Grass - Clover - Bur clover	302	53	0.74	0.47	0.57
Summer Wheat	116	21	0.58	0.52	0.55
Winter Rye	97	17	0.85	0.65	0.73
Triticale	97	17	0.67	0.59	0.63
Pea	71	13	0.86	0.92	0.89
Summer Oats	68	12	0.13	0.08	0.10
Sugar Beet	60	11	0.91	0.91	0.91
Other Seeds and Herbs	37	6	1.00	0.67	0.80
Field Bean	31	6	1.00	0.67	0.80
Potato	27	5	0.25	0.20	0.22

Table 13: Precision, recall and F1-Score for each output class (own table).

8.3. Evaluation of miss-classification

As described above, of 1 754 samples in the evaluation dataset 134 or 7.64% were misclassified. There are multiple possible reasons why the LSTM network does not recognise a sample correctly. Reasons can be found within the network itself, e.g. the architecture or the hyper-parameters, or in the quantity and quality of the input data, e.g. more input data needed, more diverse training data, etc. The complexity of NNs and their internal computations make it very challenging to extract why data was classified as it was. To examine in detail the possible sources of error of the developed LSTM network is beyond the scope of this master thesis. However, within the geo-informatics context of this study, it is possible to analyse relations between the spatial input data and the output results and thus influencing factors of the input data on the classification results. The following 4 will be analysed:

1. Influence of size: The size of the field polygon.
2. Influence of geography: The location of the field polygon.
3. Influence of features: The similarity of classes.
4. Influence of quantity: The quantity of samples of each class.

8.3.1. Influence of size

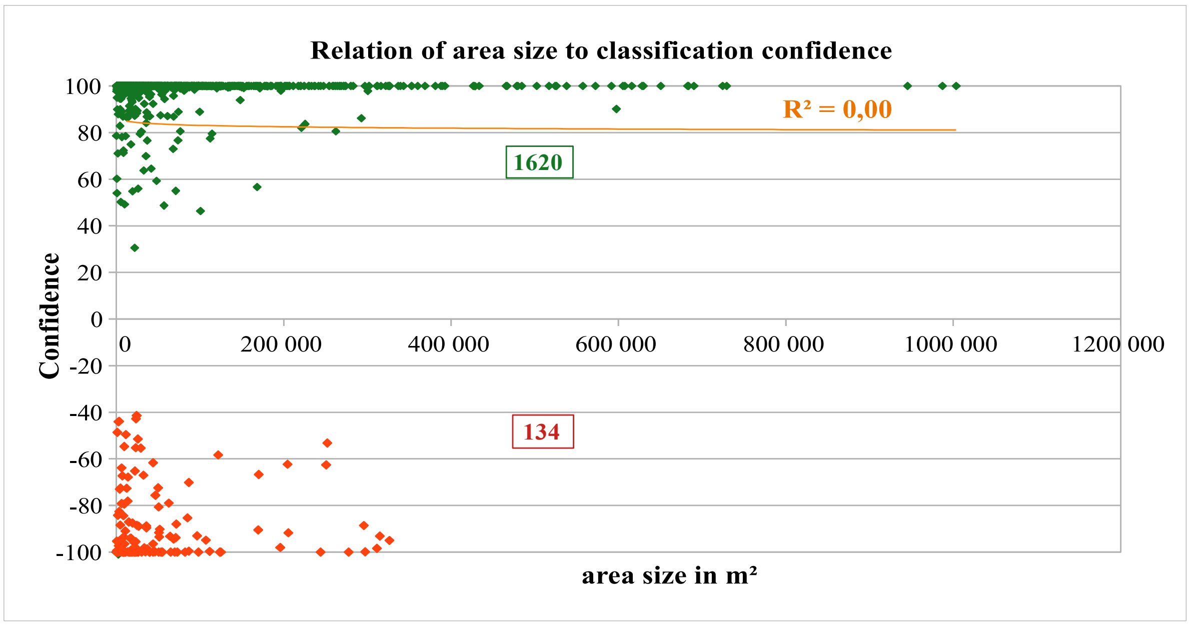


Figure 26: Relation of area size to classification confidence (own figure).

Since the input data is the result of the averaged pixels of each polygon, the fewer pixels a field has, the higher the influence of individual pixels, which might disturb the prediction. To analyse this relation, a diagram was developed which sets the confidence of a prediction in relation to the field size (cf. figure 26). For each data sample in the evaluation dataset one point was placed. The y-axis presents the maximum confidence for that predicted sample, while the x-axis presents the size in m² of that field. The confidence of any incorrect prediction (red dots in figure 26) are multiplied by -1, so they contribute negatively to the correlation of the field size. Most of the correctly classified samples are located in the top left and are visually not distinguishable due to the high concentration. The coefficient of determination (R^2) between size and confidence is 0. Thus, on an individual field level there is no influence of the field size on the prediction results.

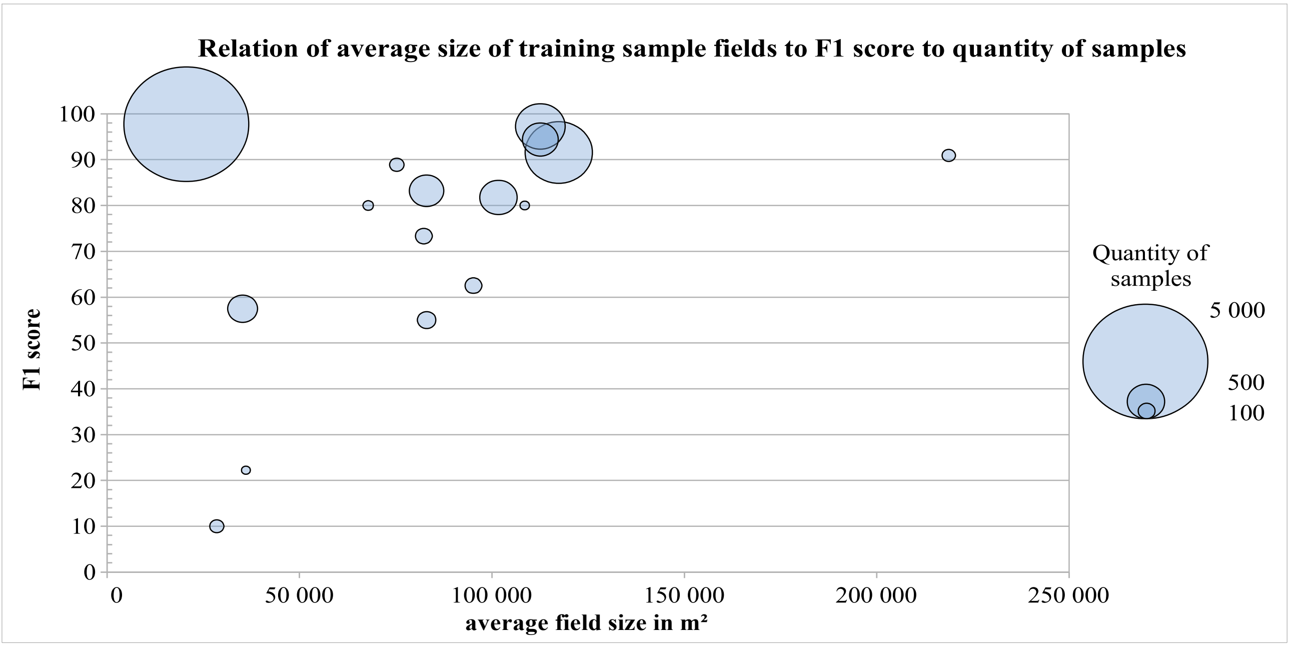


Figure 27: Relation of average field size of training sample to F1-score to quantity of samples (own figure).

On a class level, i.e. if the prediction results are grouped by crop type and set in relation with the F1-score, they have a correlation of R^2=0.26 (cf. figure 27). However, it has to be noted that this correlation is calculated unweighted, i.e. the quantity of the class is not taken into account. Therefore, figure 27 also shows the quantity of each crop type as size of the data points. For example the largest points in the top left corner represents the grassland class, with a sample size of 5 323, an average field size of ca. 20 000m² and a F1-score of 97.75, while the point to the far right represents sugar beets with a sample size of 60, an average field size of ca. 220 000m² and a F1-score of 90.91. Combining the size and position of the data points in figure 27 indicates already that the average field size influences classes with a small sample size much more than large classes. Figure 28 visualizes this effect. If only the smallest 10 classes, with each less than 400 samples (cf. table 7), are taken into account the coefficient of determination is 0.43, while in contrast using only the 6 largest classes, returns a coefficient of -0.05 (cf. figure 28). Two conclusions can be drawn from this:

1. The quantity of samples has a major influence on the F1-Score (and thus Precision and Recall). This will be further analysed in chapter 8.3.4.
2. The more samples are within a class, the less it is influenced by the average field size and vice versa. In detail:
   a) The classification accuracy of small classes is influenced by the area size of the field so that generally the larger the average field size in m² the better the F1-score. This is because at a small area, spikes in pixel values have a much stronger impact on the classification result of a field and, due to the small class size, this affects the overall class accuracy.
   b) The classification accuracy of large classes is not influenced by the area size of the field. This was true for the dataset used in this thesis if a class has more than 400 samples.

