



Technical University of Munich
Department of Civil, Geo and Environmental Engineering
Chair of Geoinformatics
Univ.-Prof. Dr. rer. nat. Thomas H. Kolbe

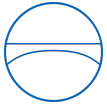
Wind throw detection using deep learning on PlanetScope and high-resolution aerial images

Wolfgang Deigele
Master's thesis

Program: Geodesy and Geoinformation (Master)

Supervisors: Univ.-Prof. Dr. rer. nat. Thomas H. Kolbe
Dr. Melanie Brandmeier
M. Sc. Bruno Willenborg

2020



Involved Organizations



Science and Education Department
Esri Deutschland
Ringstraße 7
D-85402 Kranzberg



Bayerische Landesanstalt für Wald und Forstwirtschaft
Hans Carl-von-Carlowitz-Platz 1
D-85354 Freising

Declaration

With this statement I declare that I have independently completed this Master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Munich, January 31, 2020

Wolfgang Deigele

Wolfgang Deigele
Riesstraße 64
80993 München
e-Mail: wolfgang.deigele@tum.de

Abstract

Due to climate change, the number of storms that cause damage to forests has been increasing over the past years. Trees knocked over by wind throw are a loss of timber if not harvested in time, therefore the damaged area has to be identified as quickly as possible. Current state of the art forest damage detection methods are mainly based on change detection using a before after comparison as well as manual damage assessment using satellite and aerial images, which takes a great amount of time and effort. In this thesis two custom deep learning models based on the U-Net architecture are trained on satellite and aerial images to predict storm damage in forests. The study area is located in the state of Bavaria in Germany and covers 160 km^2 of forest, fields and villages. PlanetScope satellite images of PlanetLabs Inc. with a resolution of $3 \times 3 \text{ m}$ are rapidly available after storm events and are used for first damage assessment. Aerial ortho images with a resolution of $0.2 \times 0.2 \text{ m}$ are used to get a more accurate prediction if necessary. The achieved accuracy of the models are comparable to current state of the art detection methods when used on similar test data, but is considerably lower when used on a different data set. Methods to improve the transfer ability are discussed and implemented using transfer learning. The models are integrated into ArcGIS Pro for damage prediction on large scale and subsequent processing steps of the results.

Table of Contents

1	Introduction	1
1.1	Damage caused by wind throw	1
1.2	Damage detection using using computer vision	1
1.3	Integration of the damage detection into a Geoinformation System (GIS)	2
1.4	Research questions and tasks	2
2	Background	3
2.1	State of the art damage detection methods	3
2.1.1	Passive damage detection methods	3
2.1.2	Active damage detection methods	4
2.2	Historical background of Deep Learning	5
3	Data	7
3.1	Study area	7
3.2	Satellite images of the study area	8
3.3	Aerial ortho-images of the study area	10
3.4	Labeling of the damage	11
3.5	Transfer ability study using an independent data set	12
4	Methods	15
4.1	Overview of the used methods	15
4.2	Tiling of the input data	17
4.3	Data augmentation of the input	19
4.3.1	Geometric data augmentation	19
4.3.2	Radiometric data augmentation	20
4.4	Used hardware for training	21
4.5	Artificial Neural Networks and Deep Learning	22
4.6	Overview over Convolutional Neural Networks (CNNs)	24
4.7	Working principle of Convolutional Neural Networks	27
4.7.1	Loss function for error determination	27
4.7.2	Back Propagation and weight updating	29
4.8	Model Implementation	29
4.9	Performance evaluation for Convolutional Neural Networks (CNNs)	31
4.9.1	Confusion Matrix	31
4.9.2	Intersection over Union	32
4.9.3	Accuracy	33
4.9.4	Precision and Recall	33
4.9.5	Receiver operating characteristic	33
4.10	Approach to find the optimal model	34
4.10.1	Testing different tile sizes	35
4.10.2	Testing different numbers of blocks and filters	37

4.10.3	Testing different learning rates	39
4.10.4	Testing the effect of additional random noise	41
4.11	Training of the models with the optimal hyperparameters	42
4.12	Application of transfer learning on the data	43
4.13	Integration of the models into a Geoinformation System (GIS)	43
4.14	Post processing using GIS tools	44
5	Results	47
5.1	Evaluation of the models	47
5.1.1	Performance evaluation for satellite images with mixed labels	47
5.1.2	Performance evaluation for ortho images	54
5.1.3	Performance evaluation for transfer learning	58
5.1.4	Test of the impact of partial reduction of the training data set	59
5.2	Evaluation of the prediction in ArcGIS Pro	61
5.2.1	Prediction of models trained on satellite images	62
5.2.2	Prediction of models trained on ortho images	66
5.2.3	Prediction of the transfer learning model	70
5.2.4	Transfer ability study using another data set	71
6	Discussion	74
7	Conclusions & Outlook	77
8	Appendices	79
8.1	Appendix A: Hyperparameters used during training	79
8.2	Appendix B: Intersection over Union and accuracy of the predictions	79
8.3	Appendix C: Summary of U-Net	80
8.4	Appendix D: Code implementation	82
	List of acronyms	83
	List of Figures	84
	List of Tables	86
	List of Equations	87
	References	88

1 Introduction

1.1 Damage caused by wind throw

In the past years the number of storms that cause damage to forests has been increasing due to climate change and its impact on the environment [Dorland et al., 1999]. Especially forests that consist of mono cultures like spruces as they can be found in large parts of the state of Bavaria in Germany are particularly prone to damage, since they do not have the same resistance to rapid wind throw as mixed forest. Around 50% of all loss in timber in Europe is caused by wind throw, in the year 2017 the storm "Kolle" alone is responsible for a loss of 2 million solid cubic meters of wood in Bavaria [Seitz and Straub, 2017]. Trees that are knocked over by wind throw are considered as a loss in timber if not removed in time, therefore a quick and reliable method to detect the damaged area is required. Leaving the dead wood unattended in the forest brings also other risks like an infestation with the bark beetle, a bug that is particularly drawn to fallen trees and can cause massive damage especially in mono cultures of spruces. Since it is unpractical to map the damaged area in the field, storm damage assessment in Germany is mostly done today via aerial and satellite images and manual digitization. Due to limited image resolution and a variety of different scenes in the images this can be a difficult task which takes a great amount of time and effort and can result in errors and low detection accuracy depending on the quality of the data set.

1.2 Damage detection using using computer vision

A rather new trend in computer science with growing popularity is the usage of neural networks and deep learning for various tasks that previously required human interaction. In this thesis it is tested whether storm damaged areas can be autonomously derived from satellite and aerial images by using methods of deep learning. The general idea for this task is to train a Convolutional Neural Network (CNN) on several images containing damaged area in combination with labels showing where the extent of the damage inside each image is. After successful training, the network can be used to predict storm damage in forests via aerial and satellite images likewise. Reference images that contain the same area before the storm are not required, since there is no before after comparison involved in the prediction. This makes it particularly useful for sudden storms events, where no image before the storm is available.

An approach utilizing this method has been done in 2018 at the Environmental Systems Research Institute (ESRI) on aerial ortho photos using a CNN, which achieved a detection rate of 92% and outperformed most existing methods that do not involve deep learning [Hamdi et al., 2019]. Since it takes a considerable amount of time after a storm event until aerial images are available, a faster and preliminary approach is to use satellite images, which can be available rapidly and up to twice a day [PlanetLabs, 2019]. Typically the spatial resolution of satellite images however is much lower than of aerial images and therefore it is much harder to detect the damage accurately. Individual trees can not be identified at all due to the low resolution, so the network has to be trained to identify damaged areas by their general shape and color. With the final model being fully trained, a prediction of the damage can be created in a matter of minutes instead of days or weeks in manual assessment, while simultaneously keeping the required accuracy.

1.3 Integration of the damage detection into a Geoinformation System (GIS)

The term Geoinformation System (GIS) stands for a system that is used to collect, process and store geographical information. This can range from map applications to landscape or city models or other types of data that has a geographical reference. Modern GIS systems allow for example for the combination of images in raster format with polygons in vector format to analyze the data or to create a map. One such system is implemented in the program ArcGIS Pro [Esri, 2019], a software developed by ESRI that is used to create, analyze, compare and manipulate geospatial data. In the last few years ArcGIS Pro extended its functionality in the direction of Deep Learning, since it is an ever growing and promising application field. Usually CNNs are quite complicated to work with, an integration into a GIS software does simplify this procedure. Not only the simpler handling, but also the possibility to apply GIS based functions makes this integration interesting. The predicted storm damage can for instance be converted to polygons and enhanced, for example by removing small artifacts like holes or damage patches with an area below a give threshold. Within the functionality of ArcGIS Pro lies also the possibility to share the storm damage detection application with other users, which makes it easy to deploy whenever needed.

1.4 Research questions and tasks

In this thesis several questions regarding the suitability for the given data set for training a CNN as well as regarding the integration into GIS are looked upon. The main topic of this thesis is to determine whether a limited satellite data set can be used to train a CNN in a way so that it delivers useful predictions of storm damage. The main issues with the available data are the small total size and the low resolution, which both can prove difficult to deal with. A second set of data is available which contains aerial ortho images in a much higher resolution. This second part of the data was already used in a preceding thesis by Zayd Hamdi [Hamdi et al., 2019], but the data is now more refined, and with some alterations in the training process it may be possible to surpass the already achieved accuracy. By using these two data sets combined, a two way approach is possible that includes rapid damage assessment using satellite data and a slower but more accurate method based on aerial data. A successful integration into a modern GIS environment is a crucial step in making the Deep Learning application easily accessible. The capabilities of ArcGIS Pro regarding this integration and further processing steps are to be tested and a ready to use tool to be created that can ideally be used after the next big storm event. Finally an important question is if the implemented methods of detecting storm damage are performing well compared to other state of the art detection methods as well as different Deep Learning model architectures that are trained on much larger data sets. Tests are also performed on an independent test data set that is composed of other scenes that are not included in the study area.

The implemented code together with the ArcGIS Pro toolbox and the results are attached in the appendices B and D.

2 Background

2.1 State of the art damage detection methods

Manual storm damage detection requires in general a great amount of time and effort, and an automation of the process has been a subject to numerous research topics. Different approaches utilize both active and passive detection methods with various kinds of sensors and data. Some examples of storm damage detection methods are presented in the following two chapters.

2.1.1 Passive damage detection methods

A method for autonomous storm damage detection described by [Honkavaara et al., 2013] utilizes change detection in Digital Surface Models (DSMs) from before and after a storm event. The DSMs themselves were constructed via Airborne Laser Scanning (ALS) in conjunction with aerial ortho imagery. The method's effectiveness stands out for larger groups of fallen trees, smaller areas and individual trees prove to be very hard to detect. The test areas consisted of $100 \times 100 \text{ m}^2$ of forest with the objective to determine whether an area is affected by wind throw or not. Overall every single test area that was used for validation was correctly classified as damaged or not-damaged, giving the classification a 100% accuracy on that large test area scale. Figure 1 shows on the left an area that is affected by wind throw with several trees knocked to the ground and thus lowering the DSM height in certain areas. On the right side is the unaffected DSM of the forest before the storm event.

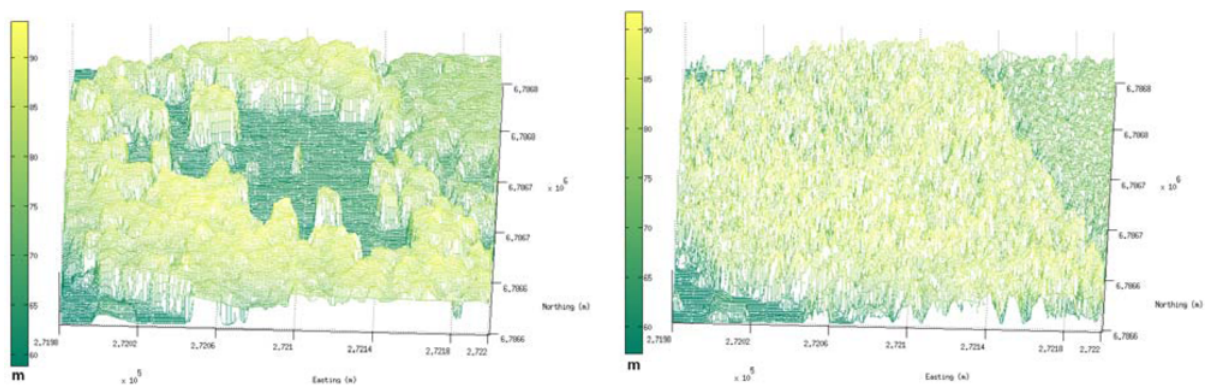


Figure 1: Visible changes in forest height caused by wind throw in a before after comparison of digital surface models used for the detection of storm damage, [Honkavaara et al., 2013]. The DSM on the left is affected by wind throw, the DSM on the right shows the scene before the storm event.

[Chehata et al., 2014] describe a method for object based change detection using high resolution multispectral images. Satellite images from before and after a storm event in the southwest of France with resolutions of 5 m and 10 m were combined in order to determine the damage caused by wind throw. To achieve the best results, not only the degree of damage is take into account, but also how much damage was already present before the storm event. The classification itself is performed unsupervised and bitemporal via a mean shift spectral classification, that determines the grade of damage based on the fragmentation of certain regions. After a clustering, the classes can be separated into damaged and not-damaged by using an automatically computed threshold for the fragmentation rate, which reaches from 0 (no damage) to 1 (damaged). On a pixel based sale with an overall accuracy

of 87.8% correctly classified pixels this method yields better results than another method described by [Duda et al., 2001] with which it was compared. It operates on a purely pixel based detection approach with an overall accuracy of 78.68%

Using Landsat Thematic Mapper data, [Ekstrand, 1996] proposes a method to detect storm damage in rugged terrain in northern Europe. The study area is located in the northern part of Norway with low sun angles that cause particularly long shadows of trees. The data consists of resampled satellite images with a resolution of 25 *m*. Correction models for the sun elevation are applied on the data to eliminate the effects of long shadows. The vegetation is consisting mostly of spruces in terrain that has a lot of slopes. By a comparison with DSMs models, various forms of damage could be detected, ranging from slight defoliation at the tree tops to entirely dead trees. The achieved pixel accuracy of damage detection in flat terrain is 80% and 72% in rugged terrain. In this approach it has been shown that not only flat terrain is suitable for change detection, but also rugged terrain combined with unfavorable sun angles using satellite image data.

[Schwarz et al., 2001] describe a two way approach on forest storm damage classification in an area around the city of Bern in Switzerland. The data consists of satellite images from the IKONOS and the SPOT4 missions with resolutions of 4 *m* and 10 *m* respectively. The study area is located in the mountain range of the alps and includes different ground heights ranging from 555 *m* meters to 2060 *m*, and therefore contains many different types of trees and vegetation in general. The damage assessment is performed using a pixel wise classification approach that uses minimum and maximum values for each image band to assign each pixel to a certain class, as well as an object oriented approach that classifies entire shapes and objects. Labeling of the entire forest area is available for the entire area and is used together with visual interpretations of the images as ground truth. The achieved accuracy of the classification is 98% for detecting forest and 74% for detecting forest damage on the IKONOS images, and 98% for detection forest and 71% for detecting forest damage on the SPOT4 images.

2.1.2 Active damage detection methods

A method using active scanning proposed by [Rüetschi et al., 2019] utilizes Synthetic Aperture Radar (SAR) data to determine the difference in back scatter that is coming from healthy forest as well as damaged forest. It uses C-band SAR images from both before and after the storm event and has a minimal operating area of 0.5 *ha*. By using different user parameters, the degree at which the area is detected as damaged can be adjusted to adapt to different scenarios. With an accuracy of 88% of detecting storm damage in various test areas, it is a very promising method, especially since SAR technology is independent from atmospheric conditions and data can be generated very quick after a storm event.

A similar approach to detect forest damage caused by storms is performed by [Fransson et al., 2002] by analysing the SAR back scatter amplitudes of healthy and damaged forest. The study area is located

in southern Sweden with a majority of the forest consisting of spruces. As data source, CARABAS-II SAR data with a very high frequency is used with images available before and after a large storm event in 1999. It is shown that damaged forest shows considerably higher amplitudes in SAR backscatter than healthy forest, and even damaged areas that are already harvested could be differentiated from unharvested areas. The detection accuracy however is heavily dependent on the angle and direction in which the SAR images are taken, but in general it is proven that SAR data can be used to detect storm damage.

Another approach that combines both passive and active methods is utilizing Unmanned Aircraft system (UAS) data as well as UAS data together with ALS data on a study area of 200 *ha* in the middle of Slovakia in the Kremnica mountain range [Mokroš et al., 2017]. The ALS sensor operates in the near infrared spectrum with about 5 points per m^2 , the UAS data is taken from approximately 700 *m* height in short flight campaigns. The total detected areas of wind throw damaged forest are compared against each other and against reference data that was collected via Global Navigation Satellite System (GNSS). The overall matching percentage between reference and the UAS based approach is 82%, in combination with ALS data the overall matching percentage is 88%.

2.2 Historical background of Deep Learning

The first concepts of deep learning were created in 1943 by Walter Pitts and Warren McCulloch in the form of a rudimentary mathematical model that was based on the human brain and was created to mimic thought processes [McCulloch and Pitts, 1943]. Only a few years later in 1947, Alan Turing predicted the impact artificial intelligence would have on the modern job situation and formulated a test for the quality of machine learning networks that was named after him [Turing, 1948]. It states that a computer is considered as able to think, if it can perform a conversation of five minutes duration with a human without the human being able to tell whether it is actually a machine or not. In the past years some discussions have been going on whether this test has been passed or not, but there is no definitive answer yet as it is difficult to measure the credibility of these tests. The term machine learning was officially invented in 1952 by Arthur Samuel, who also created first computer programs that implement rudimentary machine learning to play the game of checkers [Samuel, 1959]. Five years later in 1957, the concept of deep learning was created by Frank Rosenblatt [Rosenblatt, 1958], and in 1960 the concept of backpropagation to modify the behavior of a system by Henry Kelley [Kelley, 1960]. The first working deep learning network which utilizes a feed forward structure that passes information and features through the network was created by Ivakhnenko and Lapa in the year 1965 [Ivakhnenko and Lapa, 1966]. Forwarding information between each layer was still performed manually via a statistical process and was therefore very slow. The first implementation of an Artificial Neural Network (ANN) used for image recognition, mainly hand written digits and characters, was created by Kunihiko Fukushima in 1979 [Fukushima, 1980]. This laid the ground stone for future image recognition models and the creation of more CNNs. In 1986 backpropagation was successfully used by David Rumelhart, Geoffrey Hinton, and Ronald J. Williams to improve the recognition of handwritten characters and other shapes [Rumelhart et al., 1986]. Another breakthrough in recognition of handwritten digits was achieved by Yann LeCun in 1989 with a combination of a CNN and

backpropagation, the resulting model was successfully used to read the numbers on handwritten checks in the United States of America [Y. LeCun, Boser, et al., 1989]. In 1998, backpropagation was vastly improved by Yann LeCun with the introduction of gradient based learning, which is a method of finding the optimal parameters to update models during the training process [Y. LeCun, Bottou, et al., 1998]. With the launch of the ImageNet data base in 2009, 14 million labeled images were made publicly available [ImageNet, 2016]. This greatly improved all sorts of ANNs in general, since large data sets have always been difficult to acquire, but are mandatory for training. A mayor breakthrough in deep learning was achieved by Alex Krizhevsky in 2011 with the model architecture *AlexNet* that consisted of eight layers and that won multiple prizes on the field of image recognition [Krizhevsky et al., 2012]. Following *AlexNet*, CNNs gained in popularity and many more successful architectures were created in the following years. In 2014, the company *Facebook* launched a face recognition system called *DeepFace* which has an accuracy of detecting faces of 97.4%, which is only slightly below the human accuracy of 97.5% [Taigman et al., 2014]. In the same year, the first Generative Adversarial Network (GAN) was created by Ian Goodfellow et al. [Goodfellow, Pouget-Abadie, et al., 2014]. GANs use two networks, one of which creates artificial data, and the other tries to determine whether the data is artificial or not. This method enables much faster learning and has lead to powerful tools like a real time image generation network that uses a simple segmentation map to create almost realistic looking scenes of landscapes ¹, or a network that generates human faces which are difficult to distinguish from real faces ². The speed in which ANNs are improving has been growing exponentially in the past years and will possibly lead to many new applications in the near future.

¹NVIDIA [2020]. *NVIDIA AI Playground - Research in Action*. URL: <https://www.nvidia.com/en-us/research/ai-playground/> [visited on 12/20/2019]

²West, Jevin and Bergstrom, Carl [2019]. *Which face is real?* URL: <http://www.whichfaceisreal.com/methods.html> [visited on 12/20/2019]

3 Data

3.1 Study area

The study area is located in the eastern part of the state of Bavaria in Germany and is marked as a red circle in figure 2.

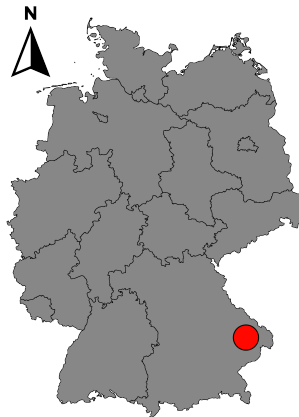


Figure 2: Location of the study area in the east of the state of Bavaria in Germany

Figure 3 shows the study area as an aerial ortho image with the damage that was caused by the big storm event of August 15th, 2017 overlaid as blue polygons.

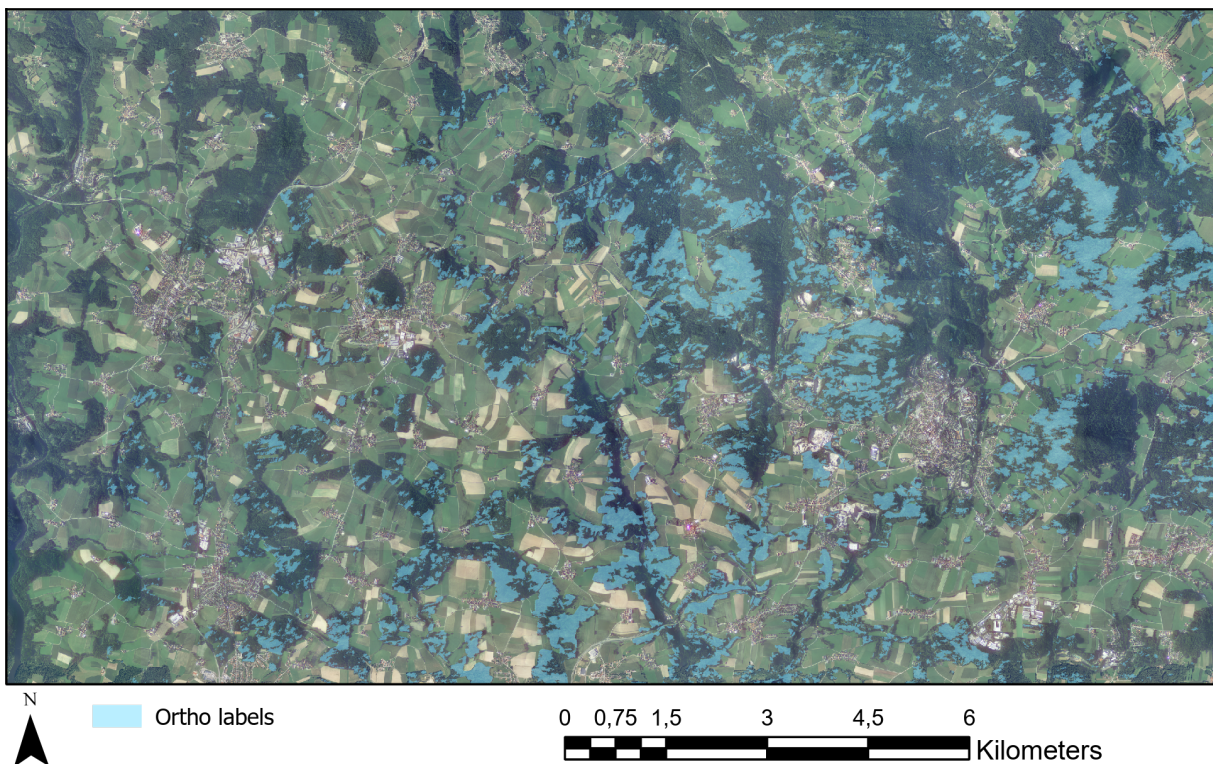


Figure 3: Overview of the complete study area with damaged forest included as blue polygons

The data is provided by the Bavarian State Institute of Forestry ([LWF](#)) and was acquired after the storm "Kolle" hit the region and caused significant damage to the forest. The data consists of two different data

sets, one containing satellite images and one containing aerial ortho-images, both covering the same area but with different resolutions and from different days. The study area covers around 160 km^2 and contains forests, fields and villages as well as a river and small lakes. The trees consist almost exclusively of conifers, which all have similar colors and shapes when viewed from above. Due to large parts of the forests being owned by private persons, the exact location of the study area is confidential and can not be published in this thesis.

3.2 Satellite images of the study area

The first part of the data includes satellite images from the *PlanetLabs* cooperation [PlanetLabs, 2019]. Its *PlanetScope* service consists of around 130 small Dove CubeSat 3U satellites each with a dimension of $10 \times 10 \times 30 \text{ cm}$. The satellites are sent in batches into earth orbit, often as additional payload to larger satellites, and their total amount is steadily increasing. With an orbit height of 475 km and a maximum latitude of $\pm 81.5^\circ$, the entire land surface of the earth can be mapped. A sun synchronous orbit and a daily revisit time of each satellite allow for at least one image per day at any given land area, the services is advertised to provide an image up to twice a day in certain regions. Each satellite can cover an area of $24 \times 8 \text{ km}^2$ in a single image, newer generations achieve $24 \times 16 \text{ km}^2$. Each individual image is composed of multiple frames before and after an anchor frame, with an alignment accuracy of 0.4 pixels. In post processing, the images are re sampled to a resolution of 3 m/pixel , however the signal to noise ratio is not ideal, which makes the images look a bit blurry. Each satellite image consists of the four bands red, green, blue and near infrared with a color depth of 16 bits per channel, although only a small part of the whole spectrum is used. This leads to the images appearing very dark up to almost almost black to the human eye if not artificially brightened up. The radiometric resolution however is still much higher than that of most ordinary images which use a color depth of only 8 bits per channel.

In total five different satellite images are available, each covering the same area, but on five different days. With the 7th and 15th of August two images before the storm and with the 23th, 27th and 30th of August three images after the storm, which occurred on the 18th of August 2017, are available. The images on the 15th and 23th contain some areas without data, the latter shows also cloud coverage in parts of the image. Each individual image scene has due to the time of day, the sensor calibration as well as atmospheric effects a unique look in terms of brightness and color composition of the individual bands. Each of the five available images consist of 5500×3334 pixels and is saved in the *GeoTIFF* file format, which allows for a lossless compression as well as georeferencing via the files own meta data. The figures 4 to 8 each show a small excerpt of the five satellites images that is brightened up for display purposes.



Figure 4: Satellite image, August 7th



Figure 5: Satellite image, August 15th



Figure 6: Satellite image, August 23th



Figure 7: Satellite image, August 27th



Figure 8: Satellite image, August 30th

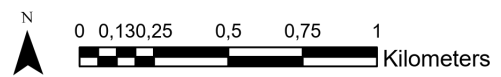


Figure 9 shows the histogram for each of the four image bands red, green, blue and near infrared for all three after storm satellite images that are combined used for training. The orange box on the right side of the figure is a zoomed in version of the orange box on the left side. A lot of pixels in the images have the value 0 due to some areas without data, to make the graphs still readable these values are excluded from the histogram curves. The individual peaks in every band are likely the result of the three satellite images being mixed together, and each image having a slightly different color range.

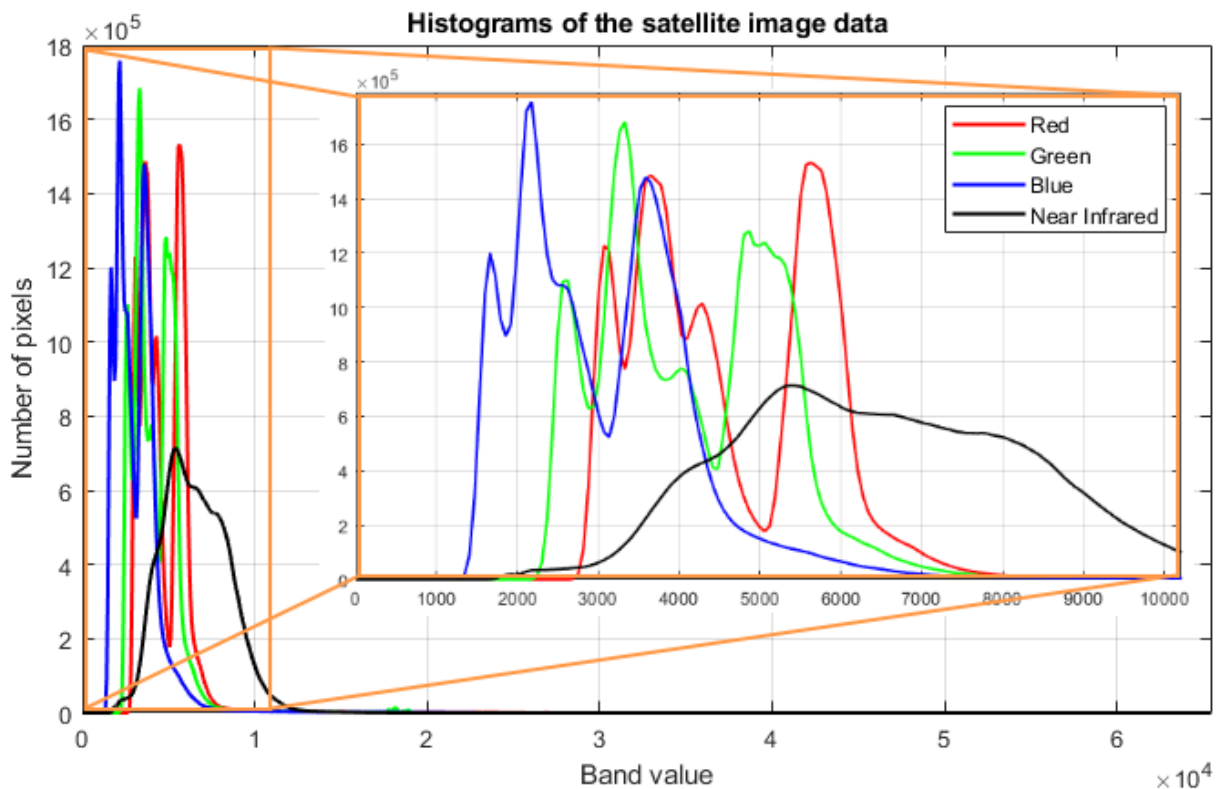


Figure 9: Histogram curves for each color band of the satellite images. The images consist of the four bands red, green, blue and near infrared. The orange box on the right is a zoomed in version of the orange box on the left

3.3 Aerial ortho-images of the study area

The second part of the data consists of an aerial ortho image that was placed in order after the storm event and that shows the area as it was on the 29th and 30th of August. The image was taken by plane as a set of small images with a horizontal overlap of 80% and a vertical overlap of 50%, that were afterwards ortho corrected, color adjusted and merged. The spatial resolution is 20 cm, which makes it possible to detect individual knocked over trees on the ground. The band composition is the same as the satellite images with red, green, blue and near infrared, however the data is available only in 8 bit color depth, but uses a much wider spectrum in each individual band. To compensate for large file sizes, the data comes spitted into 45 tiles without overlap, each tile containing 10000 × 10000 pixels. The data is compressed in the *JPEG 2000* file format, which allows for a loss less compression and integration of geospatial meta data into the file itself. Figure 10 shows the same scene as the figures 4 to 8, and figure 11 shows a close up version that contains individual trees. The shadows in the image scene are falling to the north east, which indicates that the time of day in which the images were taken is in the afternoon.

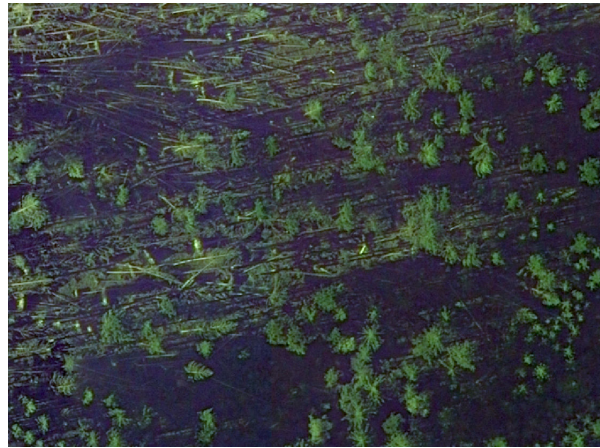
Figure 10: Ortho image, August 29/30thFigure 11: Ortho image close up, August 29/30th

Figure 12 shows the histograms for each of the four bands red, green, blue and near infrared of the ortho image data set.

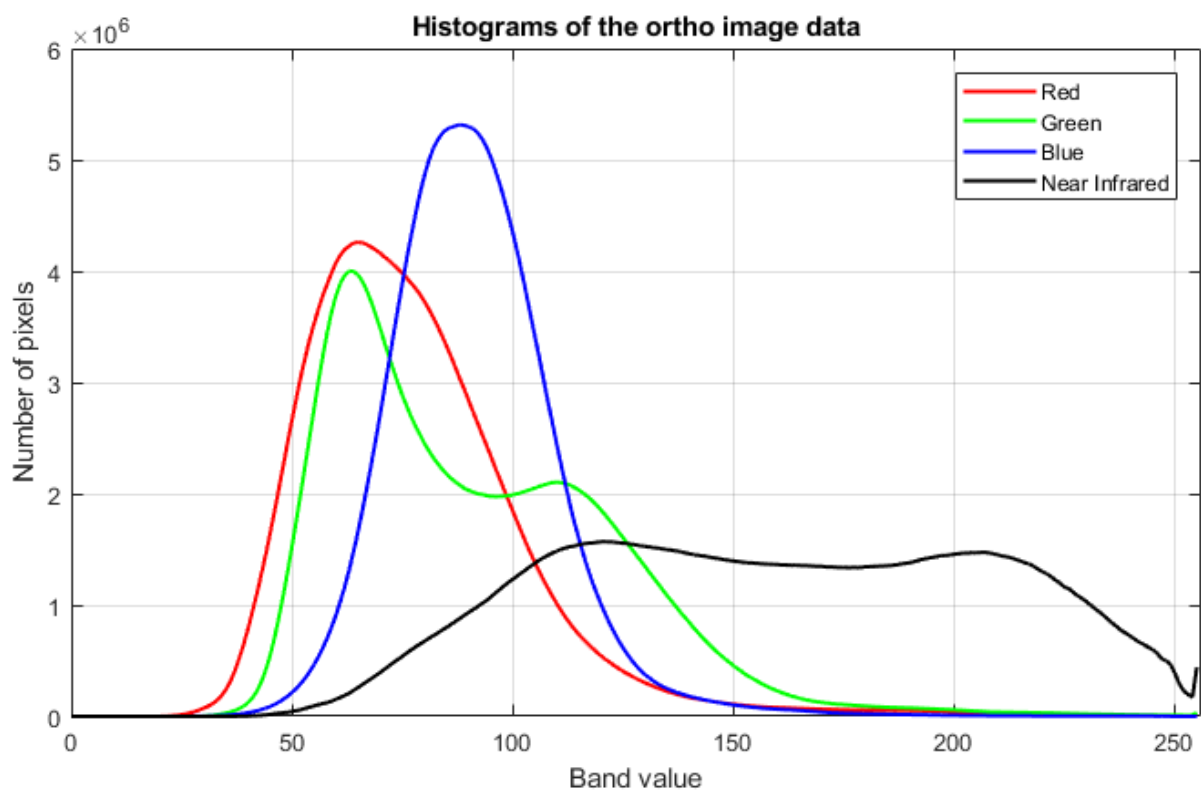


Figure 12: Histograms for each color band of the ortho images

3.4 Labeling of the damage

In the course of assessing the inflicted damage, the LWF ordered labeling of the data in form of polygons. This labeling was first performed on the satellite images and later, once they were available, also on the ortho images, resulting in two different sets of damage masks. The polygons created using all three post-storm satellite images are clearly limited in their detail grade due to the pixel size of 3 m. They also miss out some damaged areas which are simply not visible to the human eye in that resolution and color

range. The labels created using the ortho images however are highly detailed, up to including individual trees in some areas if necessary. The total area of damage in the satellite labels is 7.85 km^2 , while the total area of damage in the ortho labels is 17.35 km^2 , which is an increase of 121%. The data shows in both sets of labels a strong unbalancing between the area inside the polygons compared to the area outside, since the available data covers an area of 160 km^2 and only a small part of it contains damaged forest. Due to the high detail grade, the ortho image labels are accepted as ground truth, which means they mark the area that has actual damage ideally without any errors. The figures 13 and 14 show both sets of labels on top of their respective images, in this case the satellite image of August 30th is used.



Figure 13: Satellite labels

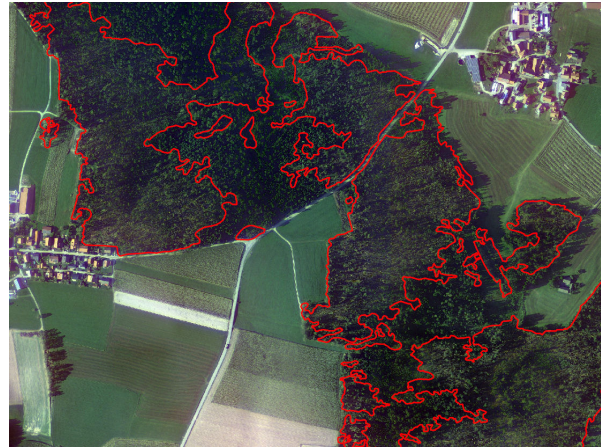


Figure 14: ortho labels

3.5 Transfer ability study using an independent data set

A third set of data which includes both satellite and aerial ortho images is provided by the company *HessenForst* [HessenForst, 2019] and covers about 7200 km^2 of land in the German state of Hesse. The data was collected after a severe storm on the 18th of January, 2018. The storm alone however is not the only source of forest damage, as the region is plagued by the bark beetle. The resolution for satellite images is 4.77 m coming from the PlanetLabs Rapideye satellite mission with the five spectral bands of red, green, blue, red edge and near infrared [PlanetLabs, 2019]. The resolution the satellite images does not match between the two data sets, in theory resampling methods could be applied, but it is also a good test to let the model predict damage in a different resolution as it was originally trained for. The ortho images were taken in June and July of 2018, several month after the storm event and are available in a resolution of 20 cm and the four spectral bands red, green, blue and near infrared. The data does also include labeling which was performed on the ortho images, but the labeling is not as precise as the labeling of the LWF. It contains also some inconsistencies, since not all of the forest is state-owned, and the labels are only available for the public areas. Also some areas that contain forest damage are not labeled, since the damage did not originate from the latest storm but from some previous events of various kinds. This makes training and creating meaningful predictions hard without extensive processing of the labels. The data from HessenForst became available after the methods of training in this thesis were completed, so it could not be used during training itself. It is however used as an independent test data set to evaluate the performance of the trained models on completely new data.

The area in which the HessenForst data is available is located in the northern part of the state of Hesse in Germany and is marked as a red circle in figure 15.