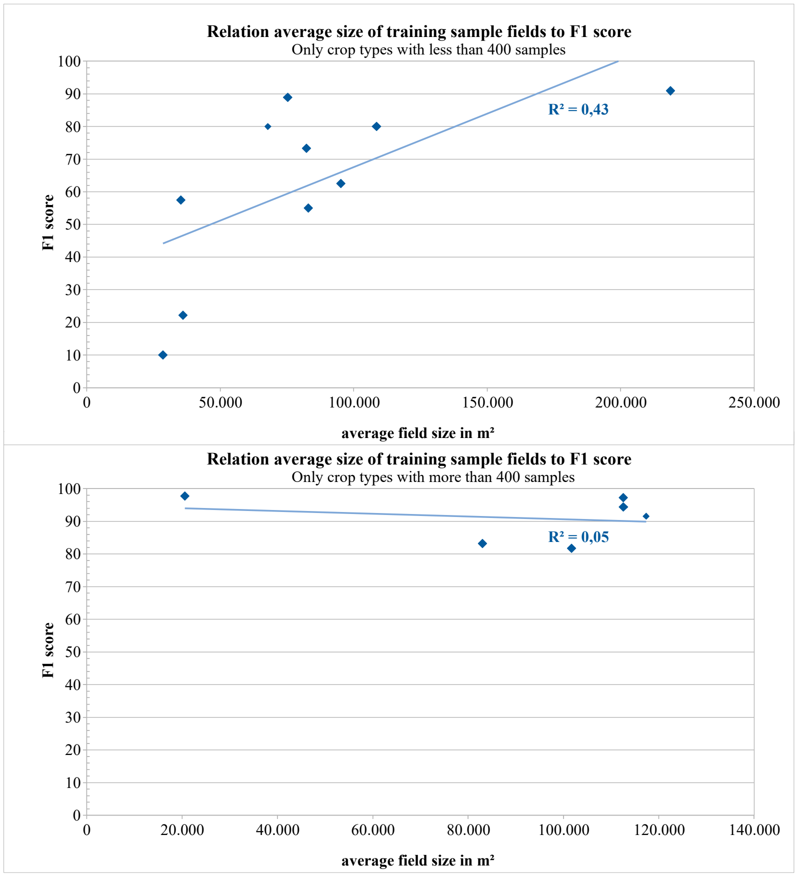


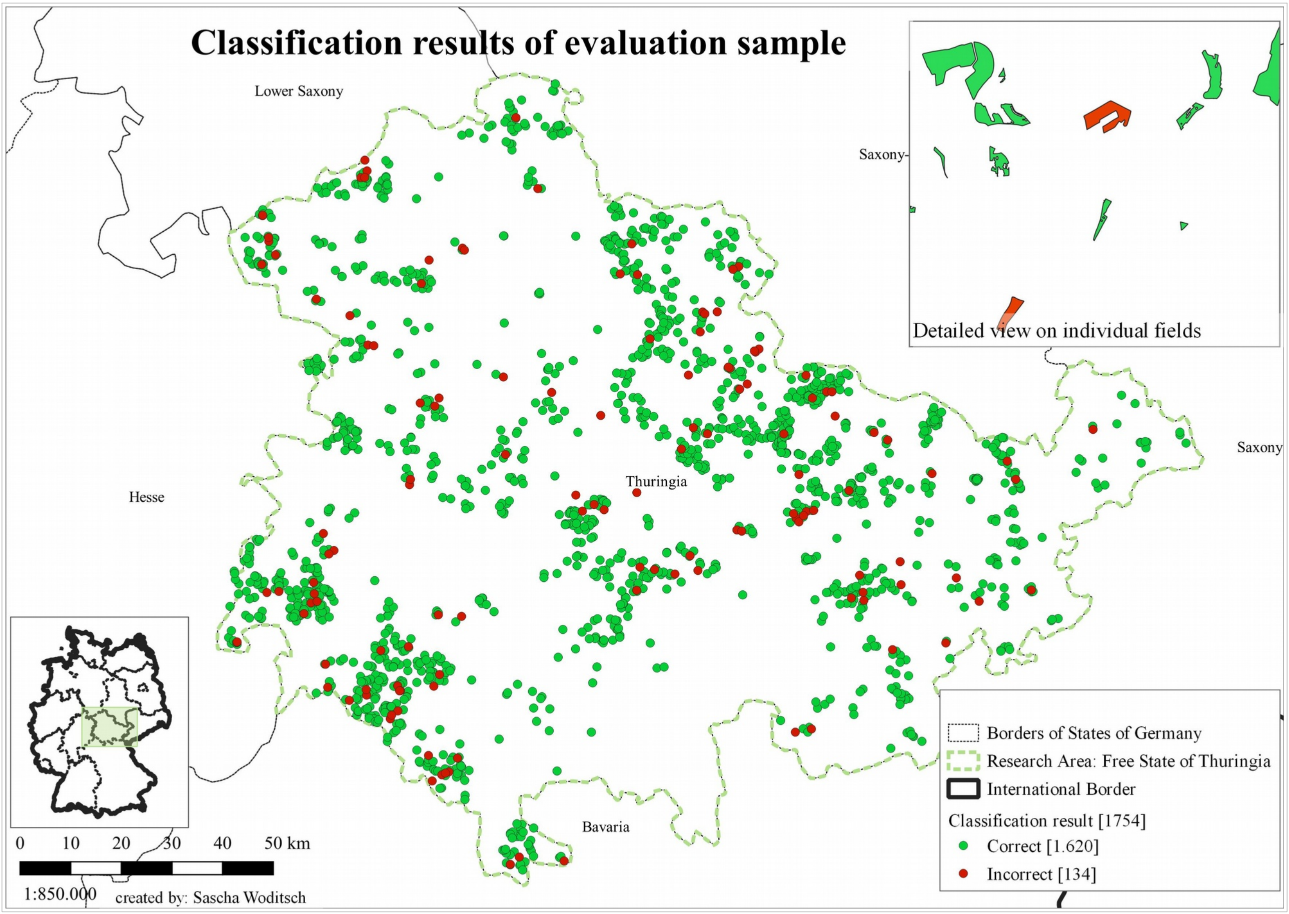
Figure 28: Relation of average field size of training samples of small and large classes to F1-score (own figure).

8.3.2. Influence of geography

If fields are clustered in one area, it might influence the outcome for various reasons, like:

- Locally worse or better input data, e.g. due to significantly more cloud coverage.
- Locally different features of the input data, e.g. due to local farming practices that differ from the rest of the survey area and thus change the actual pattern of the sequential data input.

To test this hypothesis, three approaches are tested. First, visually by creating a map that shows all correct and incorrect classified samples of the evaluation dataset. Second, the correlation between the classification confidence and the X- and Y-UTM position of each sample is analysed. Lastly, a spatial point pattern analysis is conducted to compare the standard distance of each crop class with its F1-score. An overview map of the correctly and incorrectly classified samples is created by reconnecting the prediction dataset via the field polygon id with the original geodata file. Figure 29



Classification results of evaluation sample
Lower Saxony
Saxony-
Detailed view on individual fields
Hesse
Thuringia
Saxony
Bavaria
0 10 20 30 40 50 km
1:850.000
created by: Sascha Woditsch
Borders of States of Germany
Research Area: Free State of Thuringia
International Border
Classification result [1754]
● Correct [1.620]
● Incorrect [134]

Figure 29: Classification results of the evaluation sample (own figure)

displays a map of the classification results. The green dots represent correctly predicted crop types, while the red ones represent the opposite. The detail map in the top right corner shows that the results can also be tracked down to the individual field level. Due to data protection legislation the detailed map is not localised within the survey area. The distribution of the green and red dots on map 29 exhibits no evident clusters. This would be the case if the classification results would be influenced by local variables so that there would be significantly more correct or incorrect predictions in one area.

Another way to analyse if a certain direction of the survey area produces better or worse classification results is the coefficient of determination between the X or Y-UTM coordinates of the fields centroids and their classification confidence. Diagram 30 displays the results of this calculation. As above in figure 26, the confidence of incorrect classification results is multiplied with -1 and is displayed in red in figure 30. Both coordinates show no signs of any correlation with the classification confidence. Thus the hypothesis of an absolute geographical influence can be rejected.

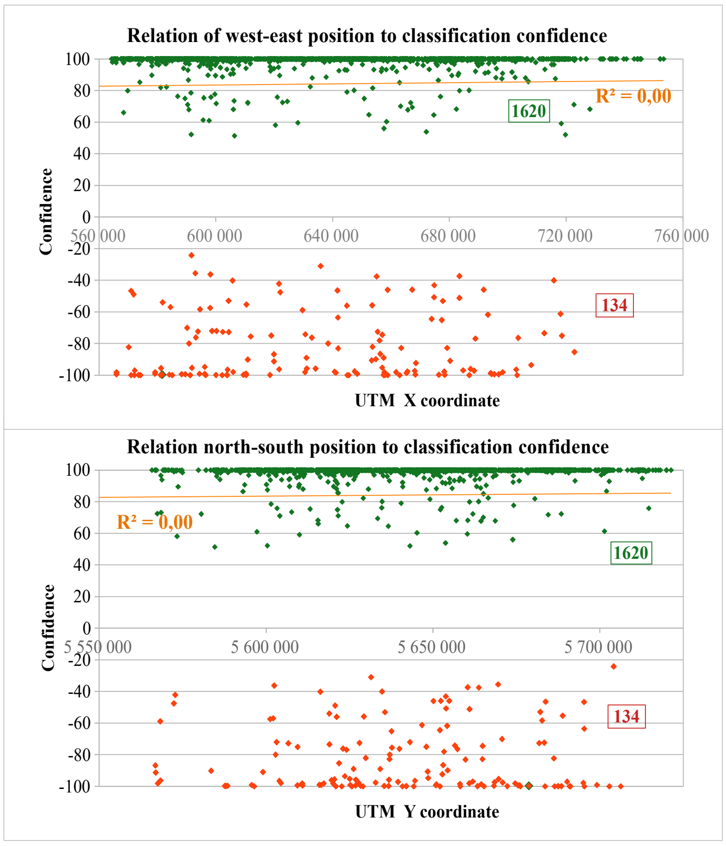


Figure 30: Relation of position to classification confidence (own figure).

However, to further analyse the influence of geography, not only the absolute but also the relative position of the fields has to be considered. For that reason, a spatial point pattern analysis is conducted for each crop type of the total dataset and the standard distance is calculated using Qgis.

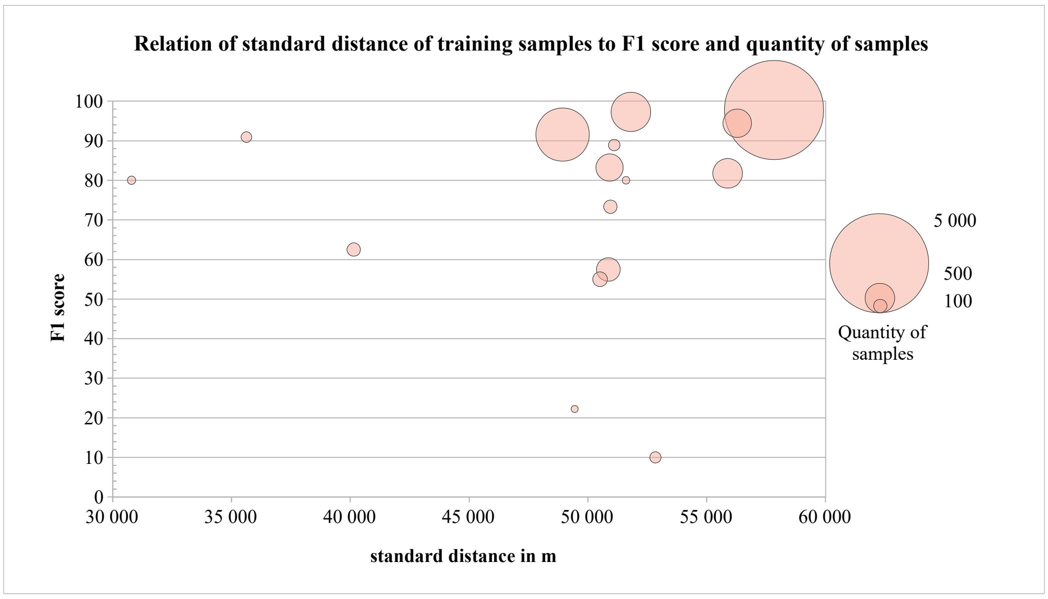


Figure 31: Relation of standard distance of training samples to F1-score and quantity of samples (own figure).

Crop Type	Standard distance in m
Grassland	57 836
Maize	56 288
Winter Barley	55 884
Summer Oats	52 841
Rape	51 808
Field Bean	51 604
Pea	51 106
Triticale	50 948
Summer Barley	50 914
Grass - Clover - Bur clover	50 861
Summer Wheat	50 511
Potato	49 442
Winter Wheat	48 937
Winter Rye	40 147
Sugar Beet	35 628
Other Seeds and Herbs	30 794

Table 14: Standard distance of crop types of the total dataset (own table).

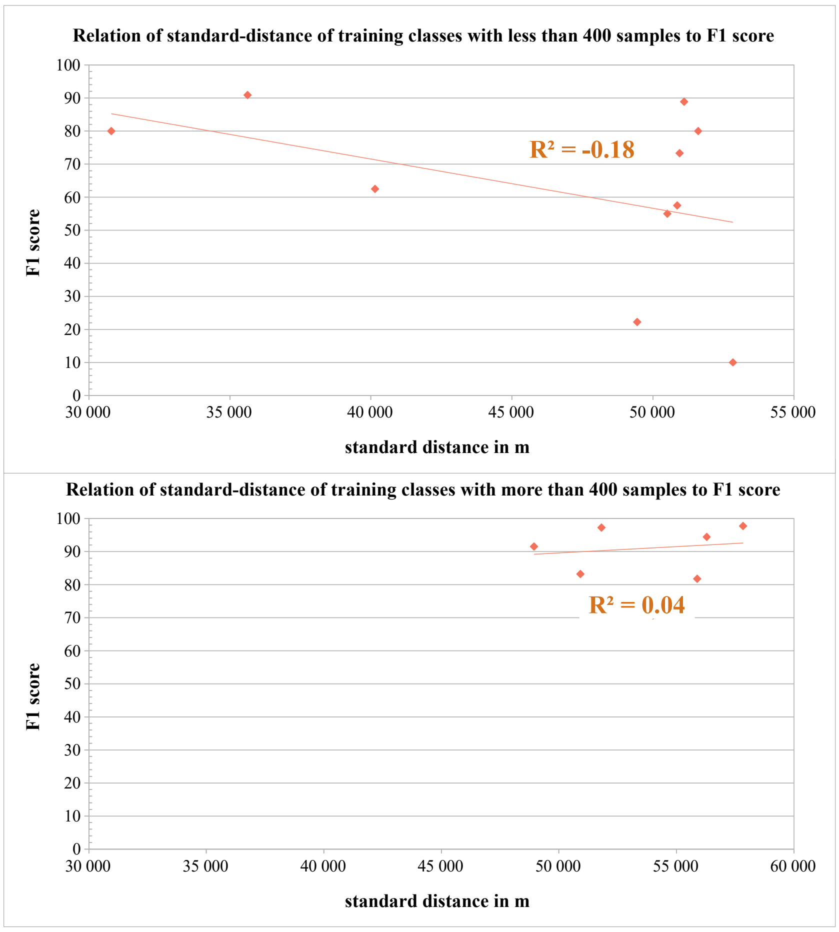


Figure 32: Relation of standard-distance of training classes with less and more than 400 samples to F1-score (own figure).

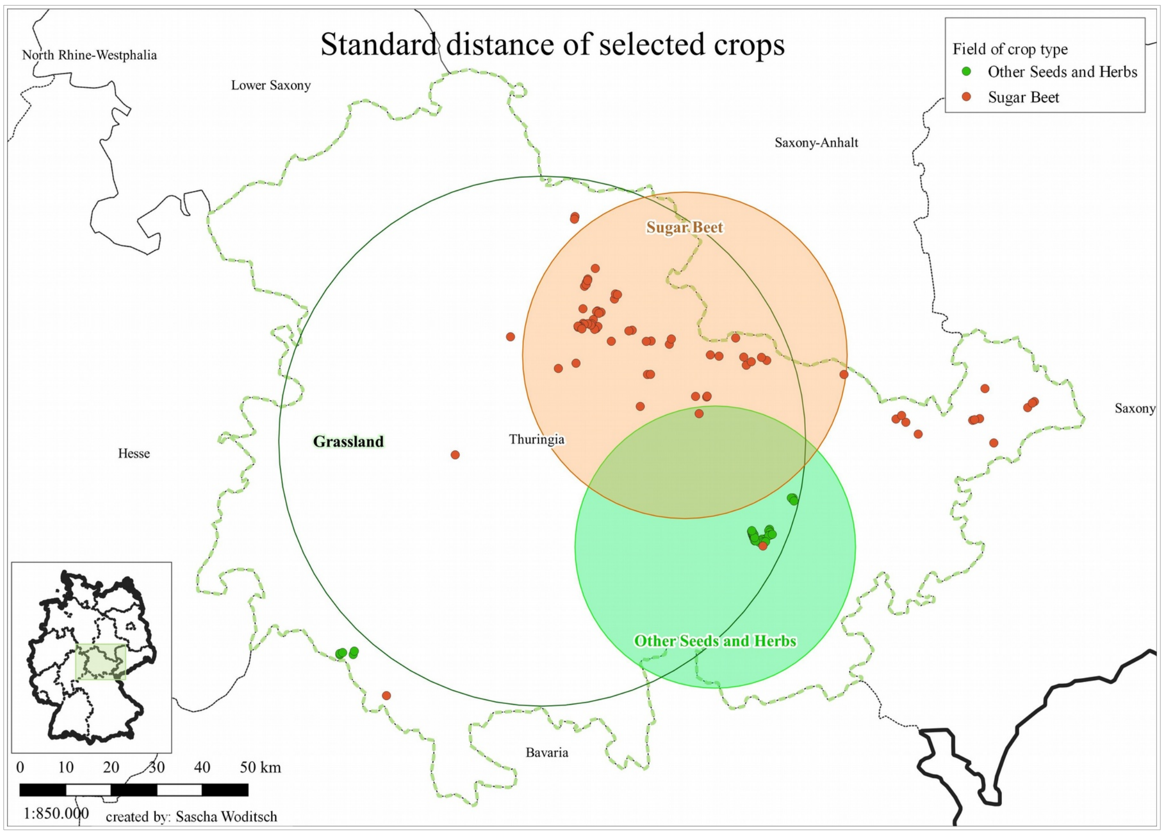


Figure 33: Map of standard distances of selected crops (own figure).

The standard distance is the shortest distance of all field centroids to the arithmetic mean centre of all field centroids. Table 14 lists the standard distance for all crop classes. The standard distance is shown in the map above (cf. figure 33) for selected crops. Since the grassland class is the largest class, with 6 263 samples, it is expected to be evenly distributed over the total survey area of Thuringia. This even distribution is visually confirmed via the transparent, larger central circle in figure 33, representing the standard distance of the grassland class. In contrast to this, are the standard distances of the sugar beets and the other seeds and herbs class. Both of these classes are locally concentrated, the latter in particular with a standard distance of only 30.794m (cf table 14). This clustering might influence the accuracy of these classes. Figure 31 displays the relation of the F1-score to the standard distance and to the number of the samples. As it is to be expected, figure 31 shows that the larger the quantity of a class, the higher the standard distance. The general coefficient of determination between the F1-score and the standard distance is 0. However, like in the last chapter, there is a difference when considering the size of the classes, as shown in figure 32. For larger classes with more than 400 samples (cf. lower figure 32) there is only a very light correlation of $R^2 = 0.04$. This is probably simply because a larger standard distance correlates with a larger class size. Thus, the actual correlation is not so much the standard distance, but rather the quantity of samples.

The correlation for smaller classes, with less than 400 samples, is reversed: $R^2 = -0{,}18$, i.e. the smaller the standard distance, the better the F1 - score. The reason for this is an easier feature detection of clustered small classes. The other seeds and herbs class, for example, exists only in two small clusters, as shown by the green field point in figure 33. Such a small class is normally harder to detect because the NN has less samples to learn its patterns. However, if all (or most) samples of such a small class share very similar and strong local patterns it would help to distinguish this class from other small classes which are much more spread out. Since the fields are clustered most input fields are taken from a small area. Therefore, nearly all field polygons of such a clustered class share exactly the same observation conditions, e.g. cloud coverage. This imprints the observation conditions as distinct patterns within the class patterns detected by the NN. Such local patterns, which are actually not part of the crop's features, could be cloud coverage or cloud shadows. Thus, the NN detects these clustered and small classes more based on the sequence of local patterns, than the actual crop features. An evenly distributed class, in contrast, contains different observation conditions, because the samples are taken from a large area. The influence of such characteristic local patterns on the classification accuracy will be analysed further in the next chapter.

In conclusion, this chapter showed: The absolute position of the input data has no general influence on the classification results of the NN. The relative position of the input data only influences the results for smaller classes, so that small classes with a low standard distance yield slightly better results.

8.3.3. Influence of features

In this chapter the influence of the distinctiveness of the feature is analysed. "*Variation between classes is necessary for the DL models to be able to differentiate features and characteristics, and perform accurate classifications.*" (Kamilaris and Prenafeta-Boldú, 2018, p.73) The features of the input data are the average pixel values of the different Sentinel 2 bands and the sequential progress. Thus, the sequential pixel values represent the crop's characteristic at a certain moment in time and the development during the crop's growth cycle. Some crops share very similar features, i.e. similar growth development, colours, textures, pattern etc., while others are very distinct. Obviously, crops with very similar features are much harder to differentiate, than those with clearly distinct features. As mentioned in the previous chapter 8.3.2 - Influence of geography, another reason why a crop class might display very distinct features is due to clustered training data, which results in local pattern being learned by the NN.