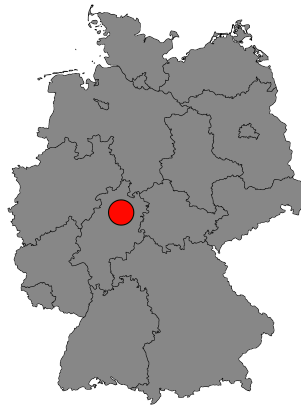


Figure 15: Location of the test data provided by HessenForst in the northern part of the state of Hesse in Germany

Figure 16 shows two small excerpts of the HessenForst data set. The image on the left side is an ortho image, the image on the right side a satellite image. Both images have the same labeling that was created using the ortho images overlaid on top as green polygons. The time of day at which the ortho images were taken appears to be afternoon, since the shadows are falling to the north east.

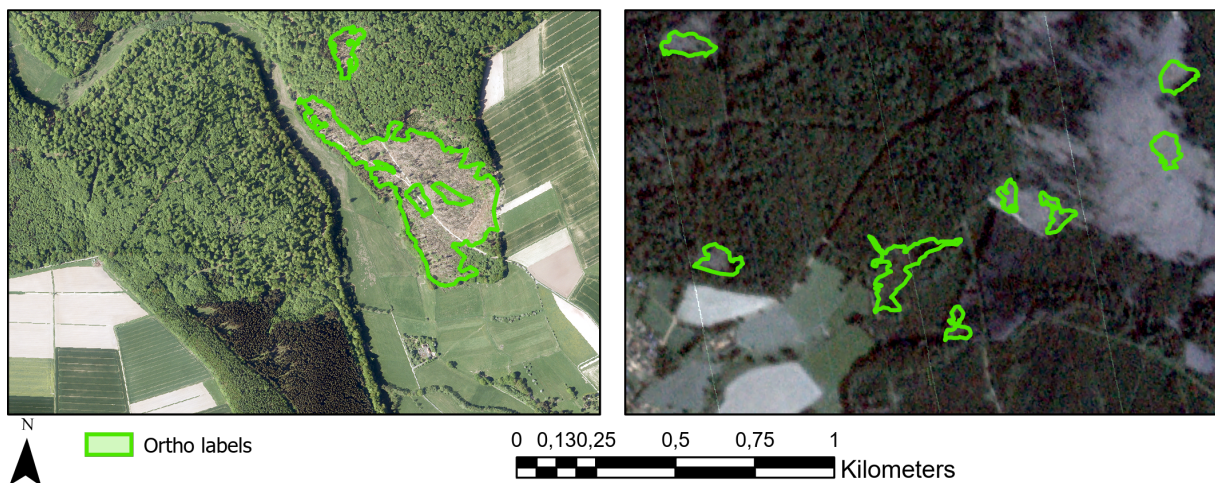


Figure 16: Example excerpt of the HessenForst data as images combined with labeling. On the left is an ortho image, on the right a satellite image

Figure 17 shows the histogram curve for each of the five image bands blue, green, red, red edge and near infrared for the Rapideye satellite images provided by HessenForst. The orange box on the right side of the figure is a zoomed in version of the orange box on the left side. In the images a lot of pixels have the value 0 due to some areas without data, to make the graphs still readable these values are excluded from the histogram curves. The near infrared band has only pixel values of either 0 or 65535, this band is also set entirely to 0. In comparison to the PlanetScope satellite images, the histogram curves are very tight and centered around very low values. The individual curves are difficult to compare to the PlanetScope images, since that data set consist of multiple images mixed together.

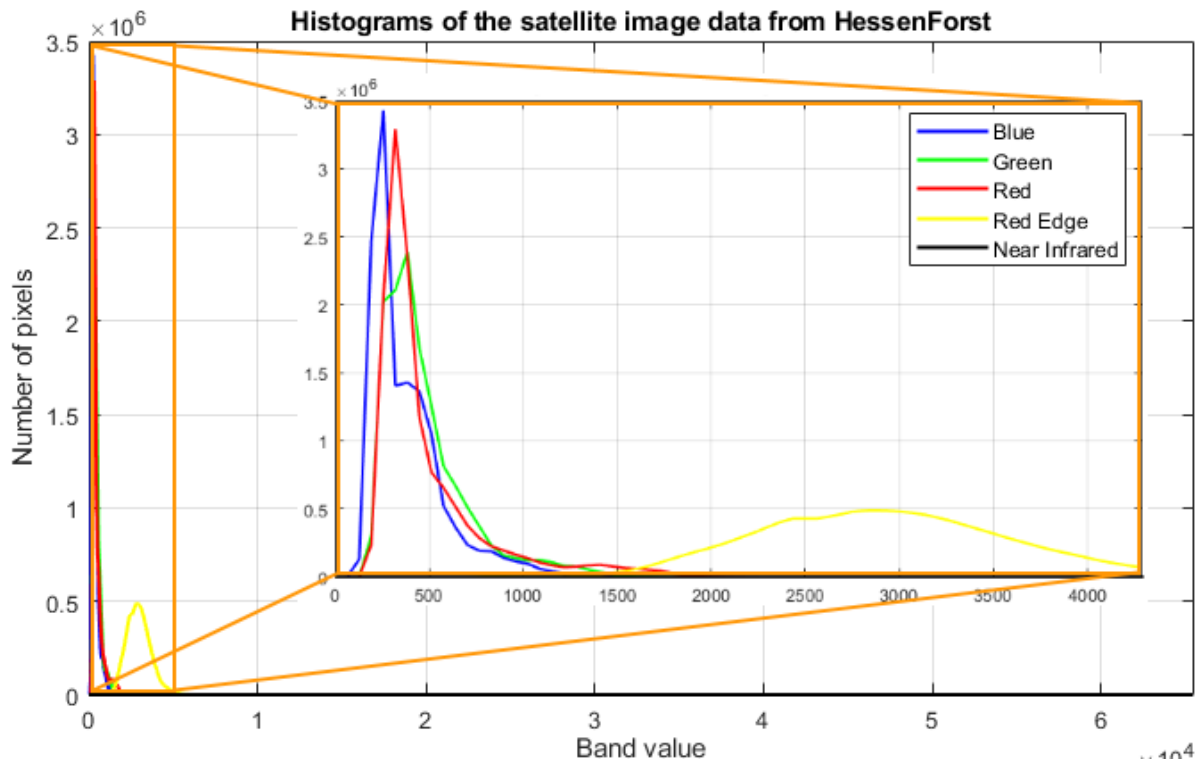


Figure 17: Histogram curves for each color band of the satellite images provided by HessenForst. The orange box on the right is a zoomed in version of the orange box on the left

Figure 18 shows the histogram curves for the four bands red, green, blue and near infrared of the ortho image data set provided by HessenForst. The pixel value distribution is about equal to that of the images provided by the LWF, with the exception of the near infrared band which has a more significant peak.

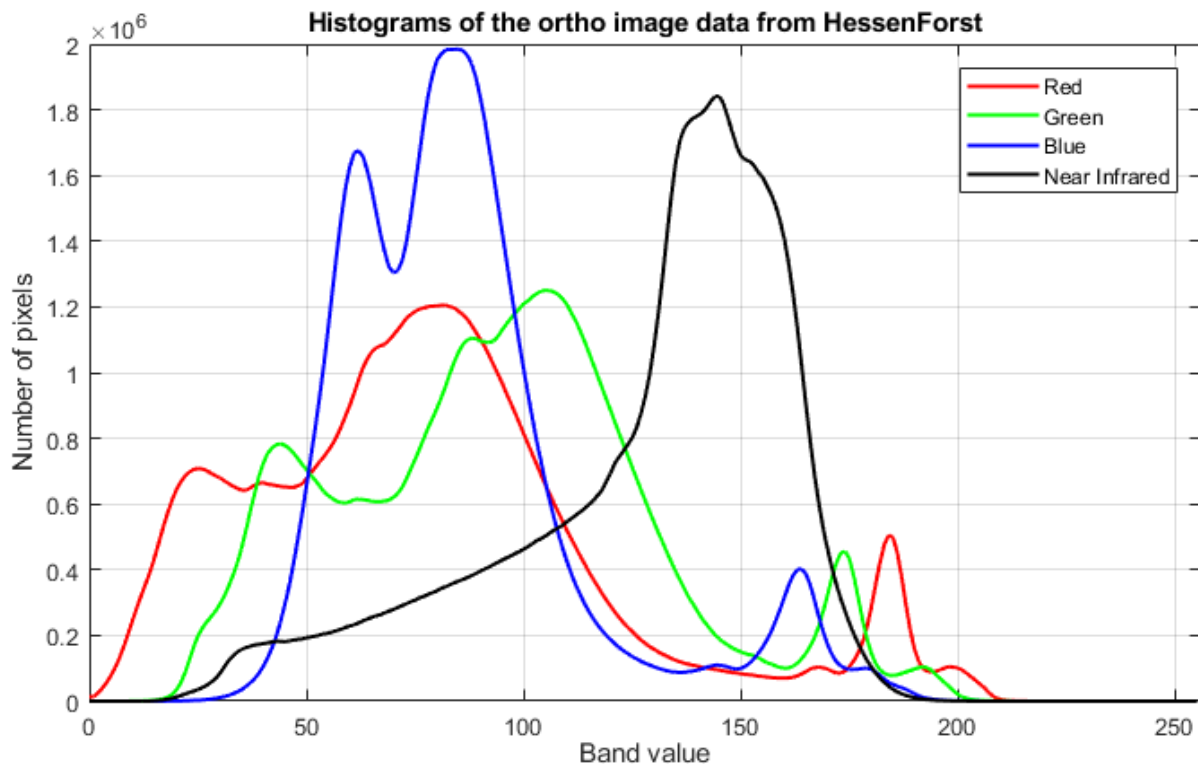


Figure 18: Histogram curves for each color band of the ortho images provided by HessenForst

4 Methods

4.1 Overview of the used methods

Before the actual training can be done, the data has to be prepared so that it consists of small image tiles each accompanied by an according label tile. Once the data is in the correct shape, augmentation techniques are applied which alter the input data in a way so that the resulting model is more robust to variations in the test data. The training itself is performed on specialized Graphic Processing Units (GPUs) at the Leibniz Supercomputing Centre (LRZ) with different parameters. The resulting models are compared against each other and the best performing models are imported into the GIS software ArcGIS Pro. By using a custom toolbox, predictions of damage can be created and transferred to vector format, which enables a usage of further GIS specific operations.

Figure 19 shows the road map in which the complete process of tiling, training and integration into GIS is performed. It starts with the software ArcGIS Pro, in which the data is preprocessed and split into a large number of small tiles. The next step is to import the preprocessed data into a python environment, in which further processing and image augmentation steps are performed. Once the data is ready, it is split into a training data set and a test data set in a ratio of 80:20. This is done for satellite and ortho data respectively, so two completely independent data sets are generated, each containing training and test tiles. The next step is to construct several CNNs using the deep learning framework *TensorFlow* [Google, 2019c]. Tensorflow, like all deep learning frameworks, is based on the modification of tensors, which are objects that can hold vectors or matrices. Since TensorFlow itself is rather complex and complicated to implement, a wrapper called *Keras* [Chollet, 2019] is used to allow for a much simpler usage of TensorFlow. Keras offers a lot of functions that call TensorFlow functions themselves, while keeping the syntax very minimalistic and clean. The models for satellite and ortho data are independent from each other and are trained on their own using different sets of parameters to find the optimal configuration. By using the test data set, the best performing models are selected, exported and integrated into ArcGIS Pro with the help of a custom toolbox. Inside ArcGIS Pro, predictions of the damage can be generated for both satellite and ortho data respectively that result in polygon masks which can be further adjusted using GIS based methods.

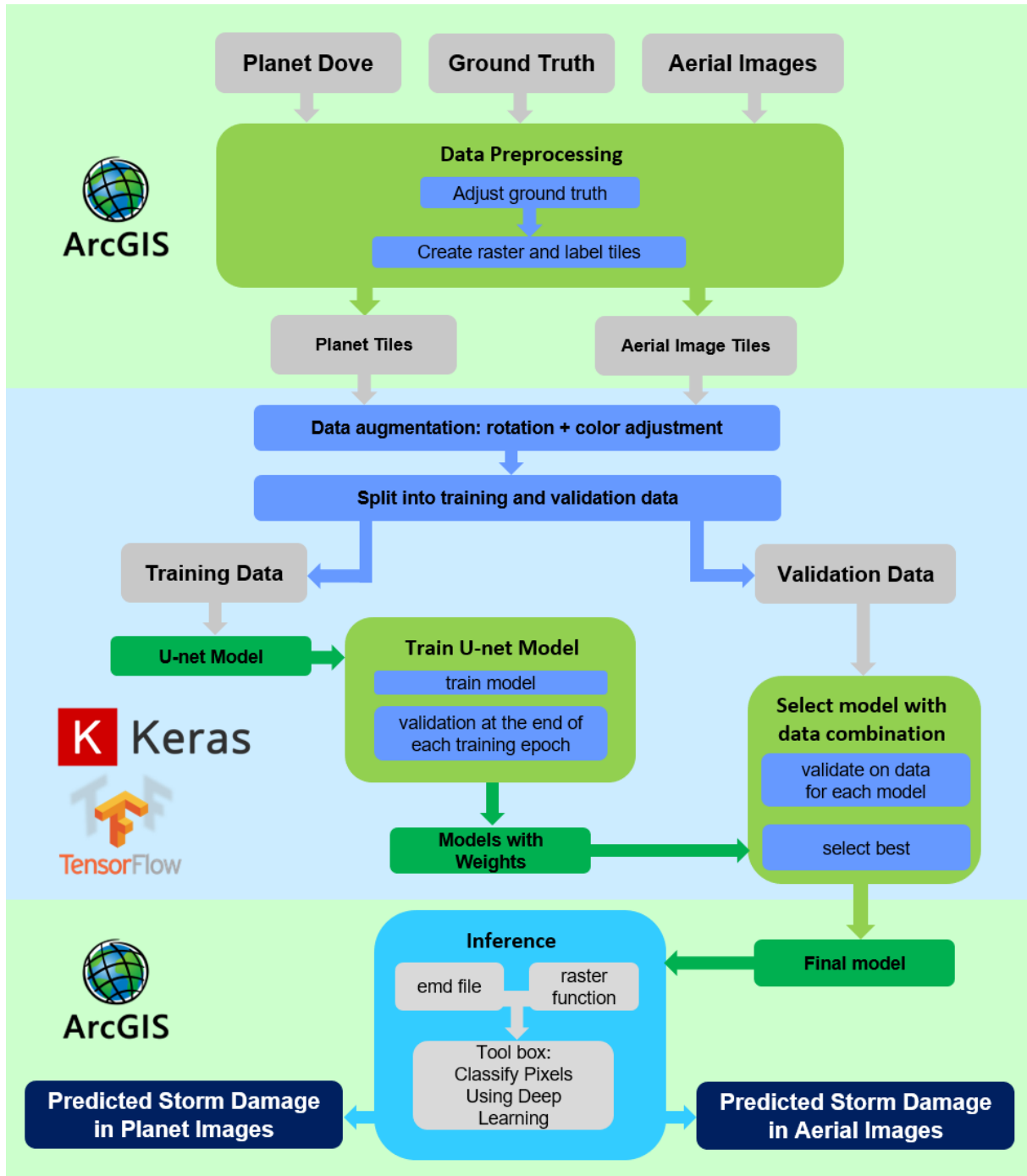


Figure 19: Road map of the complete training and prediction process. It is divided into three sections: Preprocessing in ArcGIS Pro using specialized toolboxes, training in a python environment using Keras and Tensorflow, and execution using a custom inference function and a custom toolbox in ArcGIS Pro

The following chapters will go more into detail about the individual steps depicted in figure 19.

4.2 Tiling of the input data

In order to feed data into a [CNN](#), it has to be brought to a certain shape. Usual tile shapes are ranging from 64 to 300 pixels, with three bands for red, green and blue, but more channels like near infrared can be added as well. In case of the labeling being also provided in image form and not just as a number that represents a class like cat or dog, it has to match the shape of the input images with one band and a certain pixel value for each class. In the case of forest damage detection the labels consist of the two classes damage and no damage, which results in a binary image.

ArcGIS Pro offers a toolbox to export training data for Deep Learning, which creates image and label tiles of a specified size from the input data. If the damage masks are provided as polygons in vector format, they are automatically converted to a raster mask, and an overlap between the individual tiles is also possible to generate more training data. In case of the ortho images, about 14600 tiles are created with a shape of $256 \times 256 \times 4$ pixels. This includes only the tiles that have at least one damage pixel in the label mask. The labels are in the dimension of $256 \times 256 \times 1$ pixels. An overlap is not used since the resulting data is already large enough for the training to take a long time. The proportion of damage to no damage pixels is around 35 : 65, an ideal proportion would be 50 : 50.

To allow for more flexible experiments regarding the tile size and overlap of satellite images, the tiling of satellite image data is performed at the [LRZ](#) with python code right before each training session. For the satellite data the tiling is performed both in 128×128 pixels and 256×256 pixels to test which size brings the better performance. A horizontal and vertical overlap with a stride of half the tile size is used to artificially enlarge the training data set. For the tile size of 128×128 pixels about 6000, and for 256×256 pixels about 2000 tiles are created. The three images after the storm event are uploaded as complete image along three sets of labels which are converted to binary images before. Each of the three label images has slight variations in the damage area depending on cloud coverage and data completeness of the corresponding image. The satellite image tiles show a heavy class unbalance with up to 95% of all pixels containing no damage, depending on the tile size and overlap. During training and for the test data set, the tiles of all three images are mixed together and considered as one big data set. Due to the low resolution, the tiling of the satellite data takes only a few seconds, whereas on the ortho data set it takes several hours until all 45 images are processed. Before the training can begin, the pixels of the image tiles are normalized between the values 0 and 1 as 32 bit floating point numbers, since [CNNs](#) perform particularly bad on large value differences, and 16 bit integer data leads to no usable results. The 8 bit ortho data could be used, but it is nevertheless also normalized. The figures [20](#) and [21](#) show an example pair of image and label tile for satellite and ortho data. The white area in the label corresponds to damage in the forest, the black area is the background without damage. The different shapes of the labels are a result of the 15 times higher resolution of the ortho images compared to that of the satellite images.

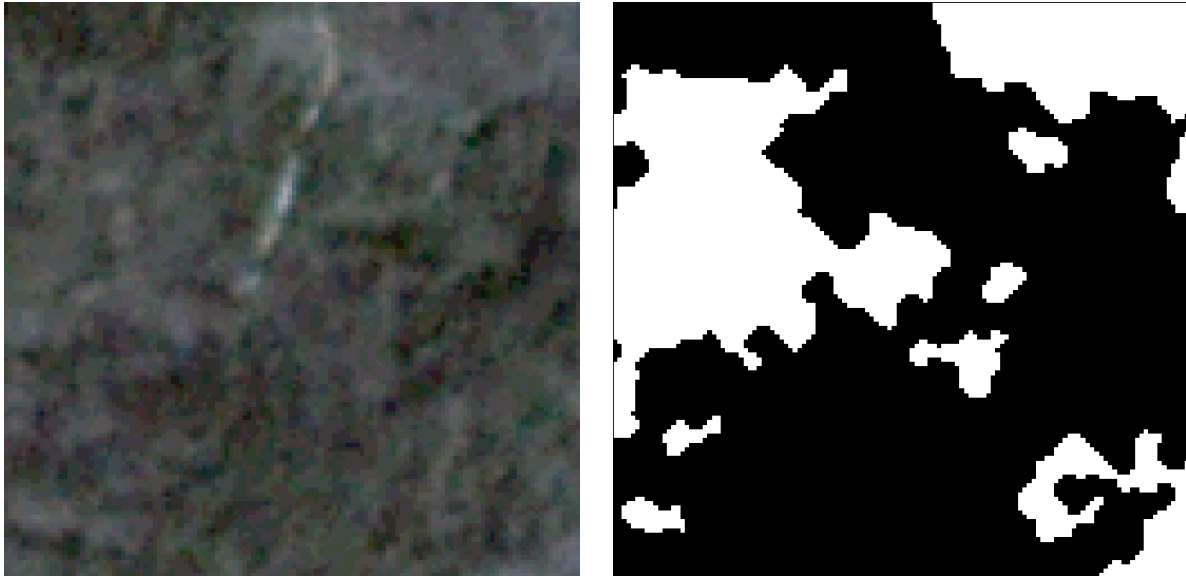


Figure 20: Example satellite image tile with a size of 256×256 pixels and 4 image bands (left) and the according label tile as binary image, white representing damage and black no damage (right)



Figure 21: Example ortho image tile with a size of 256×256 pixels and 4 image bands (left) and the according label tile as binary image, white representing damage and black no damage (right)

It is important to have a test data set to measure the performance of classification methods. This test data may not be used during training and should ideally contain all aspects of the original data, like for example especially bright regions or certain types of forest. To create this test data set for satellite data, a portion of the original images and labels is split off during the tiling process. This portion is about 15% of the bottom part of each image, which translates to 500 pixels in height, so two full rows of tiles of 256 pixels height or four rows of 128 pixels height can fit into it. Figure 22 illustrates this split. For the ortho test data set, 8 out of the 45 10000×10000 pixel images are picked, which corresponds to about 18% of the available data. The test tiles are split evenly across the study area to ideally include every facet. Figure 23 illustrates the selection of the ortho test data.

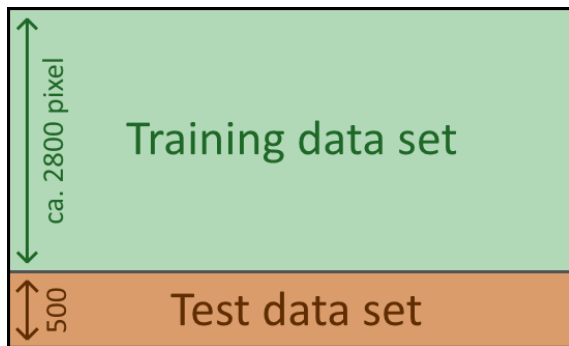


Figure 22: Split of satellite data set into training and test data, so that two or respectively four full rows of test tiles can be produced at the bottom

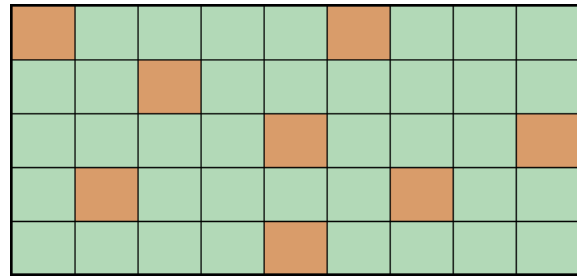


Figure 23: Split of the ortho data set into training (green) and test (orange) sections. The distribution is made in a way so that the test data is spread out and contains all occurring scenery types

4.3 Data augmentation of the input

Data augmentation can be used both to increase the amount of available training data and to increase the robustness of the models to variations in the data. There are two basic ways to perform augmentation, on the one side is geometric augmentation and on the other side radiometric augmentation. Geometric techniques are performed both on the image tile and the label tile in an identical way, radiometric techniques can be performed on the image tiles only.

4.3.1 Geometric data augmentation

Simple geometric image augmentation techniques include rotations and mirroring. If the angle in which the rotations are performed is a multiple of 90° , the rotated image can be created without any pixel interpolation. If the rotation angle is no multiple of 90° , the pixels of the resulting image are interpolated, and at the corners of each image tile are pixels without a value. Since the tiling process on satellite data is performed each time the training is started, a solution to compensate the latter problem is to rotate the entire original image in for instance 45° and then perform the tiling. This way only some tiles at the borders have pixels without value, these tiles can be left out without reducing the data set too much. Experiments have however shown that interpolated tiles perform a lot worse in the training than the tiles with rotations in multiples of 90° . This is likely due to the fact that the interpolation makes the image even more blurry, and also creates small artifacts at the edges of the damage mask. Since every pixel corresponds to 3 m on the ground, a straight line that is changed to a zigzag pattern and vice versa can lead to large aberrations. Because of this, only rotations around degrees in multiples of 90 are used. The effect on the performance would most likely be much lower with the ortho images, but the tiling for this data set is performed in ArcGIS Pro, which at the time this thesis is written does not support tile rotation, with the feature being planned for a future release. Besides rotations, mirroring can also be applied, with the easiest forms being a flip along the vertical and a flip along the horizontal axis. However, the result after a rotation of 180° equals the result of a horizontal flip followed by a vertical flip. To not create redundant data, only one rotation is performed at 90° , while mirroring is done on both axis subsequently after the rotation. These augmentation steps increase the total amount of training data by a factor of 8. The augmented data however is correlated to the original data and does not bring the same training value as completely new images.

The geometric image augmentation does, if performed as described, create data that is unlikely up to impossible to occur in real world scenarios. For instance, shadows that fall to the south require the sun to shine from the north, which is impossible to occur in Germany. In the satellite images, shadows are not a significant feature, since in areas with damage only a more or less uniformly grey shape is visible that stretches over a large area, so all described rotations and flips are used on satellite data. In the ortho images however, shadows do play a very important role, and the model does most likely learn to consider them when creating the prediction. Therefore only geometric augmentations that result in scenes which can naturally occur in Germany are used for ortho images. This includes a flip of the original tile along the vertical axis, a counterclockwise rotation of 90° of the original tile as well as a flip along the vertical axis of the rotated tile. Figure 24 shows the different sun angles that are created by the data augmentation techniques for ortho tiles. In the original tile on the left, the sun is shining from the south west, so the shadows fall to the north east as indicated by the yellow arrows.

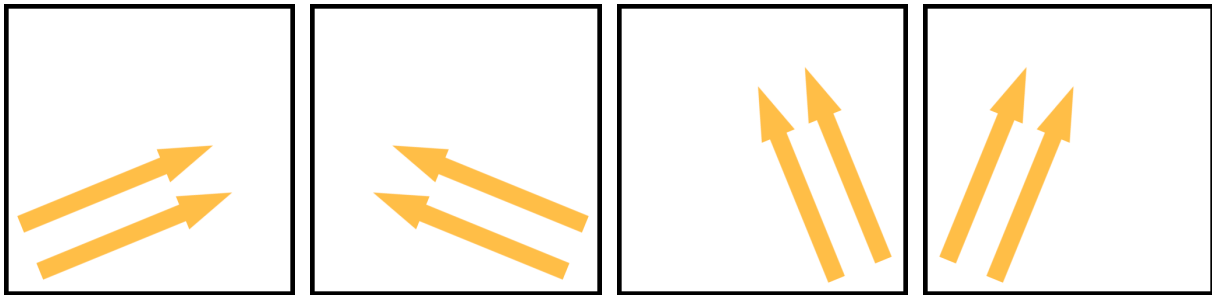


Figure 24: Different sun angles indicated by arrows that occur when rotating a training tile. From left to right: 1. Original sun angle, 2. flipped along the vertical axis, 3. rotated counterclockwise 90° , 4. rotated counterclockwise 90° + flipped along the vertical axis

4.3.2 Radiometric data augmentation

With radiometric image augmentation, the colors of the image tiles are altered. This can be used to mimic sensor noise and to create additional training data, which increases the pool of available data and also increases the robustness of the classifier. Each of the four image bands is altered separately by adding a normally distributed value to every pixel. This results in every augmented image having slightly different color values, some appear more red, some more blue, etc., some are brighter and some darker. To create the normal distributed values, a Gaussian normal distribution with sigma values between 1 and 3 are used in different tests. A sigma of 1.5 results for example in a random distribution of values as shown in figure 25. Since the image tiles are normalized between 0 and 1, the random noise is divided by 256 to also fit in this range. In theory, the same sigma should yield different results for satellite and ortho images, since the satellite images do not use the full band spectrum but only a part of it, and the ortho images use the full width. This results in the noise being larger in relation to the one data set, and smaller in relation to the other data set. The satellite data however is rather inconsistent on its own throughout the five available images, with possible further inconsistencies in the future, which probably requires a larger noise to make the model more robust. The ortho images however are quite consistent, since they are specially processed to give them a uniform appearance, which should lead to a smaller required noise value. Therefore, the through tests as best performing selected sigma, will be applied to both models. In theory the training data set can drastically be increased by just adding the same tiles

repeatedly each time with different additional noise. However to not create too much redundant data, the original tiles along with two iterations of color adjusted images are used during training of both models. This guarantees that the training data set does not consist solely of augmented data, but does also contain some original tiles.

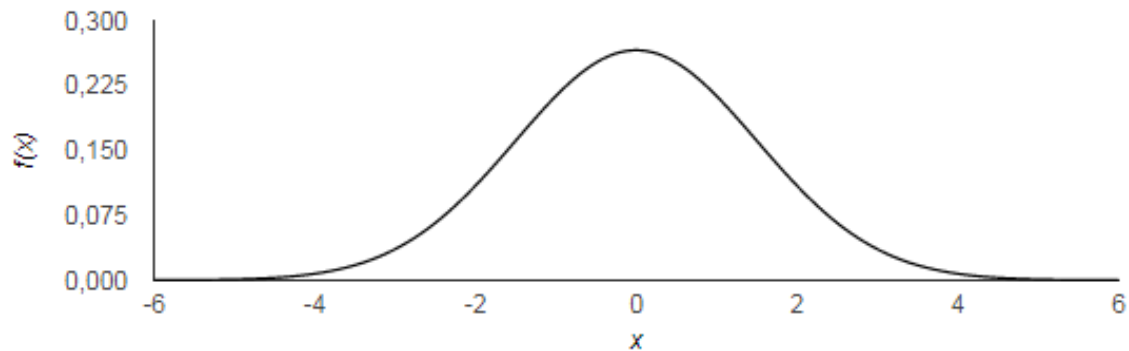


Figure 25: Normal distribution of the random noise that is added to the training data with a sigma of 1.5 and the center at 0, before the division by 256

An other technique to increase the amount of training data and especially the performance on the discrimination between healthy and damaged forest is to include some image tiles that contain exclusively healthy forest. This helps the model to decrease the area that is incorrectly predicted as damaged, since it has more images without damage in its training repertoire. By adding a selection of tiles without damage, the ortho data set is increased by another 4.8%. Since the satellite images already have a strong class unbalance in favor of no damage, they will not be augmented with additional tiles. In total after the various steps of data augmentation are applied, about 24000 satellite tiles and about 60000 ortho tiles are available for training.

4.4 Used hardware for training

In terms of required hardware for training a [CNN](#), the most important part is the [GPU](#). With its special parallel processing architecture it offers the possibility to process multiple images at once and thus greatly decreases the time required for training. Depending on the architecture of the model, the training can require a lot of memory on the [GPU](#), therefore high-end hardware with 16 gigabytes of graphic memory or more can be required. For this thesis, a Nvidia Tesla P100 [GPU](#) with 16 gigabytes of memory is used alongside 488 gigabytes of main memory. Students in Bavaria have the possibility to rent such hardware at the [LRZ](#) during their study. With the help of an online registration tool, one of four available [GPUs](#) can be rented for a maximum duration of eight hours, but the duration can be extended as long as it is not blocked by another user. As soon as the reservation ends all data is wiped from the memory to give the next student a clean new system. At the beginning of each session all necessary [GPU](#) drivers are already installed and a special Nvidia-Docker image is running. With this image other images containing custom code and data for the training process can be downloaded onto the system. One downside of this implementation is that it requires the data to be transferred to the system every time a training session starts, which can take some time at large data sets. A solution to maximize the amount of time that is actually used for training is to perform all image augmentation

steps at the LRZ and only upload the raw data, ideally in compressed form. Another drawback is the automatic removal of all data once the reservation has run out, which can lead to the loss of all training results. A proposed solution could be an additional storage space that can be accessed several hours before and after the training, so no precious GPU time is wasted with uploading or downloading data.

4.5 Artificial Neural Networks and Deep Learning

ANNs have proven in the last years to be suitable for various kinds of tasks including process control, face identification, 3D reconstruction and also image classification. Most approaches are based on pattern recognition, which can be used to predict future events, recreate certain input scenes or extract certain features of data. It is not uncommon for neural networks to outperform human operators, both in accuracy and especially in speed. Due to their complex nature, a multitude of different approaches to construct ANNs are available, the following chapter will give an introduction into the topic with a special focus on Deep Learning.

The term "Neural Network" refers to the fact that the networks are inspired by a real world biological neural network like it can be found in brains of animals, on a much smaller scale and with less complexity. Each neuron of the artificial network is capable of receiving, processing and emitting data, and combined they can be used to mimic real life learning. The process of data handling inside the network at each individual neuron can be described as follows:

A neuron receives input signals x_i from other neurons via its synapses, and the signals get multiplied with the weight ω_i of each individual synapse. The weight, also called strength of the connection, is a trainable parameter that is used to control the connection and influence between the individual neurons. The signals arriving at the synapses are then carried on via the so called dendrites to the cell body, where they get summed up. An additional bias b is added to the signal as a controlling factor to be able to better fit the network prediction to the data. Each neuron has an artificial signal threshold, above which the output a of the activation function f is sent via the so called axon to a single or multiple other neurons, depending on the architecture of the network. Simple activation functions are for example a classic step function, which either outputs 0 or 1 depending on if the threshold is reached or not, or linear functions. Usually however the activation function is a non-linear function, examples are the *Sigmoid* function (1), the *Tangens Hyperbolycus* function (2), or the *ReLU* function (3). The *ReLU* function will be used to train the networks on the satellite and ortho images.

$$A(x) = \frac{1}{1 + e^{-x}} \quad (1) \quad A(x) = \frac{2}{1 + e^{-2x}} \quad (2) \quad A(x) = \max(0, x) \quad (3)$$

$$\text{The output of the neuron fits the equation: } a = f\left(\sum_i \omega_i x_i + b\right) \quad (4)$$

Figure 26 shows a biological neuron along with an artificial neuron as it is used in neural networks.

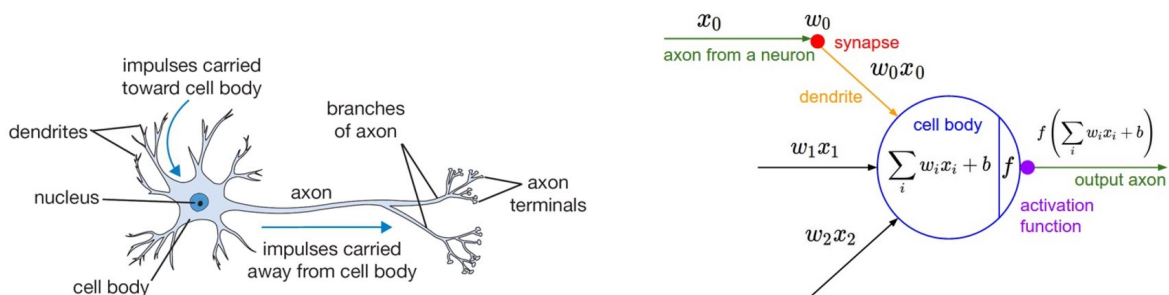


Figure 26: Comparison between a biological neuron (left) and an artificial neuron (right)³

The connection of all artificial neurons is called the neural network, where each neuron passes data forward to one or more other neurons without any cyclic connections. A group of neurons connected in a certain way is called a layer. If multiple layers are connected in a single model, one talks of deep learning. Deep learning is therefore a special field within the field of artificial neural networks.

In general machine learning is used for mathematical pattern recognition, while the focus of deep learning lies on image classification and object detection, sometimes in real time as it is used for example in autonomous driving. In a deep learning model there are several layers chained together which all pass information from one layer to the next, the network is therefore called a feed forward network. In a deep learning network there are three types of layers: Input layers, output layers, and all layers in between, which are called the hidden layers. The size of a neural network refers to the amount of connections between the neurons (synapses), while the number of hidden layers is called the depth of a network. If every neuron in a layer has connections to every neuron in the previous and the following layer, the layer is called *fully connected*. Fully connected layers require a lot of computation power, and therefore are usually only used as input and output layers. When speaking of **CNNs**, there are also other types of layers which do not consist of artificial neurons but of other mathematical operations like convolutions, padding functions or pooling operations. The way the individual layers are structured is called the architecture of the network, and there are numerous architectures which have already proven to work well for their respective tasks. A usual practice to work with new data or a new task is to take an already existing architecture and to alter or add only some layers to create different forms of output data.

³Jordan, Jeremy [2017]. *Neural networks: representation*. URL: <https://www.jeremyjordan.me/intro-to-neural-networks/> [visited on 12/20/2019]

4.6 Overview over Convolutional Neural Networks (CNNs)

CNNs derive their name from convolution filter layers that are used for feature extraction. Each filter layer contains a multitude of individual convolutional filters usually with a size of 3×3 pixels. The values of each filter can be adjusted, this is basically the process how CNNs are trained and learn to create predictions. CNNs excel especially at image recognition tasks, since they allow for deep architectures and are able to extract also small and rather non prominent features in the input data. Besides convolutional layers there are also other types of layers, an overview of the used layers in this thesis is given in the following section.

2D convolution layer

The 2D convolution layer performs basic convolution operations as they are used in image processing. A number of filters, usually a power of two, is applied to every layer of the data, the filters itself are created with random values using the LeCun normal filter initializer [Y. A. LeCun et al., 2012]. These random values are adjusted in each iteration by an activation layer, this is where the training process takes place. The standard filter size is 3×3 , which is also used in this thesis. Before the filters are applied, a padding operation is used on the input data, which adds a border of one pixel with 0 in every value to the input, so the generated output has the same dimension as the input. Figure 27 shows an example filter which is applied to an input image. In the example a sobel filter, which is used for horizontal edge detection, is used.

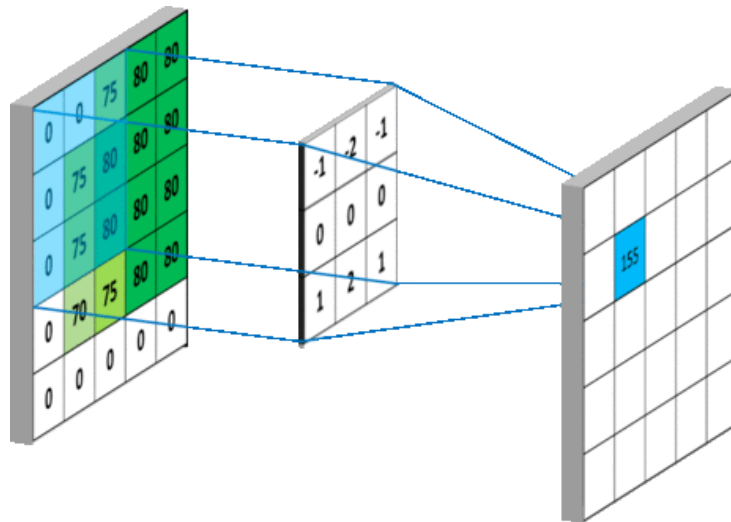


Figure 27: 2D convolution filter layer with an example horizontal edge detection filter kernel⁴

⁴Khandelwal, Aman [2018]. *Convolutional Neural Network*. URL: <https://medium.com/@aman.khandelwal.howrah/convolutional-neural-network-927b20a678d7> [visited on 12/20/2019]

Transposed 2D convolution layer

The 2D convolution layer can also be used to upscale an input image. The difference to simple upscaling is the trainability of the filter kernel, which is also initialized with random values at the start using the LeCun normal filter initializer. This special convolution layer is called a transposed 2D convolution layer.

Activation layer

An activation layer updates the weights of other layers, in this case the convolution layers. It uses a special function to determine the updated weights, examples for this function are the *Sigmoid* function (1), the *Tangens Hyperbolicus* function (2) and the *ReLU* function (3). It is possible in Keras to integrate the activation functionality into the convolutional layers themselves, but leaving it as a separate layer gives the possibility to add other layers in between.