To analyse the features of a crop type and the differences between two different crops, the input data received by the NN was visualised in an image, here called: crop maps. Such a crop map is generated by preparing all training data as described in chapter 6 - Data preprocessing with python. The final normalised 3d vector of one crop type consists of 3 axis: First, the Sentinel 2 bands second, the time steps and third, the individual samples pixel values. By averaging the 3d vector along the third axis, i.e. the pixel values, a 2d mean vector over all samples is created. This 2d vector contains the mean values of all samples and thus the general pattern as it is processed by the NN to detect a class. Such a vector can be visualised with a Python library, such as "mathplotlib" ("MATHPLOTLIB") to create a crop map. Figure 34 shows a crop map of winter wheat. The date of the observation is recorded on the vertical axis, whereas the horizontal axis shows the 14 used Sentinel 2 bands (cf. table 4 and 5). The input date column has been removed as it is simply a more or less regular sequentially increasing number. Since the data was normalised within each band (cf. chapter 6.3.6) all values are between 0 and 1. A darker area thus represents a higher and a lighter area a lower pixel value of the original Sentinel 2 band. The repeating horizontal darker bands show situations with a higher cloud coverage over all or most of the survey area. Since it occurred over all (or most) of the survey area the cloud coverage is slightly imprinted into the mean values of all winter wheat fields.

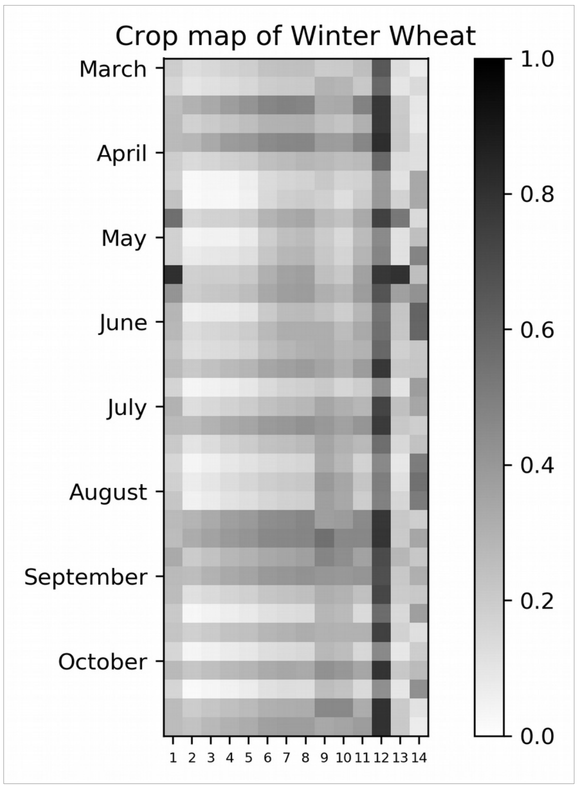


Figure 34: Crop map of Winter Wheat (own figure).

To visualise the distinctness between two crop types, two crop map vectors are subtracted from each other and the absolute difference value is visualised. As an example base crop, to compare other crop types against, winter wheat has been selected because its class size is large enough to not be affected by local patterns and it is very similar to some crop types and at the same time very distinct from others.

The following three different situations are presented in figure 35:

1. Winter Wheat vs. Maize, as an example of crop types with a high features variation and without a single misclassification between each other during the classification of the evaluation sample (cf. confusion matrix - table 12).
2. Winter Wheat vs. Winter Barley, as an example of two crop types with very similar features and a lot of mix-up in both directions, i.e. winter wheat fields were predicted as winter barley and vice versa (cf. confusion matrix - table 12).
3. Winter Wheat vs. Other Seeds and Herbs, as an example of a crop type with a strong local pattern, as described in chapter 8.3.2 - Influence of geography.

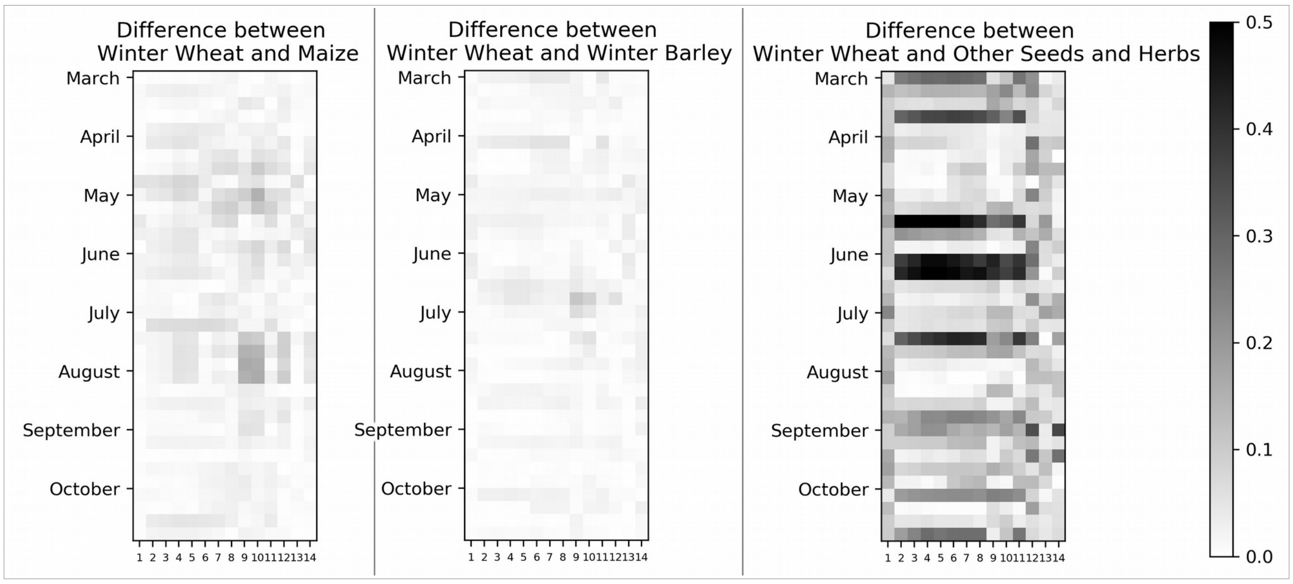


Figure 35: Feature difference between winter wheat and selected crops (own figure).

The higher the absolute mean difference between two crop classes the darker the area in figure 33 and the higher the feature variation. The visualisation of the difference between winter wheat and maize displays a clear difference in April until the beginning of May for almost all Sentinel 2 bands. This is the growth phase, when winter wheat is already growing but maize hasn't been sown or grown substantially, yet. In June until the beginning of July, there are less differences visible, as both plants start to grow and to cover the whole field. From mid of July until mid of August a clear difference, especially in the Sentinel 2 bands 9 and 10, is visible. During this time winter wheat is already harvested while maize is still growing on the field. During August and September, the difference is becoming less and less as winter wheat field might be regrown with catch crops and maize might already be getting harvested due to the very dry year of 2018. When comparing winter wheat and winter barley the lack of feature variation compared to winter wheat and maize becomes immediately evident. As winter wheat and winter barley share very similar seeding, growth and development phases, there is much less difference between them as the almost white visualisation shows. Thus, to differentiate between those two crops is obviously much harder than between winter wheat and maize. Therefore, there are multiple mixed-up samples between winter wheat and winter barley, whereas there are no such confusions between winter wheat and maize (cf. confusion matrix table 12).

Lastly, the difference between winter wheat and other seeds and herbs in figure 35 demonstrates the effects of local patterns. The input data of the other seeds and herbs class is clustered, as described in chapter 8.3.2 - Influence of geography. This results in strong local patterns and variations, i.e. the dark horizontal bands. These dark bands (cf. left crop map figure 35) represent cloud coverage which covers the whole cluster due to its small expansion. The area of the other seeds and herbs cluster (cf. figure 33) is 100% cloud covered on all available Sentinel 2 images from the 08.06.2018 to the 24.06.2018. Thus, the good classification accuracy of the other seeds and herbs class (cf. table 13) despite the very few training samples, can be explained by its distinct features due to local patterns.

In conclusion, as it is more difficult for humans to distinguish similar looking objects, it is similar for NNs. The differences between the crop maps showed that classes with distinct or unique feature variations, due to their growth cycle or due to local pattern, are easier to differentiate, are less mixed-up and thus have better classification results.

8.3.4. Influence of quantity

The discussion of the influence of the field size and geography in chapter 8.3.1 and 8.3.2, gave an indication that the quantity of samples within a class has an influence on the classification accuracy for that class. To test this hypothesis the class frequency was set in relation to the F1-score. Figure 36 displays the result in relation to the standard distance (cf. chapter 8.3.2). The y-axis returns the F1-score, the logarithmic x-axis the number of samples within that class and the size of the class point represents the size of the standard distance. The distribution of the circles shows a logarithmic positive correlation, i.e. a larger quantity of samples results in a better F1-score. The logarithmic coefficient of determination is $R^2 = 0.14$ over all samples. In the top-left area of figure 36 three data points are marked red. These are the three classes with significant lower standard distances than all other classes. These three classes are other seeds and herbs, sugar beet and winter rye. The standard distances of these three classes are 30 794, 35 628 and 40 147, while the standard distance of the other 13 classes ranges from 48 937 to 57 836 (cf. table 14), thus there is a clear break for these three classes. Furthermore, all three classes have fewer than 100 samples in the training-validation dataset (cf. table 7).

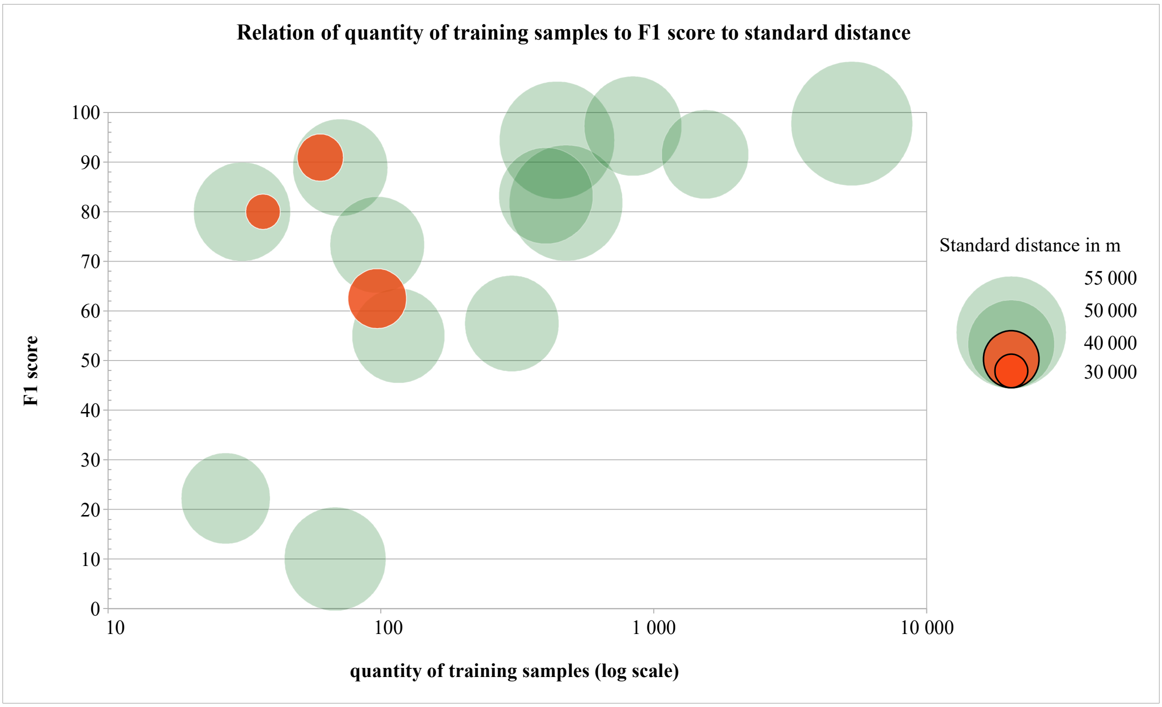


Figure 36: Relation of quantity of training samples to F1-score to standard distance (own figure).