Batch normalization layer

A batch normalization layer is used to control the activations of other layers. It normalizes the individual values in such a way, that the mean activation is close to 0. This averaging is performed on multiple input tiles at once. Depending on the number of tiles, also called the batch size, the effect of the batch normalization can vary. A benefit of batch normalization is a lower risk of overfitting (see chapter 4.7.1), while simultaneously allowing for grater steps in the weight updating to be used and thus increasing the rate in which the model adjusts itself to the data. This leads to a reduced amount of time that is required to achieve the same accuracy as it would be without batch normalization [Ioffe and Szegedy, 2015].

Dropout layer

Another method to reduce overfitting is presented in dropout layers. It removes in each epoch random weights, which forces the other weights to not solely depend on a few strong weights, but to learn to predict features on their own. The idea behind this concept is that if from few very strong weights one is incorrect, the whole prediction is effected. For training purposes it is better if the weight values are more spread out. The dropout layer can also be normalized, so that the total sum of all weights stays the same after the removal, it is then called an alpha dropout layout. Batch normalization and alpha dropout have similar effects in terms of reduced overfitting, so usually only one of the described methods is used.

Pooling layer

In order to reduce the image size but to also keep the feature information, pooling operations can be used. Out of a mask that slides over the input image, one pixel is picked and transferred to the output image. Usually the mask has a size of 2×2 , which means one out of 4 pixels are transferred, thus decreasing both image dimensions by a factor of 2 and decreasing the total size by a factor of 4. A commonly used method is *Max Pooling*, which copies the pixel with the highest value from the mask. An other method is to pick the average value (Average Pooling), but since the goal is to keep the relevant feature information, the highest value is chosen in most cases. Figure 28 illustrates a Max Pooling operation with a 2×2 mask.

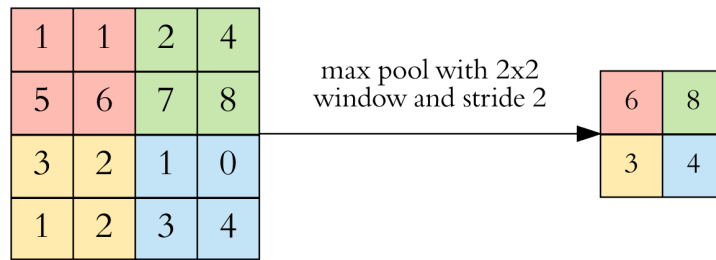


Figure 28: Max Pooling operation in a 4×4 grid with a pooling window size of 2×2 and stride of 2^5

Concatenation

Concatenation layers transfer features from one layer to another. This is useful if features from the start of the network are to be mixed with further processed features in another part of the network.

Most model architectures that are used for image segmentation consist of two parts, one is called the encoder, the other is called the decoder. The first half of the network is the encoder which utilizes convolutions and pooling operations to decrease the dimensions of the input image while keeping the important features, thus increasing their weight compared to the background information. This decrease of the dimension in image-space increases the dimensionality in feature-space, which means multiple versions of the reduced data are stored in additional image dimensions. The output of the encoder part can be coupled with some additional layers to create a prediction that states to which class the input image most probably belongs. It can either be a binary classification, for example dog versus cat, or contain a multitude of different classes, like car, bicycle, road sign and pedestrian.

If the desired output is not only a label that states to which class of several available classes the input image belongs to, but an image describing in each pixel which class this particular pixel most probably belongs to, a decoding part is added to the encoder. The decoder increases the image dimensions up to the original dimensions while integrating the extracted features. This results in a segmentation map, which shows for each pixel its probability to belong to a certain class. In case of only two available classes, the output image has only one dimension, for each additional class an additional image dimension is generated. To determine which class a certain pixel most likely belongs to, the values of said pixel in all dimensions are compared against each other and the highest probability score is presumed as the correct class. Examples of architectures that have proven to be working well for image segmentation tasks are *ResNet* [He, Zhang, et al., 2016], *Inception Net V1* [Szegedy et al., 2015], *AlexNet* [Krizhevsky et al., 2012] and *U-Net* [Ronneberger et al., 2015]. Figure 29 shows the network architecture of AlexNet, which is used to predict the class of individual images out of 1000 available classes.

⁵Karpathy, Andrej [2019]. *Convolutional Neural Networks for Visual Recognition*. URL: <http://cs231n.github.io/convolutional-networks/> [visited on 12/20/2019]

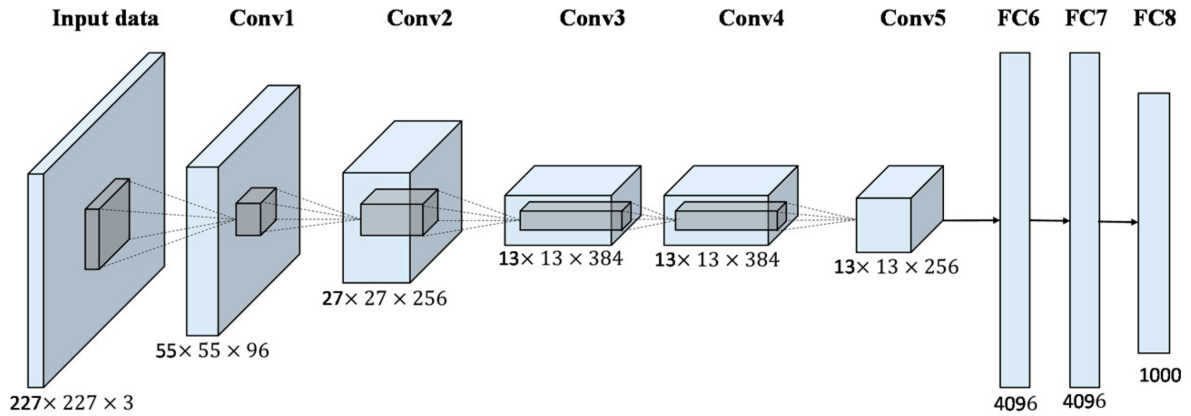


Figure 29: Visual representation of the AlexNet model architecture. Its primary focus is classification of remote sensing image scenes with high spatial resolution[Krizhevsky et al., 2012]

4.7 Working principle of Convolutional Neural Networks

During training of a CNN, the input data is fed through the network and output predictions are generated. A small portion of the training data is used to evaluate the performance during training, this is the so called validation data set, and is usually around 20% of the training data. The validation data is not to be confused with the test data, which is split from the original data before the training itself begins. During training and with the help of the validation data, the individual weights of the model are updated iteratively so that the predictions are getting better and better. Once all the input data has been processed, one epoch of training is complete. At the start of every following epoch, the weights of the previous epoch are kept in the model, so the training process does continue and the weights are iteratively adjusted.

4.7.1 Loss function for error determination

In each training step the weights are adjusted by using a function that determines the error of the prediction, also called the *loss* function, to minimize the error of the prediction. Commonly used loss functions are the *Mean Squared Error* (5) and the *Binary Cross Entropy* (6). Both equations use y_i as the correct class, and \hat{y}_i as the prediction, while iterating over all N pixels that are used for the evaluation.

$$L = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (5) \quad L = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (6)$$

To compensate for large class unbalances, the loss function can be weighted, so that an error in one class has more impact than an error in the other class. A typical factor for custom weights is the direct proportion of pixels that belong to each class in the training data set. Equation (7) shows the formula of the weighted binary cross entropy, which uses the additional terms w_1 and w_2 as the individual weights of the classes.

$$L = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i)w_1 + (1 - y_i) \log(1 - \hat{y}_i)w_2] \quad (7)$$

The loss function is applied to both the training data set as well as the internal validation data set. This results in two different loss values, from which only the validation loss is used to update the weights. The training loss however can be used to determine the epoch at which the training can be terminated in order to get the best performing model. If the training is performed too long, the model learns to memorize the individual features of the training data set rather than to recognize features in general. This behavior is called overfitting and is described in a very simple way in figure 30. The optimal model is found in the center of the figure, with the left part being trained to short, and the right part being trained too long.



Figure 30: Simple visual representation of overfitting by approximating points with a curve of different complexity. ⁶

In order to determine the epoch after which overfitting occurs, the training loss can be compared against the validation loss. At the start of training, both loss values should be about equal and go down with every epoch of training, until the model is trained optimally. Once it starts to fit to the training data too well, the validation loss starts to rise, this is the time when the training should be terminated. Figure 31 shows the typical behavior of the training and the validation loss curves over time. Depending on the amount of training data as well as its diversity, the functions can become steeper or flatter with the termination point shifting to the left or to the right.

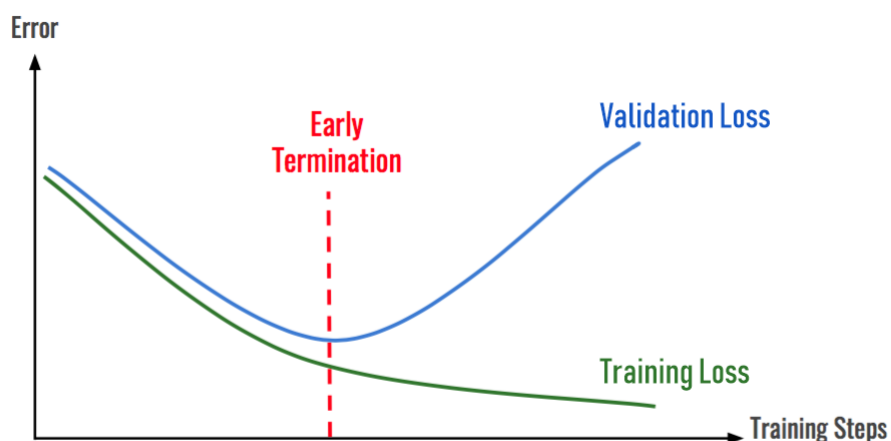


Figure 31: Graph that shows the connection between training and validation loss and the optimum point to stop training in order to avoid overfitting to the data, marked as red line. ⁶

⁶Despois, Julien [2018]. *Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning*. URL: <https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42> [visited on 12/20/2019]

Overfitting can be reduced by sophisticated data augmentation methods to increase the size of the training data set as well as certain model layers which have a regulating effect like the Batch Normalization layer. Given enough epochs, the phenomena does still occur, but using these steps there are more epochs available for training without overfitting, giving the model a chance to train to a higher overall accuracy.

4.7.2 Back Propagation and weight updating

The weights in CNNs are updated by certain functions called *optimizers*. A common method in which optimizers work is called backpropagation. The new value for each weight is calculated in a way so that the prediction would be closer to the actual correct values, which minimizes the error of the model. The weight updates are then backpropagated through the network to be included in the next step of training. This is performed over and over again for every single input tile and every single epoch. Typically used back propagation functions are *Momentum* [Rumelhart et al., 1986], *RMSProp* (Root Mean Square Propagation) [Hinton, 2019] and *Adam* (Adaptive Momentum Estimation) [Kingma and Ba, 2014], which is an improvement of RMSProp. The Adam optimizer is the optimizer of choice for this thesis, [Ruder, 2016] recommends it as the best optimizer in terms of speed and performance. The working principle of the Adam optimizer is described in equation (8). w^t represent the current weight parameters, L^t the loss function, in this case the weighted binary cross entropy. t indicates the current epoch of training, ϵ is a very small factor which prevents a division by 0. β_1 and β_2 are factors which introduce an artificial forgetting.

$$w^{t+1} = w^t - \mu \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon} \quad (8)$$

$$\text{with } \hat{m}_w = \frac{m_w^{t+1}}{1 - \beta_1^{t+1}} \quad \hat{v}_w = \frac{v_w^{t+1}}{1 - \beta_2^{t+1}}$$

$$\text{and } m_w^{t+1} = \beta_1 m_w^t + (1 - \beta_1) \nabla_w L^t$$

$$v_w^{t+1} = \beta_2 v_w^t + (1 - \beta_2) (\nabla_w L^t)^2$$

4.8 Model Implementation

The architecture chosen for storm damage assessment on satellite and ortho data is called U-Net [Ronneberger et al., 2015], the name is derived from its shape which resembles the letter U. U-Net was originally designed at the university of Freiburg for biomedical image segmentation by Olaf Ronneberger and was able to achieve a first place at the ISBI Cell Tracking Competition 2015 [ISBI 2015, 2019]. One of the benefits of U-Net is the fact that it can be trained with a limited amount of training data, which makes it suitable for the application on the satellite images. Another benefit is that the output is already a segmentation map with the same size as the input images, so no custom decoder has to be created. U-Net has proven in other studies to work particularly well not only on biomedical,

but also on remote sensing data. Figure 32 shows a graphical representation of the original architecture of the model as it was invented by Ronneberger. The model that is used in this thesis has some slight variations that are oriented to boost the performance on the given data set. Starting on the left side, the input image and label tiles are fed into the network, which processes the data while updating its weights and generates a prediction image on the right side.

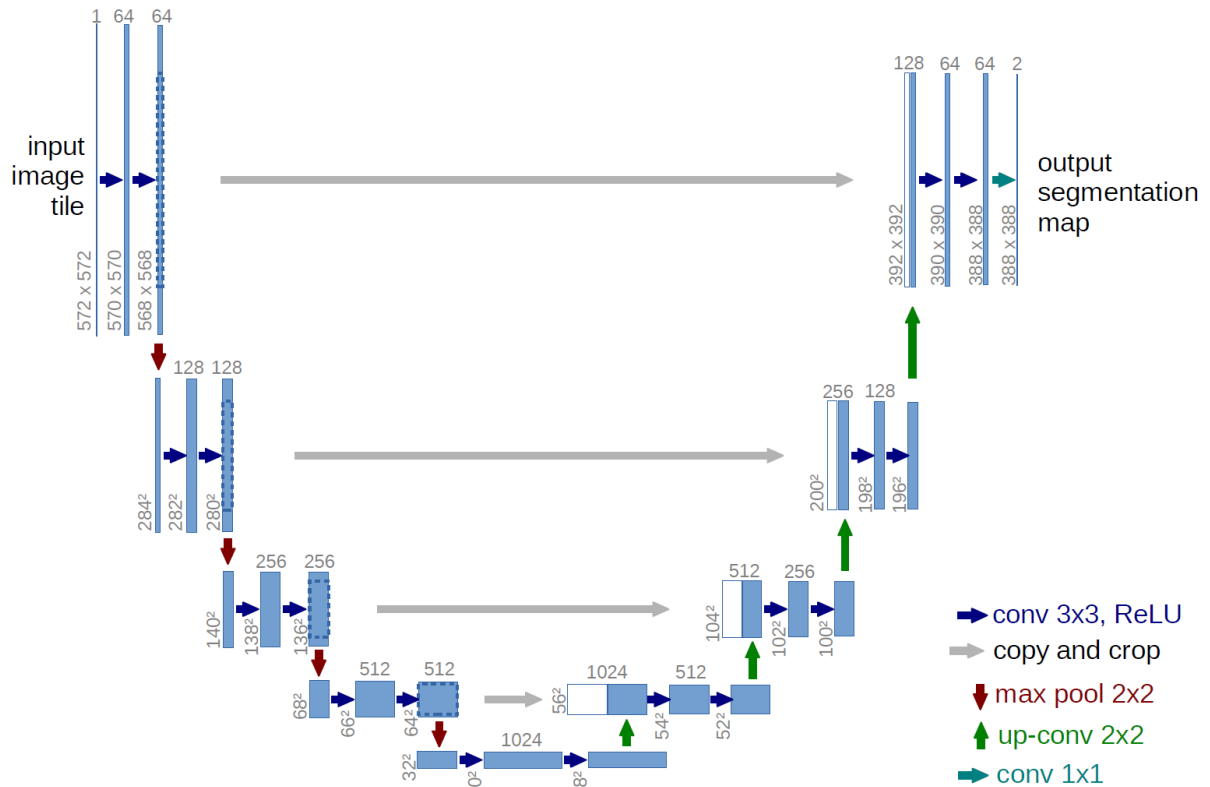


Figure 32: Diagram of the U-Net architecture as it was invented by [Ronneberger et al., 2015]. The exact architecture that is used in this thesis does have slight variations

U-Net consists on a very basic level of a decoding part and an encoding part with concatenations in between. The decoding part which is displayed on the left half of figure 32 gradually decreases the tile size over several steps with a factor of two in every step, while simultaneously increasing the dimensions. This procedure is used to extract certain features in the input image by focusing on the features themselves while losing the exact position of said features. The right half of the model is called the decoder, its purpose is to increase the dimensions again gradually until the original tile size is reached. To enhance the output in terms of accuracy, since the spatial accuracy is quite low at the bottle neck in the middle, concatenation steps are introduced which copy the extracted features of every decoding step to its encoding counterpart. This results at the end in a highly detailed segmentation map that shows in each pixel its likelihood to belong to a certain class. The implementation used in this thesis is based on the implementation of Zayd Hamdi [Hamdi et al., 2019] with improvements regarding the exact composition of the individual layer blocks. The number and composition of the individual layer blocks, the parameters for functions as well as the tile sizes are called hyperparameters. Hyperparameter tuning is an important step to get an optimal model configuration and requires often a lot of time and many tests. The model used by Hamdi operates with

a tile size of 256×256 pixels with the four image bands red, green, blue and near infrared. To allow for an increased learning rate and to decrease the time required for training, as well as to decrease the effect of overfitting, certain adjustments in the model composition are made like the introduction of Batch Normalization layers. Each step on the decoder and encoder path as seen in figure 32 consists of multiple layers which are stacked on top of each other. The exact order and composition of the individual layer blocks can be modified according to the input data, the layers used in this thesis for the encoder and decoder are listed as follows.

Encoder block

1. 2D Convolution Layer
2. Batch Normalization Layer
3. Activation Layer
4. Alpha Dropout Layer
5. 2D convolution Layer
6. Batch Normalization Layer
7. Activation Layer
8. 2D Max Pooling Layer

Decoder block

1. Concatenation Layer
2. Alpha Dropout Layer
3. 2D Convolution Layer
4. Activation Layer
5. 2D Transposed Convolution Layer

Each encoder and decoder block is used multiple times in a row, the amount is specified by the size of the blocks hyperparameter. At the end of the decoder a final 2D transposed convolution layer is added to shape the output again to the input size. The complete model architecture is listed in appendix C, the optimal hyperparameters are listed in appendix A.

4.9 Performance evaluation for Convolutional Neural Networks (CNNs)

The output of U-Net is a segmentation map that has the same dimensions as the input tiles but with only one band, since only two classes are used. Each pixel of the segmentation map has a value between 0 and 1 and represents its probability to belong to either of these classes. To convert this probability to a distinct binary value, a threshold is introduced. By adjusting the threshold, either less or more pixels will be assigned to the classes, which gives the threshold a controlling factor that can be used to adjust the segmentation map to certain scenarios. To evaluate the resulting binary mask, different metrics are considered.

4.9.1 Confusion Matrix

With the help of a confusion matrix, the amount of pixels that are correctly or incorrectly labeled can be determined. For a binary segmentation map with two classes, one class is called positive while the other class is called negative. The positive class is the class that usually is in the focus, in this case the forest damage, and the negative class is called the background, in this case everything else but forest damage. In the confusion matrix the prediction and the ground truth are compared against each other, and there are four different states possible:

- **True Positive:** A pixel is both in the prediction and the ground truth labeled as positive
- **False Positive:** A pixel is predicted as positive but is negative in the ground truth
- **False Negative:** A pixel is predicted as negative but is positive in the ground truth
- **True Negative:** A pixel is both in the prediction and the ground truth labeled as negative

Table 1 shows the confusion matrix for binary classification.

	Prediction A	Prediction \bar{A}
Ground Truth A	True Positive (TP)	False Positive (FP)
Ground Truth \bar{A}	False Negative (FN)	True Negative (TN)

Table 1: Confusion matrix for two classes

The confusion matrix is calculated as sum over all pixels in the image scene. The four possible states can be used in other metrics to further evaluate the performance of the model. In general it is desirable to maximize the values TP and TN, while minimizing FP and FN. Depending on the image scene and the task, one matrix entry might be of greater importance than another. By using the threshold, the distribution of pixels that fall into these categories can be adjusted.