Considering all these factors, there is a strong indication that theses classes are influenced by local patterns as described in chapter 8.3.2 - Influence of geography and 8.3.3 - Influence of features. The F1-score of these three classes is thus less influenced by their quantity, but rather by the location and distribution of the fields. To reduce this wrongly correlated influence and to get a more accurate number of the actual correlation between the sample size of the training data and the classification result, the coefficient of determination was calculated again without the classes other seeds and herbs, sugar beet and winter rye. The results are visible in figure 37. The R^2 value changes to 0.39. This demonstrates the strong correlation between the quantity of samples and the classification accuracy. It has to be noted that the correlation is calculated on a logarithmic scale, as visible by the logarithmic x-axis in figure 36 and 37. This means, to improve the F1-score, the number of samples in a class has to be increased exponentially. Furthermore, figure 37 also displays the strong variance of the F1-score of classes with fewer than 100 samples, showing that the results from these classes are less stable.

In conclusion, the quantity of training samples per class has a very strong influence on a class' classification accuracy. Classes with less than 100 samples have to have very distinct features or they will produce bad or unstable accuracy results. As a rough overview, based on the input data and LSTM used in this thesis, the following quantity of samples is needed to produce reliable classification results: 400 to 800 samples yield a F1-score of approximately 80 to 90%, 800 up to 5 000 samples in a class are needed to get a class F1-score of 90 to 95%, while 5 000 and more samples deliver an accuracy of 95% and higher. These numbers are of course only rough estimates and are – as shown in the previous chapter – influenced by multiple factors. Still, the following rough conclusion can be drawn: To get stable, reliable results, thousands, preferably 5 000 or more samples of each class are needed.

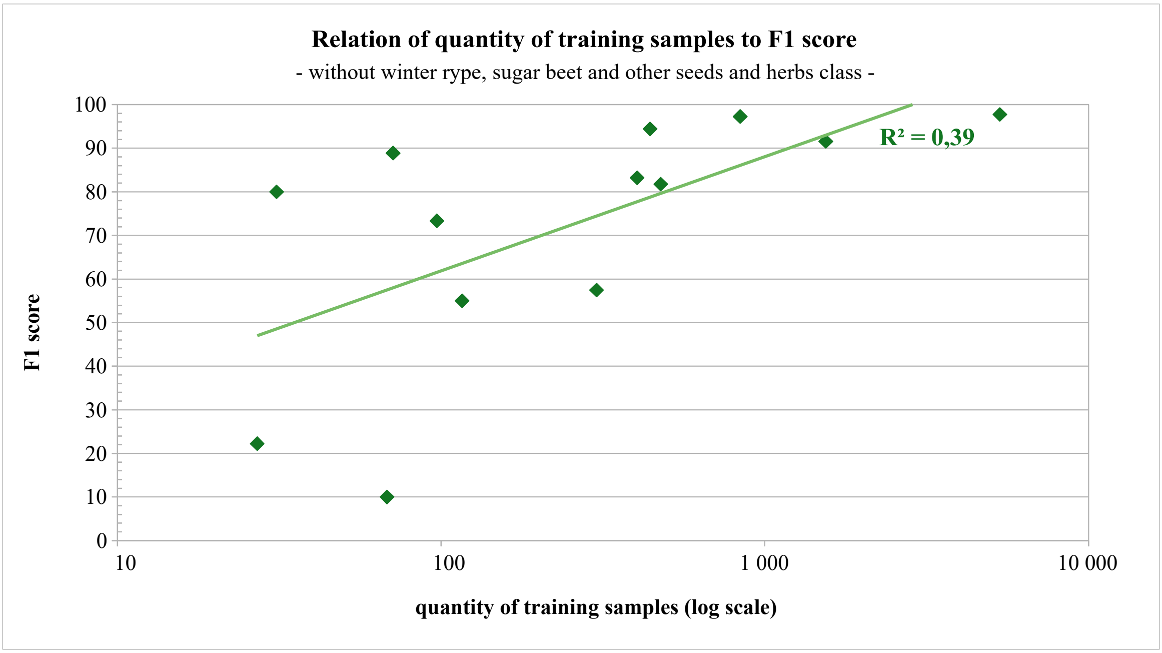


Figure 37: Relation of quantity of selected training samples to F1-score (own figure).

9. Discussion

This chapter takes a look a the results and discusses the research questions posed in chapter 1.2. The evaluation of the LSTM model showed that is is possible to use multi-temporal and multi-dimensional Sentinel 2 images as an input for a NN and to successfully classify different crop types. The developed network achieved an overall accuracy of 92.36% on an independent dataset. The close range of the k-fold splits training accuracies, ranging from 93.16% to 94.16%, with a mean of 93.57% and a standard deviation of only 0.003 corresponds with the accuracy rate of the evaluation dataset. This demonstrated that the model is not overfitting. An accuracy of over 90% indicates a generally good performance, comparable with other approaches. In their summary study Kamilaris and Prenafeta-Boldú, 2018 find that: *"In 19 out of the 24 papers that involved CA [Classification Accuracy] as a metric, accuracy was high (i.e. above 90%), indicating good performance."* (Kamilaris and Prenafeta-Boldú, 2018, p.74). When comparing the accuracy of this thesis to other studies, two factors have to be considered:

1. In this study, the input data was taken from polygons. Thus, the borders of the input objects are not detected by the network. This improves the accuracy because the network does not have to detect border pixels which often contain values from different actual land covers and are harder to classify correctly.
2. Most of the studies include cover classes (such as water, forest, snow, sealed soil etc.) into their overall classification accuracy. These cover classes are usually comparatively easy to recognize (Rußwurm and Körner, 2017) and thus achieve a very high accuracy value of 98-99%. Thus, when comparing the results of this thesis to other studies, it has to be noted that no cover classes are classified in this approach, which would increase the general mean accuracy.

Therefore, considering these two factors, a comparison of the classification accuracy with other studies can only provide a limited rating of the models quality compared to others. Two studies with similar LSTM networks and classification tasks are: Ienco, D. et al. (2017) and Rußwurm and Körner (2017). *Ienco, D. et al.* achieve an overall accuracy of ca. 86.2% (Ienco, D. et al., 2017, p.1688). Rußwurm and Körner list the different types of accuracies they achieved explicitly. With their LSTM model they achieve a general accuracy of 84.4, for cover classes 98.5 and for crop classes 76.2 (Rußwurm and Körner, 2017, p.556). These numbers show that the approach developed in this thesis generally delivers comparable results and presents a way to classify crops with high accuracy.

The thesis also tried to answer questions of how to differentiate between very similar classes (cf. chapter 1.2). This challenge also occurred in other studies: *"Further chance for confusion was observed in the case of classes with* [sic] *are botanically related to each other and thus share similar spectral and temporal features. For instance, the classes triticale, wheat, and rye have been commonly confused, as triticale is a hybrid of the latter two classes."* (Rußwurm and Körner, 2017, p.555). The developed LSTM model also confused classes, especially those with very similar features. The confusion matrix (cf. table 12) presented all cases of mixed-up classes. Chapter 8.3.3 - Influence of featuresexplained how similar the features of some classes are and how this affects the accuracy. Even though there were class confusions, the individual class results in table 13 and the high AUC score show that the developed approach, given enough training samples, can differentiate classes with very similar features. An example of this are the precision scores of the very similar classes: winter wheat (89%), winter barley (87%), winter rye (85%) and triticale (67%).

Further influence of the general and spatial characteristics of the input data was analysed in chapter 8.3 Evaluation of miss-classification. Subchapters 8.3.1 and 8.3.2 presented the influence of the spatial properties: area size and absolute and relative position of the input data. The area size only influenced classes with a low number of samples, while classes with a larger sample size were not influenced. In addition, the absolute position didn't influence the classification. The relative position, however, did influence the prediction results. Clustered classes can worsen but also improve accuracy due to local patterns. These local patterns can create misleading recognisable features, e.g. due to cloud coverage, which are then wrongly attributed to the clustered class. This emphasises the importance of well distributed training samples. Furthermore, the strong correlation between the quantity of samples of one class and its classification accuracy was demonstrated on the individual class accuracy results of the evaluation dataset. This stresses the importance of at least 1 000, preferably more than 5 000 samples per class, to achieve good and reliable results.

The Chapter "6.3.4- Reduction of low information data within the time dimension" answered the third research question (cf. chapter 1.2). A time window of 4 consecutive Sentinel 2 images was used to search for the best available pixel values. With this approach different scenes for individual fields can be used and thus more valid and actual ground pixel values can be extracted from the available Sentinel 2 raster images. This resulted in a better and much faster classification of input data in contrast to using all Sentinel 2 scenes.

Lastly, with the developed approach it is possible to classify vast areas using raster data, labelled polygon-datasets and consumer-grade hardware (cf. chapter 1.2, 2.2 and 3.3). Whereas an area-wide classification requires a hard to obtain, pixel-accurate labelled y dataset, the developed approach requires only an easy to create or already availability labelled y polygon-dataset and, therefore, simplifies the data preprocessing of aerial images for NNs substantially. The thesis demonstrated how the average pixel values, generated from the field polygons and the Sentinel 2 raster data, can be used as input x and the field polygon's crop types as y data. A LSTM using this type of input data will classify input polygons with an acceptable general and class-specific accuracy. Furthermore, by using only polygons as input data the extent of the whole free state of Thuringia could easily be trained by the network in ca. 2 hours. Therefore, even larger data samples could be trained within a reasonable time. Assuming linear scaling and a similar dataset (considering the number of objects per km^2), a LSTM model for whole Germany (approx. 22 times the size of Thuringia), with approx. 220 000 data samples, could be trained in approx. 44 hours.

In summary, the results of the data preprocessing, the development and training of the LSTM and the evaluation of the test dataset, described in chapters 6 to 8, answered the primary and secondary research questions (cf. chapter 1.2) on a practical level and verified the advantages of this approach, stated in chapter 1.3.

10. Conclusion

This thesis presented an approach to classify pre-selected objects on Sentinel 2 images with a LSTM. The network proved to reliably predict an independent test sample with an acceptable general and individual class accuracy. The thesis thus demonstrated a possible way to classify crops on multi-spectral and multi-temporal Sentinel 2 images using a LSTM. Several points influencing the accuracy have been discussed. The most influential point is the quantity of class samples. The main advantages of the developed approach are the good accuracy, even with similar classes, the simple structure of the y data and the low training time, which allows a rapid classification of a vast area.

The overall accuracy and robustness of the LSTM network could be further improved with the following steps:

1. Increased class samples: As the quantity of samples influence the class accuracies (cf. chapter 8.3.4), increasing each class to a minimum amount of 1 000 samples (preferably even more) would certainly improve the overall accuracy.

2. Well distributed classes: Since the two clustered classes influenced the detection of the actual class features, better distributed classes would further increase the model's robustness, against local and/or temporal anomalies and improve its prediction capabilities of new unseen data.

3. Increase area extent of samples: By using a larger area (e.g. data samples from all over Germany) an increased amount of feature patterns for each crop type would be available. This could contribute to a greater geographic area the network can be used on and also to a more stable classification accuracy because local influences are less relevant.

4. Use multiple years as input data: The year 2018, for example, was an extremely dry year, which resulted in a specific situation for crop growth and harvest ("BMEL – Harvest Report"). By using input data from the growth cycle from multiple years such temporal anomalies can be balanced. Thus, the LSTM would be more robust to classify crops in different circumstances.

5. Use of additional input data: Other data sources that observe the fields in a different way add a new information layer to provide information about the crop types. An example of this is

Sentinel 1 data. Adding radar data would provide a new view on the crop fields and thus might increase the accuracy of the LSTM model.

Besides changes in the input data, the architecture of the NN itself could be changed to improve classification quality. For the developed approach pre-selected polygons are required, these provide clear borders for all objects. The developed LSTM model could also be used to classify whole aerial images, by creating an equivalent table, not from the field polygons, but from the individual pixels. This would, however, increase the complexity of the approach because each pixel has to be classified individually and a pixel-accurate labelled y dataset is required. Furthermore, a weak point of such an approach would be that it does not consider contextual spatial information. To develop a network which considers contextual spatial and sequential temporal information is a topic of current and future research. One possibility, as proposed by Rußwurm and Körner, is *"[...] a CNN encoder prepended to the LSTM network [...], as richer textural features would be extracted in a perceptive field optimally chosen by the network."* (Rußwurm and Körner, 2017, p.557). Another possibility, proposed by Pelletier et al., is to use a temporal CNNs. This DL approach *"[...] applies convolutions in the temporal dimension in order to automatically learn temporal (and spectral) features."* (Pelletier et al., 2019, p.1).

Whatever approach will prove to be the best for future classifications, it is certain that multi-spectral and multi-temporal sequential information will become more important for the classification of aerial images. Since the world and all its features have a multi-spectral and multi-temporal nature, as much of these information layers as possible have to be used to achieve the most accurate classification results. The growing availability of such spatial data, one of the most prominent being the Sentinel 2 satellite images, increases the possibilities for such image analysis. Thus, future DL networks suitable for the classification of aerial images need to improve their abilities to understand the spectral and temporal dimension of SITS.

11. Literature

Cresson, R., 2018. A framework for remote sensing images processing using deep learning technique. CoRR abs/1807.06535.

Drusch, M., Del Bello, U., Carlier, S., Colin, O., Fernandez, V., Gascon, F., Hoersch, B., Isola, C., Laberinti, P., Martimort, P., Meygret, A., Spoto, F., Sy, O., Marchese, F., Bargellini, P., 2012. Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services. Remote Sensing of Environment Vol. 120, pp 25-36.

ESA, 2015. Sentinel-2 User Handbook (User Guide No. 1.2). European Space Agency (ESA), pp 48-60
online available at: https://sentinel.esa.int/documents/247904/685211/Sentinel-2_User_Handbook

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning, Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts. Pp 1-26, 96-133, 164-200, 224-265, 367-390

Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural computation 9, pp 1735–1780.

Ienco, D., Gaetano, R., Dupaquier, C., Maurel, P., 2017. Land Cover Classification via Multitemporal Spatial Data by Deep Recurrent Neural Networks. IEEE Geoscience and Remote Sensing Letters 14, pp 1685-1689.

Kamilaris, A., Prenafeta-Boldú, F.X., 2018. Deep learning in agriculture: A survey. Computers and Electronics in Agriculture Vol. 147, pp 70–90.

Kussul, N., Lavreniuk, M., Skakun, S., Shelestov, A., 2017. Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data. IEEE Geoscience and Remote Sensing Letters Vol. 14(5), pp 778-782.

Mitchell, T.M., 1997. Machine Learning, McGraw-Hill series in computer science. McGraw-Hill, New York, p. 2

Mou, L., Ghamisi, P., Zhu, X.X., 2017. Deep Recurrent Neural Networks for Hyperspectral Image Classification. IEEE Transactions on Geoscience and Remote Sensing 55, pp3639–3655.

Mueller-Wilm, U., 2018. Sen2Cor Configuration and User Manual (User Guide Issue 2). European Space Agency (ESA).

Nielsen, M.A., 2015. Neural Networks and Deep Learning. Determination Press. Web-book: http://neuralnetworksanddeeplearning.com/index.html (accessed 28.07.2019)

Pelletier, C., Webb, I.G., Petitjean, F., 2019. Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series. Remote Sensing Vol. 11- Issue 5

Petitjean, F., Kurtz, C., Passat, N., Gançarski, P., 2012. Spatio-temporal reasoning for the classification of satellite image time series. Pattern Recognition Letters 33,pp 1805–1815.

Raschka, S., Mirjalili, V., 2018. Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow, Second edition, fourth release, Expert insight. Packt Publishing, Birmingham Mumbai. pp 1-13, 55, 17-19, 120-123, 189-215, 380-450, 539-550

Rosenblatt, F. The Perceptron: A Perceiving and Recognizing Automaton, Cornell Aeronautical Laboratory, 1957), pp 1-3

Rußwurm, M., Körner, M., 2017. Multi-temporal Land Cover Classification with Long Short-term Memory Neural Networks. ISPRS International Journal of Geo-Information 7, pp 551-558.

12. Internet Sources

BMEL – Harvest Report, Bundesministerium für Ernährung und Landwirtschaft - Erntebericht 2018 *https://www.bmel.de/DE/Landwirtschaft/Pflanzenbau/Ackerbau/_Texte/Ernte2018.html* (accessed 03.10.2019 - website only available in german language).

Copernicus-Open-Access-Hub, ESA Copernicus Open Access Hub, https://scihub.copernicus.eu/dhus/#/home (accesses 03.10.2019)

Dev-google-machine-learning: Google Developer Crash Course: Classification: ROC Curve and AUC, *https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc* (accessed 03.10.2019)

Gondaliya, A, 2014. Regularization implementation in R: bias and variance diagnosis. http://pingax.com/regularization-implementation-r/ (accessed 03.10.2019).

NumPy: NumPy package for scientific computing with Python *https://numpy.org/* (accessed 03.10.2019)

NP-API-Res, SciPy.org Numpy-API: *https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html* (accessed 03.10.2019)

NP-API-where, SciPy.org Numpy-API: *https://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html* (accessed 03.10.2019)

GDAL-S2: Geospatial Data Abstraction Library, Overview of Sentinel-2 Products, *https://www.gdal.org/frmt_sentinel2.html* (accessed 03.10.2019)

GDAL–API: Geospatial Data Abstraction Library, Build of Virtual Datasets, *https://gdal.org/gdalbuildvrt.html* (accessed 03.10.2019)

Karpathy, Andrej, 2015. The Unreasonable Effectiveness of Recurrent Neural Networks. Hacker's guide to Neural Networks,
http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (accessed 03.10.2019).

KERAS, Keras: The Python Deep Learning library,
https://keras.io/ (accessed 03.10.2019)

KERAS – API – AboutModels, About Keras models
https://keras.io/models/about-keras-models/ (accessed 03.10.2019)

KERAS – API – SequentialGuide, Getting started with the Keras Sequential model
https://keras.io/getting-started/sequential-model-guide/ (accessed 03.10.2019)

KERAS – API – Sequential, The Sequential model API,
https://keras.io/models/sequential/ (accessed 03.10.2019)

KERAS – API – Optimizers, Usage of optimizers,
https://keras.io/optimizers/ (accessed 03.10.2019)

MATHPLOTLIB, Python 2D plotting library,
https://matplotlib.org/ (accessed 03.10.2019)

Olah, C., 2015. Understanding LSTM Networks. colah's blog.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed 03.10.2019).