4.9.2 Intersection over Union

The intersection over union, also called the *Jaccard-Coefficient* after the botanist Paul Jaccard, is an evaluation method to determine the similarity between two sets of data. It is a score that is calculated either for the positive or for the negative class. In case of the positive class, it divides the sum of all pixels that are both labeled as positive in the prediction and the ground truth by the sum of all pixels labeled as positive in both the prediction and the ground truth. The score of the intersection over union can range between 0 and 1, with 0 being no similarity at all, and 1 being a perfect concordance. The formula of the intersection over union is shown in figure 33.


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 33: Visual representation of the calculation of the intersection over union score,⁷

⁷Rosebrock, Adrian [2016]. *Intersection over Union (IoU) for object detection*. URL: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> [visited on 12/20/2019]

4.9.3 Accuracy

The overall accuracy of a prediction is the total amount of pixels that are classified correctly and is in the range between 0 and 1. It can be directly derived from the confusion matrix and fits the equation (9).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

The accuracy is most suited for data sets that have an equal distribution of the two classes, if one class outweighs the other, it is more likely that a lot of pixels will correctly be classified which gives the prediction an overall high accuracy. In case of the class unbalancing of 95% in favor of no damage in the satellite data set, the accuracy is almost guaranteed to have a very high value, therefore it is not an ideal metric to evaluate the performance on this specific data set.

4.9.4 Precision and Recall

With the help of the precision, the proportion of pixels that are labeled as one class in the ground truth in relation to the pixels that are classified as the same class in the prediction can be found. It describes the proportion of correctly classified pixels in the prediction that all belong to one class. The recall, also called sensitivity or true positive rate (TPR), on the other hand describes the proportion of pixels of one class in the ground truth to the total amount of pixels from that class in the prediction. It gives information about the percentage of pixels that belong to one class in the ground truth that are also found in the prediction. Another commonly used metric is the false negative rate (FNR) which is directly derived from the true positive rate via the formula $1 - recall$. It basically tells how many pixels were incorrectly predicted as negative in relation to the total amount of pixels that are predicted and labeled as positive.

$$precision = \frac{TP}{TP + FP} \quad (10)$$

$$recall = \frac{TP}{TP + FN} \quad (11)$$

4.9.5 Receiver operating characteristic

Another way to measure the performance of a binary classification method is the Receiver Operating Characteristic (ROC). It is a two dimensional plot with the false positive rate on the x axis and the true positive rate on the y axis. The false positive rate (FPR) follows the formula $1 - specificity$, the specificity, also called the true negative rate (TNR) is described in equation (12). It is an indicator for the proportion of pixels that are classified as negative in the ground truth to the pixels that are classified as negative in the prediction.

$$specificity = \frac{FP}{TN + FP} \quad (12)$$

To get multiple values for both axes, the threshold which transforms the probability segmentation map into a binary segmentation map is adjusted step wise from 0 to 1. The smaller the steps are chosen, the smoother the resulting ROC curve will be. An example for different ROC curves is shown in figure 34 with an almost perfect curve in blue, a mediocre but expected curve in green and a random guess of the prediction in black.

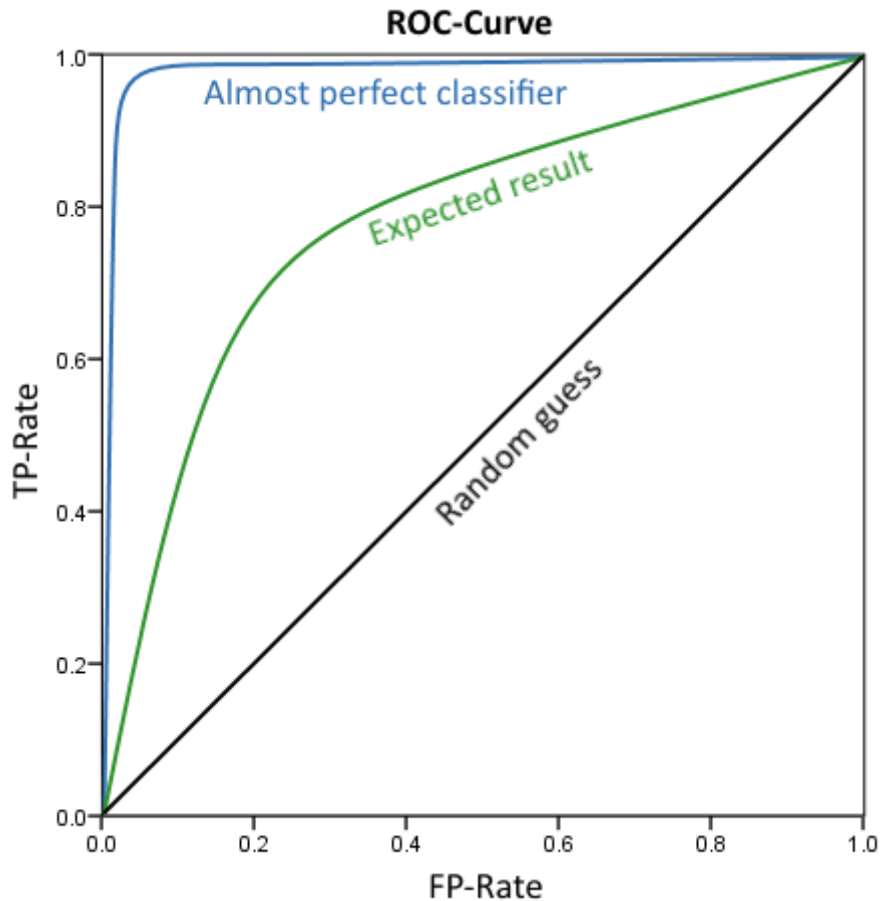


Figure 34: Receiver operating characteristic with three different possible curves

4.10 Approach to find the optimal model

Since it is not always clear in the beforehand what effect changing a certain hyperparameter might have, a lot of estimating and guessing has to be done in order to find the best model configuration. For large scale model implementations usually a random search algorithm is used, which picks random values for the hyperparameters, trains the model and then assesses its quality. This is done numerous times in a row, often times with the same model architecture multiple times in parallel. This approach takes a lot of time and computational power, which is both not available, so a simpler iterative approach is used in this thesis. To find the optimal value for every parameter, one parameter at a time is altered to see if the model improves or not. The order in which the hyperparameters are tested is important, since different parameters can influence each other. The first parameter to be tested is therefore the one with the most likely highest impact on the results, followed by all other parameters in decreasing significance. For every single hyperparameter multiple values are used and the predictions are compared against each

other until the optimum is reached. It is still possible that this method does not result in the combined optimum for every parameter, which is very hard to determine. The tests are only performed on satellite data, since an already sufficient set of parameters for ortho images is presented in the by [Hamdi et al., 2019]. To ensure a consistency between the results during the tests, all random values that are used for image augmentation, filter initialization and other layers are initialized with the same seed. In total the following hyperparameters and parameters are tested:

1. Tile size
2. Number of blocks and number of filters in each block
3. Learning rate
4. Additional noise

The performance visualization software *TensorBoard* [Google, 2019b] is used to see the training and validation loss as well as the training and validation Intersection over Union (IoU) score at each epoch while the training is done. TensorBoard can show multiple models at once to give a direct comparison between the current training session and previous ones. An additional feature is a detailed graphical representation of the model configuration, which can be used for debugging purposes as well as to visualize the model.

4.10.1 Testing different tile sizes

The first step in hyperparameter tuning is to find the optimal tile size, which is ideally a number that is dividable multiple times by 2, since in every down sampling step of U-Net the tile size gets halved, and odd values of the tile size can cause issues. Therefore only powers of 2 are used, which leaves 128×128 and 256×256 pixels as valid options, smaller or larger tile sizes are seldom used in deep learning. To compare the two tile sizes, five models with different hyperparameters that are chosen on a broad spectrum to maximize the meaningfulness of the test, are trained on both tile sizes respectively. The models are trained in 15 epochs with a batch size of 80 and a learning rate of 0.001, the resulting IoU scores on the validation data set are presented in figure 35. The final values of the 15th epoch are presented in table 2 along with the average training time in seconds for every epoch. Figure 36 acts as a legend for figure 35 and shows the last epoch as close up so the individual metric curves can be distinguished.

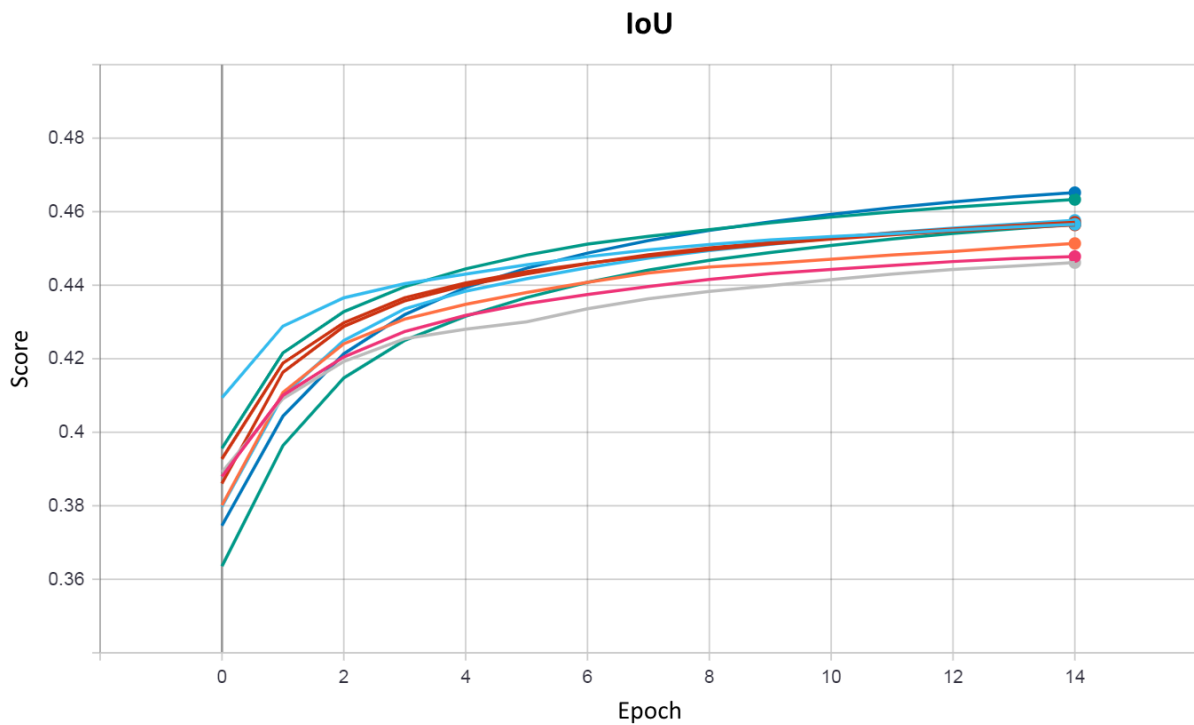


Figure 35: Different **IoU** scores over time for comparing tile sizes. The visualization is done using the software TensorBoard. The epochs are shown on the x axis, the **IoU** score on the y axis. A close up is presented in figure 36, which also acts as a legend.

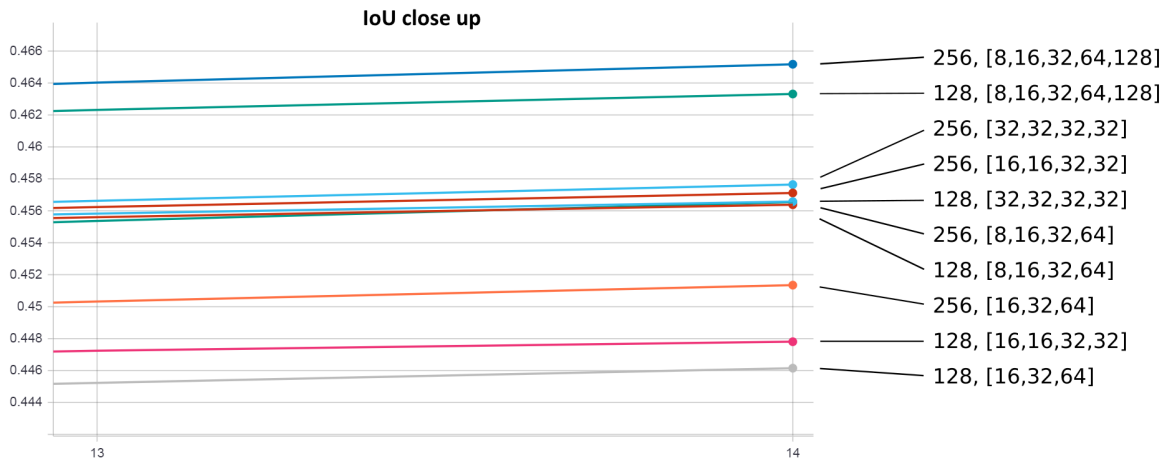


Figure 36: Close up of the different **IoU** scores over time for comparing tile sizes. The epochs are shown on the x axis, the **IoU** score on the y axis. Each line in the plot is marked with a label that shows which particular block configuration was chosen for that training session. The labels can be compared with table 2.

Number	Learning rate	Blocks	128 × 128		256 × 256	
			IoU	Seconds	IoU	Seconds
1	0.001	[32, 32, 32, 32]	0.4573	290	0.4588	475
2	0.0015	[8, 16, 32, 64]	0.4566	260	0.4574	445
3	0.001	[16, 32, 64]	0.4461	370	0.4512	530
4	0.002	[16, 16, 32, 32]	0.4481	310	0.4577	420
5	0.001	[8, 16, 32, 64, 128]	0.4632	410	0.4658	610

Table 2: IoU scores at epoch 15 for comparing tile sizes

Although the results are fairly close when comparing the different tile sizes, 256×256 pixels is in every test the superior size and will therefore be used in further training steps. There is a significant difference in the time that is required for each epoch, which leaves the argument that it may be better to train on 128×128 pixels, since more epochs can be fit in the same time. Figure 35 suggests that the increase of the IoU score is already quite low at epoch 15, and more epochs do not have a big effect anymore. Adding more epochs can also result in overfitting, which makes the model perform worse in the end. The difference in the measured IoU score between the two tile sizes is most likely explainable by the fact that larger tiles contain larger and more continuous features, which makes it easier to distinguish the damage from the background. Smaller tiles make it harder to detect features by their shape, which shifts the focus more to recognition by colors, and thus the model loses a part of its detection capability.

4.10.2 Testing different numbers of blocks and filters

By adding more blocks to the encoder and decoder of the U-Net architecture, the model becomes deeper and is in theory able to extract and predict smaller and less prominent features. This comes however with an increased computation time and it is not guaranteed that the model does actually perform better. Also, increasing the number of filters in each convolution does have a similar effect, but a lot of training time can be saved if this number is kept low in blocks that do not require many filters. The training is done with a learning rate of 0.001 with a batch size of 80 and the tile size of 256×256 . Different block configurations are chosen with the intent to cover a broad spectrum of possibilities, ranging from only three blocks to up to six. In general the assumption can be made that more filters lead to more time required for each epoch, but also the index of blocks which have a high amount of filters plays a big role. The earlier in the architecture, the higher the impact many filters is regarding training time. Figure 37 shows the IoU scores for every epoch and block configuration, table 3 presents the IoU scores at the epoch 15 sorted in descending order by the IoU score. The configurations 1, 8 and 15 require a smaller batch size of 64, otherwise the Tensorflow framework runs out of memory.

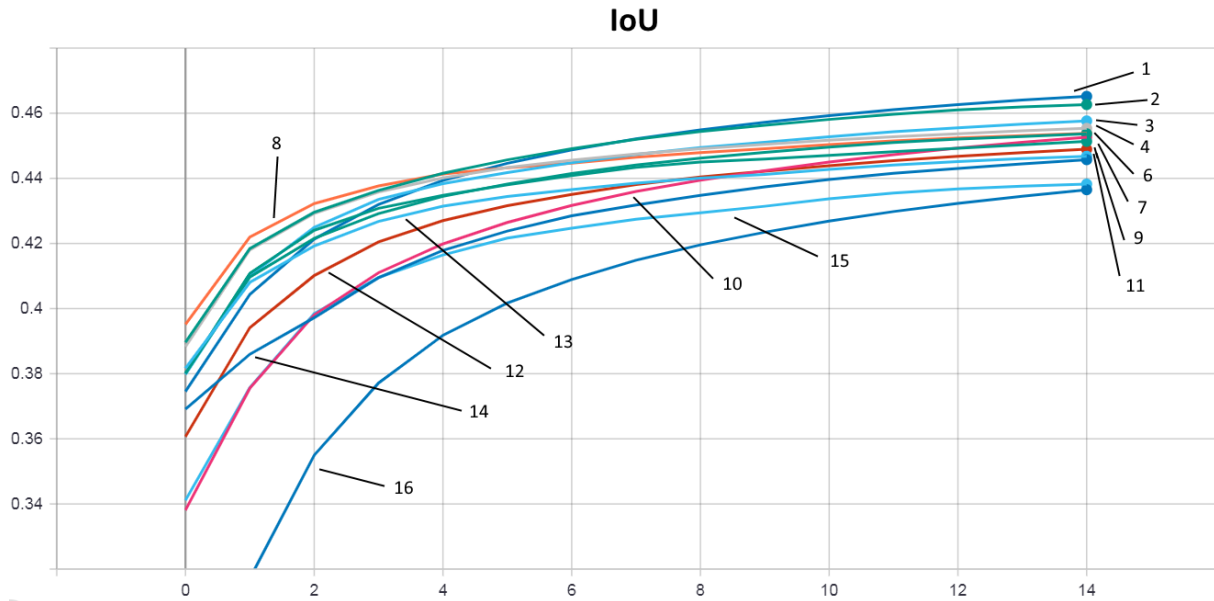


Figure 37: Different IoU scores over time for comparing block configurations. The labeling is linked to the numbers in table 3. The epochs are shown on the x axis, the IoU score on the y axis

Number	Blocks	IoU	Seconds/Epoch
1	[64,64,64,64]	0.4666	880
2	[8,16,32,64,128]	0.4658	610
3	[16,32,64,128]	0.4640	760
4	[32,64,128]	0.4629	830
5	[16,16,64,64]	0.4627	660
6	[32,32,32,32]	0.4576	480
7	[32,32,16,16]	0.4554	430
8	[64,32,16,8]	0.4538	560
9	[16,16,32,32]	0.4537	410
10	[4,8,16,32,64,128]	0.4527	600
11	[16,32,64]	0.4513	530
12	[8,16,16,32]	0.4489	360
13	[16,16,16,16]	0.4468	400
14	[16,16,16,16,16]	0.4457	430
15	[64,32,16,8,4]	0.4382	600
16	[4,8,16,32,64]	0.4365	410

Table 3: IoU scores at epoch 15 for comparing block configurations

The maximum **IoU** score is found with the block configuration of [64, 64, 64, 64] with a score of 0.4666. The runner up configuration [8, 16, 32, 64, 128] reaches a score of 0.4658. Downsides of the first configuration are the large number of filters in blocks which handle large tiles, which increases the required training time a lot as well as forces a lower batch size. Considering these drawbacks, the second configuration is chosen as the optimal block configuration. The idea behind this particular block ordering is to have few filters which are applied to the larger tiles and many filters which are applied to the smaller tiles. All in all this saves computational power and leads to a good detection rate of large as well as small features. This block combination will therefore be used for further training runs.

4.10.3 Testing different learning rates

The learning rate is a hyperparameter that influences how much the individual weights get updated. A low learning rate can make the training very slow and require a lot of epochs to reach a certain accuracy, while a too high learning rate can lead to the model getting stuck on local minima during gradient descent of the parameters and not finding the global minima and thus not training properly. To find the optimal learning rate, different values are used in every training process starting from 0.0006 with an increase of 0.0002 in following training iteration. The resulting intersection over union scores are presented in figure 38, the scores at epoch 15 are shown in table 4.