Pandas, Pandas - Python Data Analysis Library,
https://pandas.pydata.org/index.html (accessed 03.10.2019)

Python – SF: Python Software Foundation,
https://www.python.org/ (accessed 03.10.2019)

Python – Datetime: Python Software Foundation - Python 3.7.4 documentation, Datetime https://docs.python.org/3/library/datetime.html (accessed 03.10.2019)

QGIS – API: QGIS API Documentation of QgsZonalStatistics Class Reference, *https://qgis.org/api/classQgsZonalStatistics.html* (accessed 03.10.2019)

SCIPY-API-Interpolate, SciPy.org Scipy-API: *https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html* (accessed 03.10.2019)

SCIKIT-LEARN: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011, *https://scikit-learn.org/stable/index.html (*(accessed 03.10.2019)

SCIKIT-LEARN-API-LabelEncoder, https://scikit-learn.org/ - sklearn API, *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html* (accessed 03.10.2019)

SCIKIT-LEARN-API-MinMaxScaler, https://scikit-learn.org/ - sklearn API, *https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html* (accessed 03.10.2019)

SCIKIT-LEARN-API-Metrics*,* https://scikit-learn.org/ - sklearn API, *https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics* (accessed 03.10.2019)

SCIKIT-LEARN-API-StratifiedKFold*,* https://scikit-learn.org/ - sklearn API, *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html* (accessed 03.10.2019)

S2-L2A Overview: ESA – Sentinel Online Technical Guide - Level-2A Algorithm Overview, *https://earth.esa.int/web/sentinel/technical-guides/sentinel-2-msi/level-2a/algorithm* (accessed 03.10.2019)

S2-oUG: Sentinel 2 - Online User Guide
https://sentinel.esa.int/web/sentinel/user-guides (accessed 03.10.2019)

S2-oUG – Definitions: Sentinel 2 - Online User Guide – Definitions,
https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/definitions (accessed 03.10.2019)

S2-oUG - Product Naming Convention:
https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/naming-convention (accessed 03.10.2019)

TDS-AUC-ROC-Curve, Towards Data Science: AUC – ROC Curve,
https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (accessed 03.10.2019)

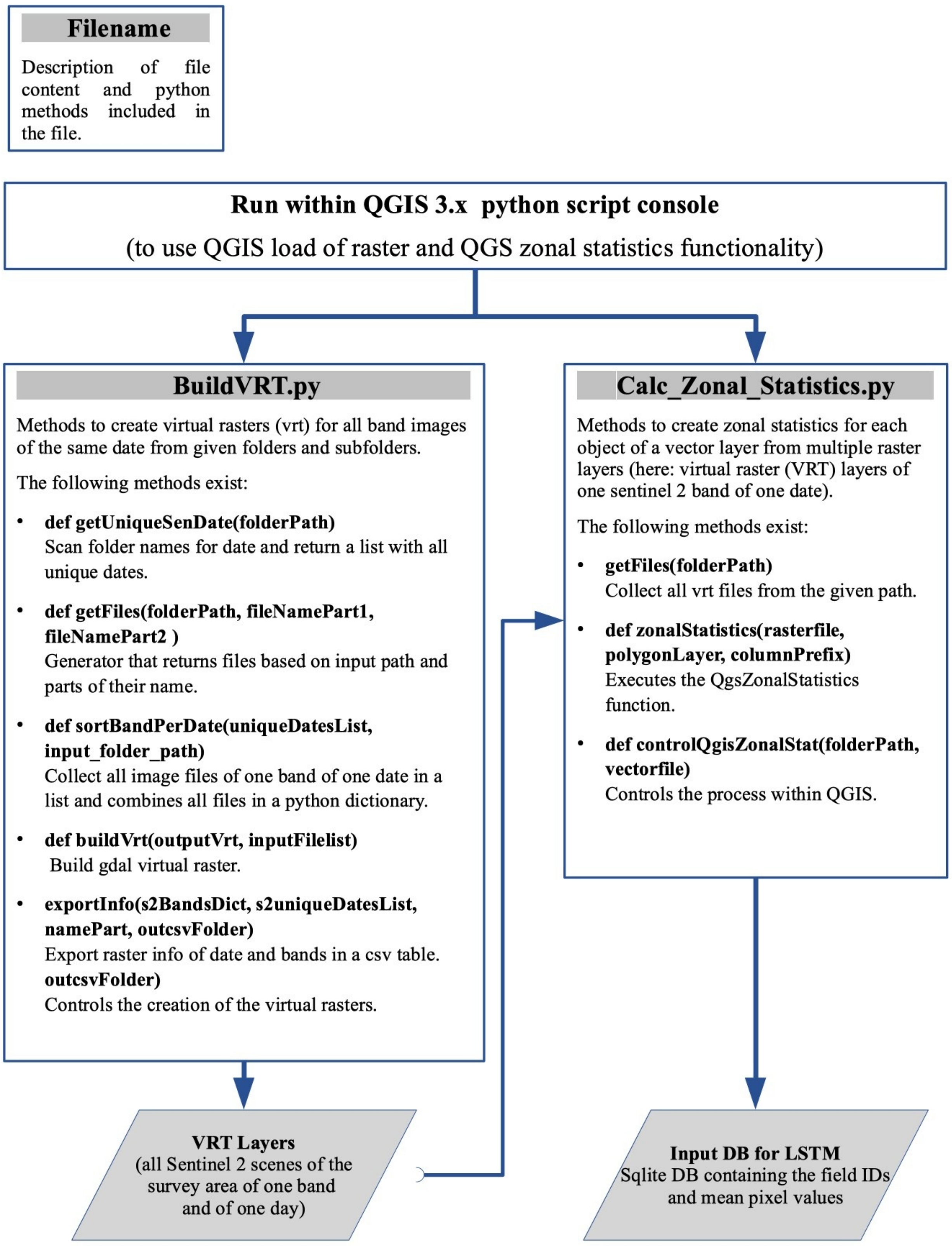
Wiki-AI-winter, Wikipedia: AI winter,
https://en.wikipedia.org/wiki/AI_winter (accessed 03.10.2019)

Wikimedia-ANN-Model, WIKIMEDIA – ArtificialNeuronModel,
https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png (accessed 03.10.2019)

Annex I: Overview of the developed python scripts and their structure

The following diagrams show an overview of the different Python methods used for data preprocessing, the training and evaluation of the LSTM. In addition to these descriptions, the actual python files include detailed comments on the code (see Annex II). Some general data processes which are not specific for this thesis, such as the manipulation of pandas data frames or the merging of csv files, were implemented directly on the data with the python console and are not described here.

Rasterdata pre-processing scripts to create input table



Filename

Description of file content and python methods included in the file.

Run within QGIS 3.x python script console

(to use QGIS load of raster and QGS zonal statistics functionality)

BuildVRT.py

Methods to create virtual rasters (vrt) for all band images of the same date from given folders and subfolders.

The following methods exist:

- **def getUniqueSenDate(folderPath)**
  Scan folder names for date and return a list with all unique dates.
- **def getFiles(folderPath, fileNamePart1, fileNamePart2)**
  Generator that returns files based on input path and parts of their name.
- **def sortBandPerDate(uniqueDatesList, input_folder_path)**
  Collect all image files of one band of one date in a list and combines all files in a python dictionary.
- **def buildVrt(outputVrt, inputFilelist)**
  Build gdal virtual raster.
- **exportInfo(s2BandsDict, s2uniqueDatesList, namePart, outcsvFolder)**
  Export raster info of date and bands in a csv table.
  outcsvFolder)
  Controls the creation of the virtual rasters.

Calc_Zonal_Statistics.py

Methods to create zonal statistics for each object of a vector layer from multiple raster layers (here: virtual raster (VRT) layers of one sentinel 2 band of one date).

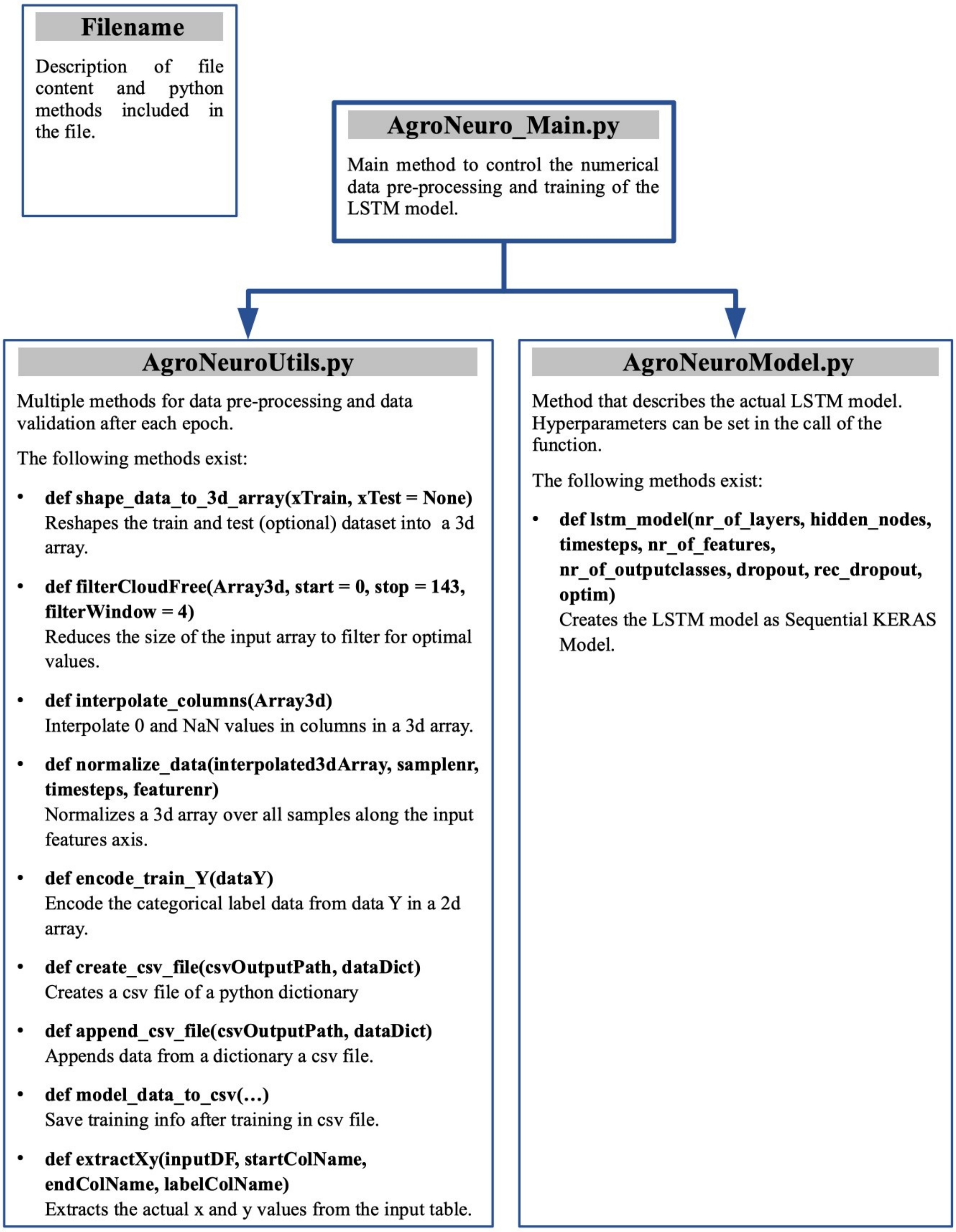
The following methods exist:

- **getFiles(folderPath)**
  Collect all vrt files from the given path.
- **def zonalStatistics(rasterfile, polygonLayer, columnPrefix)**
  Executes the QgsZonalStatistics function.
- **def controlQgisZonalStat(folderPath, vectorfile)**
  Controls the process within QGIS.

VRT Layers
(all Sentinel 2 scenes of the survey area of one band and of one day)

Input DB for LSTM
Sqlite DB containing the field IDs and mean pixel values

LSTM numerical data pre-processing and training scripts



Filename

Description of file content and python methods included in the file.