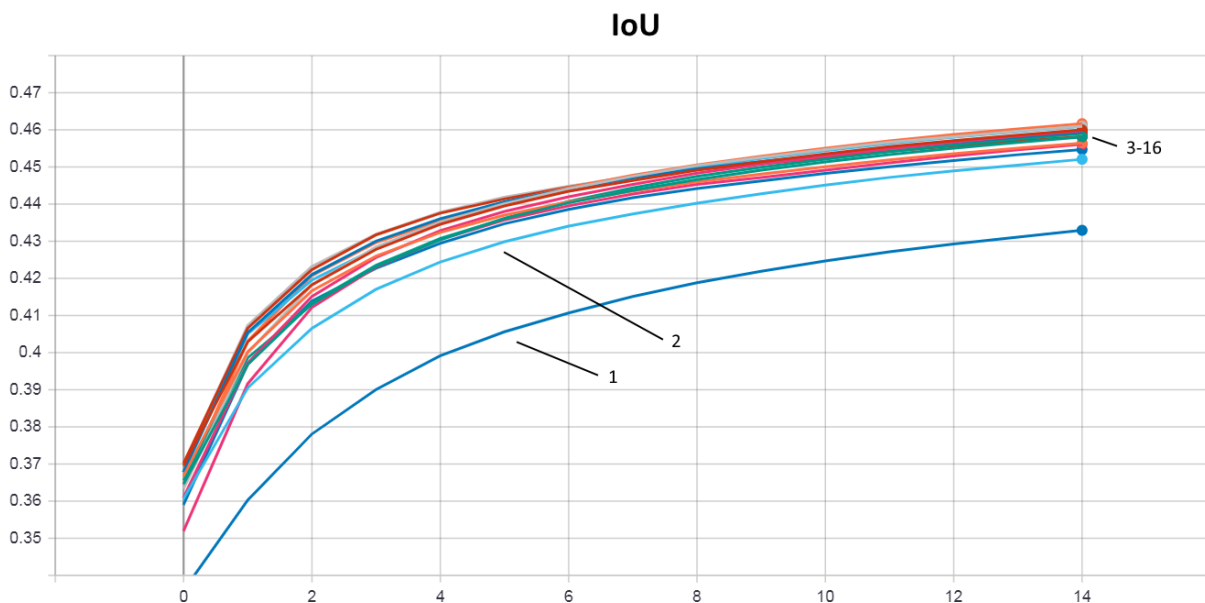


Figure 38: Different **IoU** scores over time for comparing learning rates. The labeling is linked to the numbers in table 4. The epochs are shown on the x axis, the **IoU** score on the y axis

Number	Learning rate	IoU
1	0.0001	0.4329
2	0.0003	0.4521
3	0.0005	0.4581
4	0.0007	0.4564
5	0.0009	0.4600
6	0.0011	0.4598
7	0.0013	0.4610
8	0.0015	0.4594
9	0.0017	0.4609
10	0.0019	0.4590
11	0.0021	0.4617
12	0.0023	0.4581
13	0.0025	0.4561
14	0.0027	0.4587
15	0.0029	0.4547
16	0.0031	0.4567

Table 4: IoU scores at epoch 15 for comparing learning rates

Figure 39 shows a plot of the results with the different learning rates on the x-axis and the IoU score at the y-axis. An approximation using the least squares method with 3 parameters helps to determine the optimal learning rate, which is ideally at the maximum of the interpolated curve.

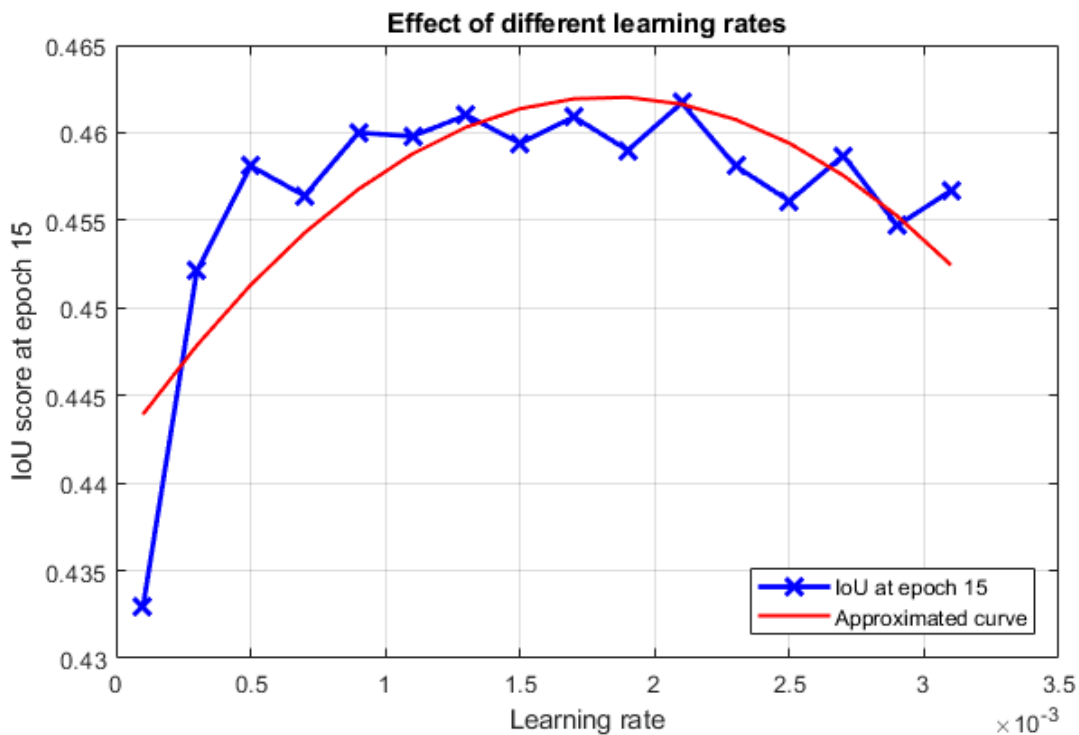


Figure 39: Different IoU scores at epoch 15 plotted against the learning rate of the according model. A approximation using the least squares method helps to determine the optimal learning rate

The optimal learning rate can be estimated to be somewhere around 0.002 at the maximum of the interpolation curve.

4.10.4 Testing the effect of additional random noise

Although technically not a hyperparameter for the training process, the sigma value which is used to add random noise to the input data for data augmentation purposes can also be altered to make the model more or less robust regarding different data. Too high sigma values however decrease the achieved total accuracy, since the data becomes more diverse and each tile is becoming to look too different than the others. To find the highest sigma value that still delivers acceptable results, the predictions of models trained with data that is augmented with different sigma values are compared against each other. In order to still keep original data and not to solely train on augmented data, every tile is used twice, once without color adjustment, and once with adjustment. The results are presented in figure 40 and table 5. Although the models seem to perform best when trained on a sigma value of 0.5, 1.5 is chosen as the desired value. There is always some randomness within the training process, but it seems that up to 1.5 the performance is rather good, and in general more noise does result in a robuster model. Since the goal is to not only perform well on the given test data set, but also on future data, choosing the higher sigma value seems to be the reasonable choice. It is interesting to note that training without the addition of random noise results in a lower performance. Adding the noise helps to diversify the data and increase the overall robustness of the model, which in terms does perform better also on the test data set.

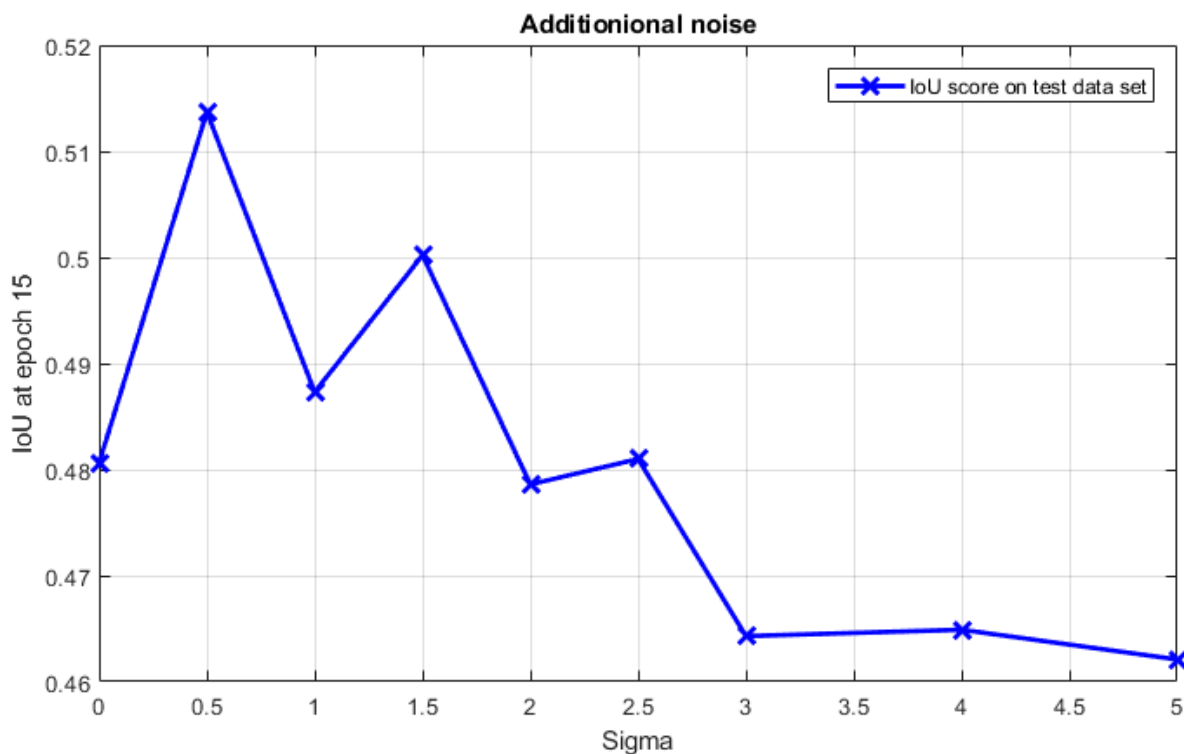


Figure 40: Different IoU scores over time for comparing the effect of additional noise intensities

Number	Sigma	IoU
1	0.0	0.4806
2	0.5	0.5137
3	1.0	0.4873
4	1.5	0.5002
5	2.0	0.4786
6	2.5	0.4810
7	3.0	0.4643
8	4.0	0.4649
9	5.0	0.4621

Table 5: IoU scores at epoch 15 for comparing different noise intensities

4.11 Training of the models with the optimal hyperparameters

Once the optimal hyperparameters are determined, a model is trained with these parameters on satellite data, and another model with the optimal parameters presented by Hamdi trained on the ortho data. Since there is no limit on how many epochs can be trained, a rational stop is either if overfitting occurs or the prediction score does not significantly improve anymore. Another approach is to train a large amount of epochs while saving the weights at each epoch and afterwards evaluating each epoch on the independent test data. The best performing weights are then kept as the final model.

When training the model on satellite data, two sets of labels can be used. Up to this point, only the original satellite labels were considered, but it is also possible to use ortho labels on the satellite images. With their much higher accuracy, the ortho labels show damage in places at which the human operators did not detect any when creating the satellite labels. It is therefore possible that the model does perform better when trained on the ortho labels instead of the satellite labels. To proof this hypothesis, the training is performed on both sets of labels with 100 epochs each and the exact same hyperparameters. When the training is complete, the best performing weights are compared against each other.

Training on the ortho data takes a lot more time, the amount of tests that can be performed is thus limited, since meaningful results often times do not occur until 10 epochs or more are finished. The chosen strategy for training is in this case to stick with the optimal parameters presented by Hamdi, with the addition of data augmentation and Batch Normalization layers. In theory the training should be aborted once overfitting is detected due to an increase of the validation loss, but it might very well not be detectable during training due to the increased training data set and the new regularization layers. Therefore a fixed number of 50 epochs is chosen, after which the training is seen as complete. Due to sudden illness which did not allow an analysis of the trained model, the number of epochs did actually rise to over 200 with a total training time of over 5 days. For the training on the ortho images, only the ortho labels are used and not the satellite labels, since there is most likely no benefit in using less precise labeling.

4.12 Application of transfer learning on the data

Since training of complex and very deep models unlike U-Net from scratch is very computation intensive and requires specialized hardware and a lot of time, a common practice is to use already pre-trained weights for large models. These models are trained on millions of images like the ImageNet data set [ImageNet, 2016] and deliver very good results for their respective tasks, but are also quite large in their file size and comparatively slow in the execution. Usually these models are trained to predict the class of an entire image tile and are used for instance for object detection or face recognition. Since most pre-trained models therefore do not implement a decoding part, a custom decoder can be added to create a segmentation map as output of the model. The custom decoder has some weights of its own which need to be trained, but all weights from the pre-trained model are frozen which means they will not be updated. This significantly reduces the amount of trainable parameters and the required training time, which allows for short training periods to deliver good results. A common practice is also to not freeze every layer but to leave some of the last layers of the encoder to also be updated, so that the encoder adjusts itself better to specific features of the new data set. This is however not done due to the limited amount of GPU memory. If necessary, the last part of the model, the so called *head* can also be removed and a custom *head* added. The head is usually a set of fully connected layers that shapes the output into the desired dimensions. Commonly used models for transfer learning are VGG16, VGG19, ResNet, ResNetV2, InceptionV3, DenseNet and more [Chollet, 2019]. In this thesis, the model VGG19 with its weights pre-trained on the ImageNet [ImageNet, 2016] data set is used together with the decoding part of U-Net. The VGG19 model expects the input data to be in the RGB format with a color depth of 8 bits. The combined model is trained 15 epochs on the first three bands of the ortho data set with only the decoder weights set as trainable, the already trained encoder weights are frozen, which means they are not altered. The satellite data however comes in 16 bit integer values, and the data is very narrowly distributed as shown in figure 9. When converting the images to 8 bit integers, a lot of information is lost, and the band coverage of the resulting image is far from ideal for the VGG19 encoder. Tests have shown that the predictions that result from this transfer learning configuration on the satellite images of this study are not useful. Therefore the VGG19 encoder coupled with the U-Net decoder is applied only to ortho data.

4.13 Integration of the models into a Geoinformation System (GIS)

An easy way to integrate Deep Learning into GIS is offered by the ArcGIS platform with the software ArcGIS Pro. Within its functionality spectrum a toolbox to apply CNN models on raster data can be found. It can be used to create predictions on raster data of arbitrary size, with a segmentation map of the same size as output. Tiling of the input data as well as merging of the output data is done by the software, the only part that has to be implemented by the user is a custom inference function. This function tells ArcGIS Pro how to handle both the model and the data, and how to perform the predictions. Inside the inference function the threshold is applied, its value is adjustable from outside the code via the toolbox. A model definition file in the Javascript Object Notation (JSON) file format is used to store all necessary parameters like the tile size, the path to the model and the Deep Learning framework. This allows the deployment of the trained CNN on basically any machine, as long as it is capable of running the predictions, which is in the case of U-Net an easy to match requirement. To speed up the prediction

process, batch processing of the data is possible, and a padding around the input data can be set so no predictions are performed at the edges of the data, which is occasionally useful. If required, the prediction can be performed on a server and the results automatically downloaded to the machine. To make the usage easier, running the toolbox is done via a graphical user interface that accepts custom parameters that are defined in the inference function like the threshold as well as a batch processing size and which GPU to use for the prediction.

4.14 Post processing using GIS tools

The result of the toolbox that uses the custom inference function is a binary segmentation map that can be further processed with GIS tools. For this task several different functions are stringed together in a model builder, which is a graphical representation of a chained list of functions. In this thesis a custom toolbox using the model builder and the following processing steps is created.

1. The prediction is created using the trained model and the input data. The input values are the data in raster format, the model definition file, the threshold, the batch size and optional padding. Output is a segmentation map containing binary pixel values.
2. The next step is to smoothen the edges of the predicted damage, since a lot of small artifacts are generated. A median filter is used that slides over every pixel and sets the center pixel of the filter to the median value of all surrounding pixels. Possible input values for this step are the filter type as well as the filter size.
3. Before the next step can be applied, a conversion of the data type of the filtered prediction from boolean to integer is performed.
4. In the next step a conversion from raster data to data in vector format is performed with function that creates polygons from the input raster data. The vector format allows for an easy calculation of the size and scope of damage patches, as well as other values like the distance between neighboring patches.
5. The model prediction creates not only the desired output, but also a lot of small artifacts which are not resolved by the median filter. These artifacts are mainly very small polygons and small holes inside larger polygons. With the help of a function, these small features can be eliminated. As default value a minimum size of 10 m^2 is chosen, which means every feature with an area less than 10 m^2 is removed. The exact threshold for this operation can be set as input parameter in the toolbox.
6. To store the processed polygons, they are converted to a new data type called feature class. Input parameters are the desired name and location of the file that is to be generated.

Figure 41 shows the ArcGIS Pro Model Builder, every list in the enumeration corresponds to one rectangular box in the figure. The round objects are parameters, if marked with a P they are user defined. The input of the data is shown in the top left, the output in the bottom right.

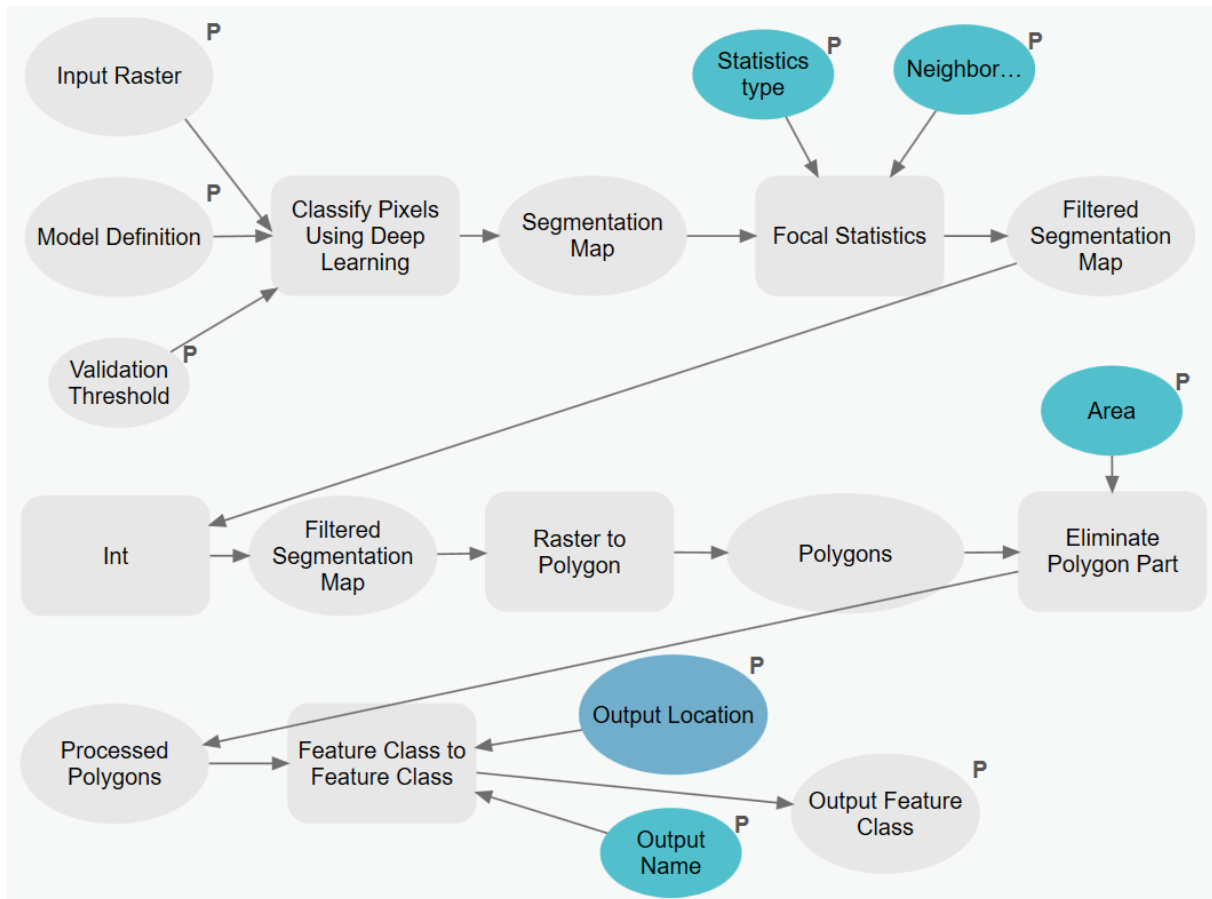


Figure 41: Composition of the custom toolbox in ArcGIS Pro using the model builder. Each round object represents a parameter or object, if marked with the letter *P*, it is a required user input parameter. Rectangular shapes represent the different operations

The graphical user interface that is automatically generated for the complete toolbox is shown in figure 42. The first parameter specifies the raster image that is used for the prediction which can be of any supported raster format and size. The model definition file tells the toolbox the location of the trained model as well as the location of the inference function and the used Deep Learning framework, in this case TensorFlow. The tile size must be specified in the definition file and can not be integrated in the graphical user interface. By analyzing the inference function, the parameters threshold, padding and batch size are added to the toolbox window. The median filter that is used for smoothing of the prediction can be adjusted in size and statistics type, by default it is a circle with a radius of 1 pixel. In order to fill tiny holes in the prediction and to remove small damage patches, an area can be defined that acts as a threshold for removal of these artifacts. Finally the file name and location of the finished prediction are selected. If any of the parameters are filled with incorrect values, for example polygons instead of a raster input, the toolbox will raise an error and refuse to start. A progress bar at the bottom shows the current state of each individual processing step, most of the execution time is spent during the prediction.