AgroNeuro_Main.py

Main method to control the numerical data pre-processing and training of the LSTM model.

AgroNeuroUtils.py

Multiple methods for data pre-processing and data validation after each epoch.

The following methods exist:

- **def shape_data_to_3d_array(xTrain, xTest = None)**
  Reshapes the train and test (optional) dataset into a 3d array.
- **def filterCloudFree(Array3d, start = 0, stop = 143, filterWindow = 4)**
  Reduces the size of the input array to filter for optimal values.
- **def interpolate_columns(Array3d)**
  Interpolate 0 and NaN values in columns in a 3d array.
- **def normalize_data(interpolated3dArray, samplenr, timesteps, featurenr)**
  Normalizes a 3d array over all samples along the input features axis.
- **def encode_train_Y(dataY)**
  Encode the categorical label data from data Y in a 2d array.
- **def create_csv_file(csvOutputPath, dataDict)**
  Creates a csv file of a python dictionary
- **def append_csv_file(csvOutputPath, dataDict)**
  Appends data from a dictionary a csv file.
- **def model_data_to_csv(…)**
  Save training info after training in csv file.
- **def extractXy(inputDF, startColName, endColName, labelColName)**
  Extracts the actual x and y values from the input table.

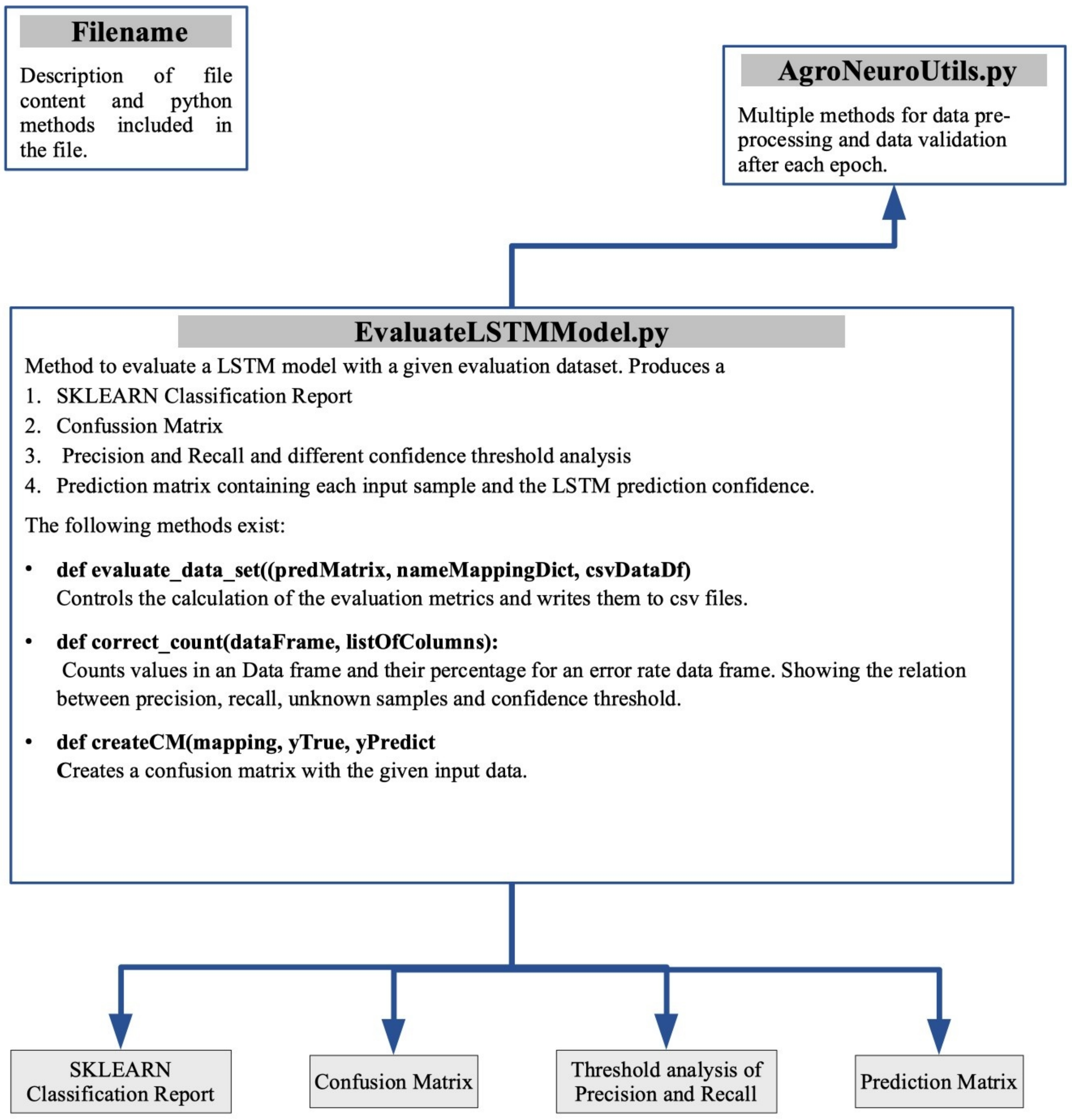
AgroNeuroModel.py

Method that describes the actual LSTM model. Hyperparameters can be set in the call of the function.

The following methods exist:

- **def lstm_model(nr_of_layers, hidden_nodes, timesteps, nr_of_features, nr_of_outputclasses, dropout, rec_dropout, optim)**
  Creates the LSTM model as Sequential KERAS Model.

LSTM Model evaluation



Filename

Description of file content and python methods included in the file.

AgroNeuroUtils.py

Multiple methods for data pre-processing and data validation after each epoch.

EvaluateLSTMModel.py

Method to evaluate a LSTM model with a given evaluation dataset. Produces a

1. SKLEARN Classification Report
2. Confussion Matrix
3. Precision and Recall and different confidence threshold analysis
4. Prediction matrix containing each input sample and the LSTM prediction confidence.

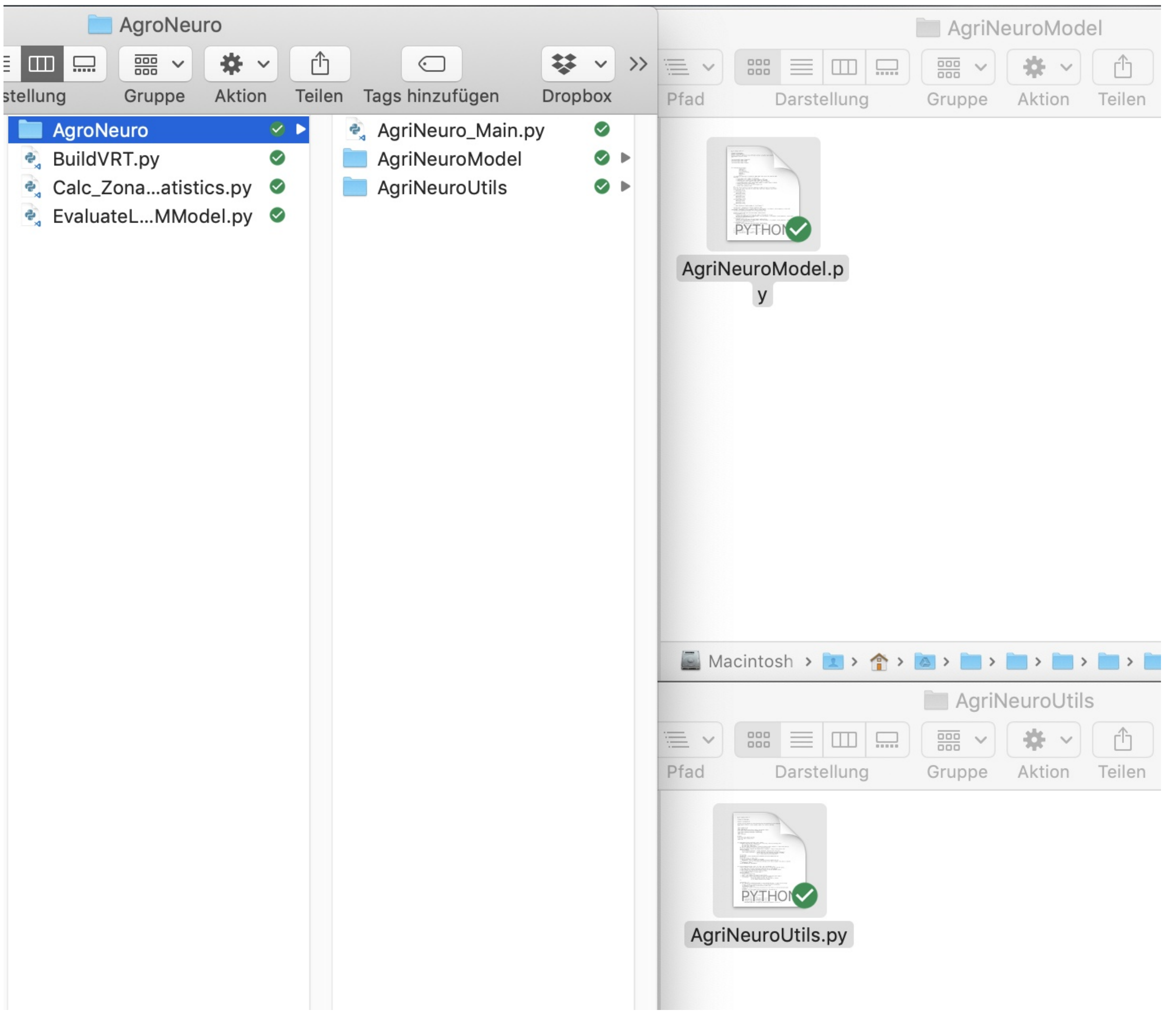
The following methods exist:

- **def evaluate_data_set((predMatrix, nameMappingDict, csvDataDf)**
  Controls the calculation of the evaluation metrics and writes them to csv files.
- **def correct_count(dataFrame, listOfColumns):**
  Counts values in an Data frame and their percentage for an error rate data frame. Showing the relation between precision, recall, unknown samples and confidence threshold.
- **def createCM(mapping, yTrue, yPredict**
  Creates a confusion matrix with the given input data.

SKLEARN Classification Report
Confusion Matrix
Threshold analysis of Precision and Recall
Prediction Matrix

Annex II: Python files used for data preprocessing, training and evaluation of the LSTM

The python files developed for this thesis are attached digitally on memory stick to this thesis. The general file structure is described in the following figure and in ANNEX I.



File structure of the developed python methods (source: own figure).