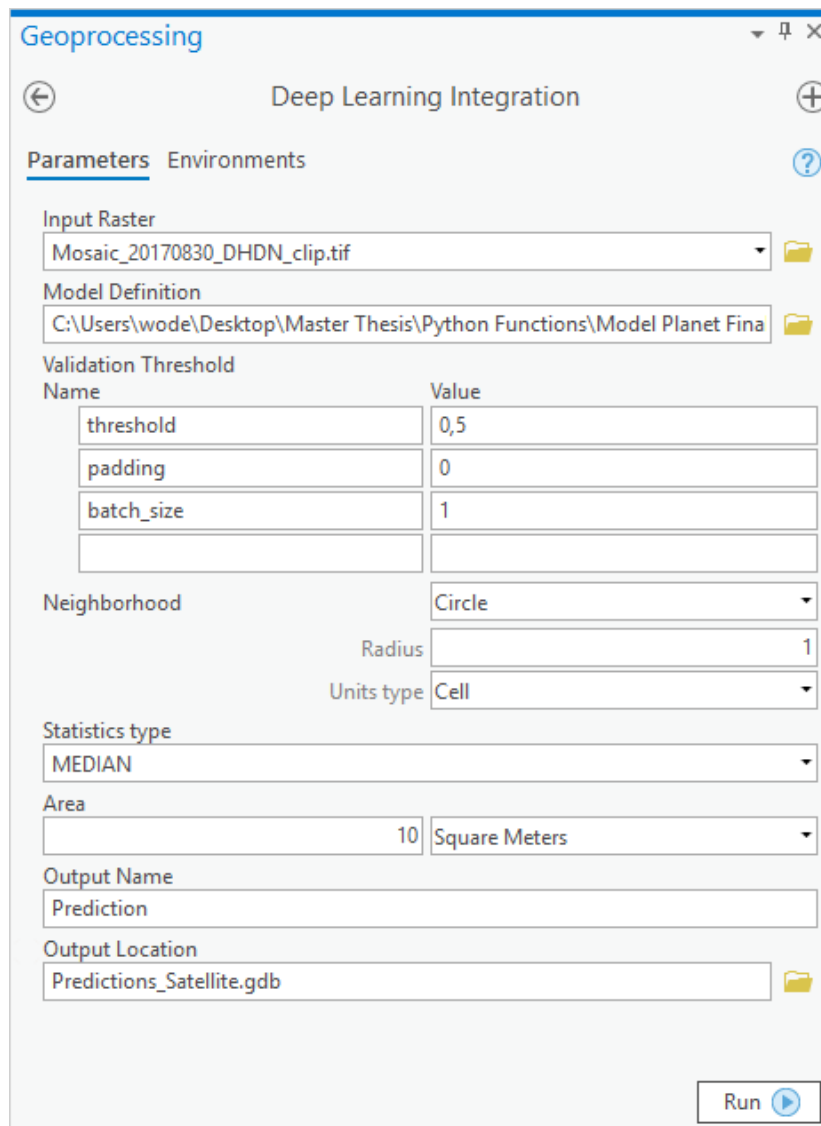


Figure 42: Graphical user interface of the toolbox with default values for the required parameters.

5 Results

5.1 Evaluation of the models

5.1.1 Performance evaluation for satellite images with mixed labels

Training on the satellite images is performed twice in total using the two different sets of labels with 100 epochs. Each individual epoch is used to create a prediction on independent test data and to calculate the *IoU* coefficient. The test can be performed on both sets of labels, which results in the following four predictions:

- Model trained on **satellite** labels, test performed on **satellite** labels
- Model trained on **satellite** labels, test performed on **ortho** labels
- Model trained on **ortho** labels, test performed on **satellite** labels
- Model trained on **ortho** labels, test performed on **ortho** labels

The weights at each individual epoch require different thresholds to deliver the maximum possible *IoU* value. To determine the optimal threshold, different thresholds ranging from 0 to 1 with a step of 0.01 are used and the *IoU* scores stored in an array. The highest *IoU* value of this array is then used as the overall *IoU* score for that epoch. This is done for all four possible combinations of training labels and test labels. Figure 43 shows for each training-test combination and epoch the according overall *IoU* value.



Figure 43: *IoU* scores of different training and test data and different epochs. Each *IoU* value is the result of the highest *IoU* value possible for that particular epoch produced by the optimal threshold.

Between different epochs there are quite some jumps in the **IoU** values visible. This may be a result of the test data set being rather small with only 107 tiles for the satellite labels and 112 tiles for the ortho labels. Between the epochs 44 and 54 as well as 94 and 99 on the ortho labels, the model did not improve during training, and thus the weights were not saved. This results in the straight lines between these epochs in figure 43. The highest **IoU** scores along with the threshold and the epoch for each configuration are presented in table 6.

	Test on satellite labels	Test on ortho labels
Training on satellite labels	0.5114, Threshold 0.05, Epoch 9	0.4951, Threshold 0.80, Epoch 4
Training on ortho labels	0.4576, Threshold 0.05, Epoch 8	0.5526, Threshold 0.26, Epoch 20

Table 6: **IoU** scores, thresholds and epochs for comparing different satellite training and test labels

The two configurations in which training and test are performed on the same labels result in higher **IoU** scores than when calculating mixed predictions. The highest value can be found at the ortho labels, despite the fact that the satellite labels appear to fit the images better. Since the ortho labels are accepted as ground truth, the test should ideally be performed on those and not on satellite labels. With an **IoU** score of 0.5114, the prediction of the model trained on satellite labels is surprisingly good, given the fact that the training labels are that much smaller in general. Satellite test labels appear to require a low threshold to keep the prediction small, the ortho test labels on the other hand require a large thresholds to increase the prediction of the model trained on satellite labels. Figure 44 shows the **IoU** values for each threshold at the four best performing epochs.

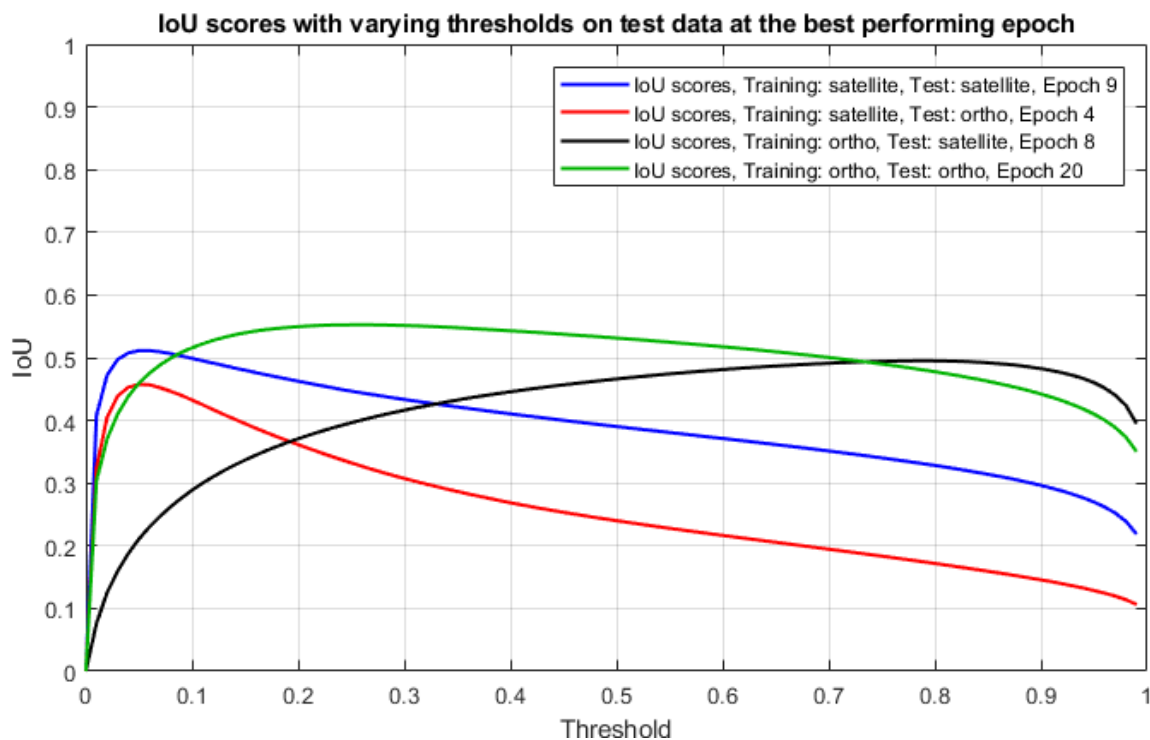


Figure 44: **IoU** scores of the best performing epoch of each of the four models with different thresholds

The satellite test labels seem to have a negative impact on the threshold, shifting the optimum to very low values. This means that models detect a lot of pixels as damaged, with a commonly used threshold of 0.5 the prediction would therefore contain a lot of false positive pixels. The threshold curves for the ortho test labels however resemble more a distribution as it is expected, with the maximum not being at 0.5, but at least closer to it with 0.26 and 0.80. Since the ortho labels are considered as ground truth, it is shown that the ortho labels are better suited for training on the satellite images than satellite labels.

Figure 45 shows various test metrics for the model trained on satellite labels and the satellite test labels as reference. The x axis shows the threshold between 0 and 1, the y axis shows the score of each individual metric between 0 and 1.

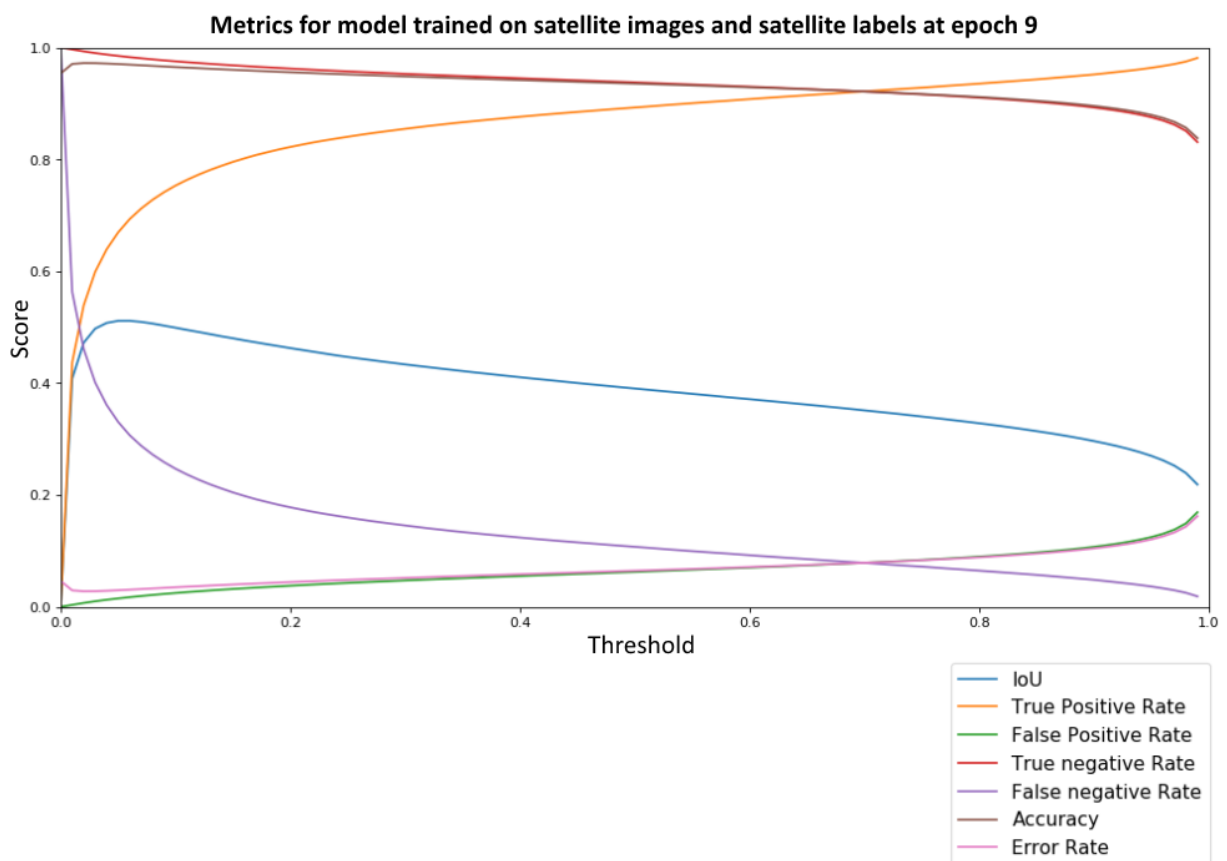


Figure 45: IoU, true positive rate, false positive rate, true negative rate, false negative rate, accuracy and error rate for epoch 3 on the satellite test data set with satellite labels

The blue IoU curve has its maximum at the threshold of 0.05 with a value of 0.511, which indicates that the prediction would normally contain a lot of false positive pixels at a threshold of 0.5, and only the very low threshold value reduces that error. This is also visible in the true positive rate, which inclines very fast at the start, showing that the first few threshold percentages have the highest impact on the total prediction size. The false positive and true negative rates are rather flat, the model is particularly trained to avoid false positive due to the large class unbalancing in favor of no damage pixels. The highest change of the inclination of the false negative rate, which is the opposite of the true positive rate, can be found

at around the maximum of the intersection over union curve. It is quite hard to get a lot of false negative pixels since the positive damage class is so sparse. Only by eliminating almost the entire prediction by lowering the threshold close to 0, the false negative rate rises significantly. The accuracy score has its maximum at 0.972 and is generally very high, but since it shows the total percentage of pixels that are correctly predicted, and an average of 95% of the pixels contain no damage, it is not particularly hard to reach high accuracy scores. The *IoU* coefficient is in this case the more meaningful metric. The error rate, being the opposite of the accuracy, shows similar behavior but may also be not a good metric in terms of its significance due to the class unbalancing.

Figure 46 shows the *ROC* curve of the prediction of the model trained on satellite labels and the test performed on satellite labels. It is a plot of the false positive rate against the true positive rate. A straight line would equal a random guess, the displayed curve is far from that straight line, which means the prediction in general is rather good. It does not require a high false positive rate in order to achieve a high true positive rate, which is always the desired result. According to figure 46, the optimal threshold might be around 0.1 at the turning point of the curve. This does however not take into account the *IoU* score, which is usually the more reliable metric. There are no values available from 0.25 onwards, since the false positive rate does not reach higher values in figure 45. The *ROC* curve does actually go beyond the maximum of the false positive rate in figure 45, this is a result of the *ROC* curve being created with a smaller threshold step.

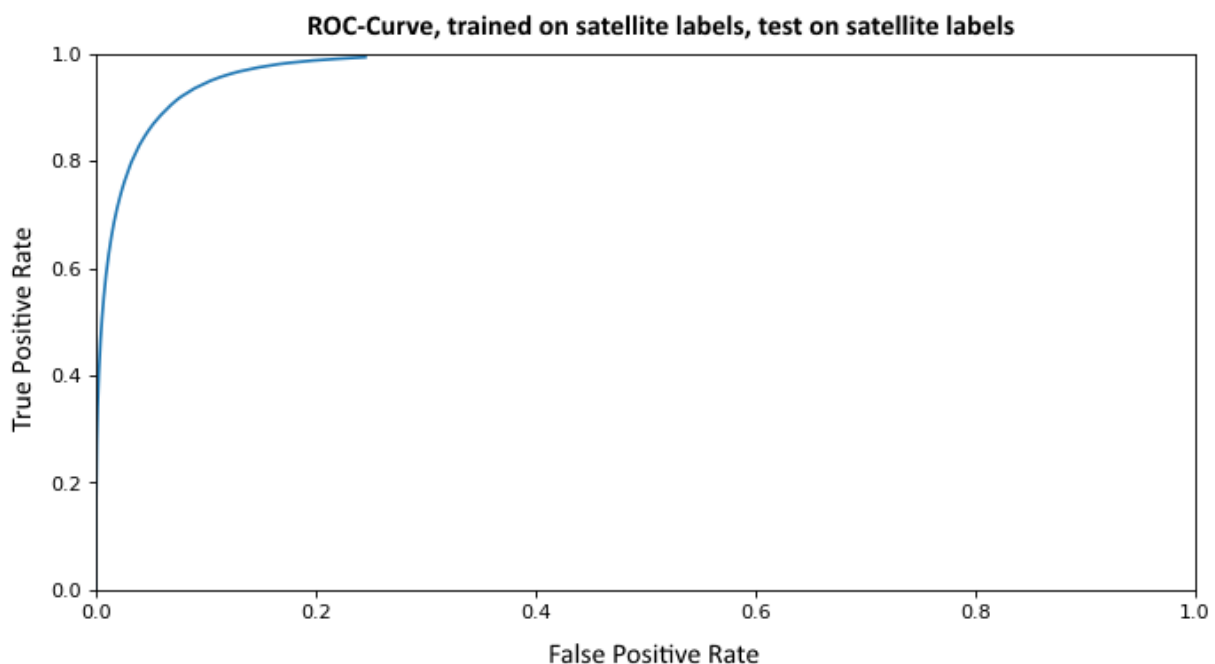


Figure 46: *ROC*-Curve of the prediction on satellite images. The model was trained on satellite labels, the test performed also on satellite labels

Figure 47 shows the same curves as figure 45, but this time for the model trained on ortho labels with ortho labels as reference for the metrics. The maximum of the *IoU* curve is at the threshold of 0.26 with a value of 0.553, which is a bit higher than the maximum of the satellite trained model, and also not as close to 0. The accuracy has its maximum at a threshold of 0.12 with a score of 0.928. The true positive and false negative rates in terms are not as steep in general, since the damage areas are larger

and there is therefore more room for error. The false positive rate and true negative rate are a bit lower and respectively higher, the accuracy and the error rate show the opposite behavior. With an average pixel ratio of damage to no damage of 0.89 the ortho damage areas are larger, but the unbalance is still too much to give the accuracy metric much significance. The difference in the class balancing can be seen in the different accuracy values. The second model should theoretically be the superior one due to the higher *IoU* score, but the accuracy is in fact lower.

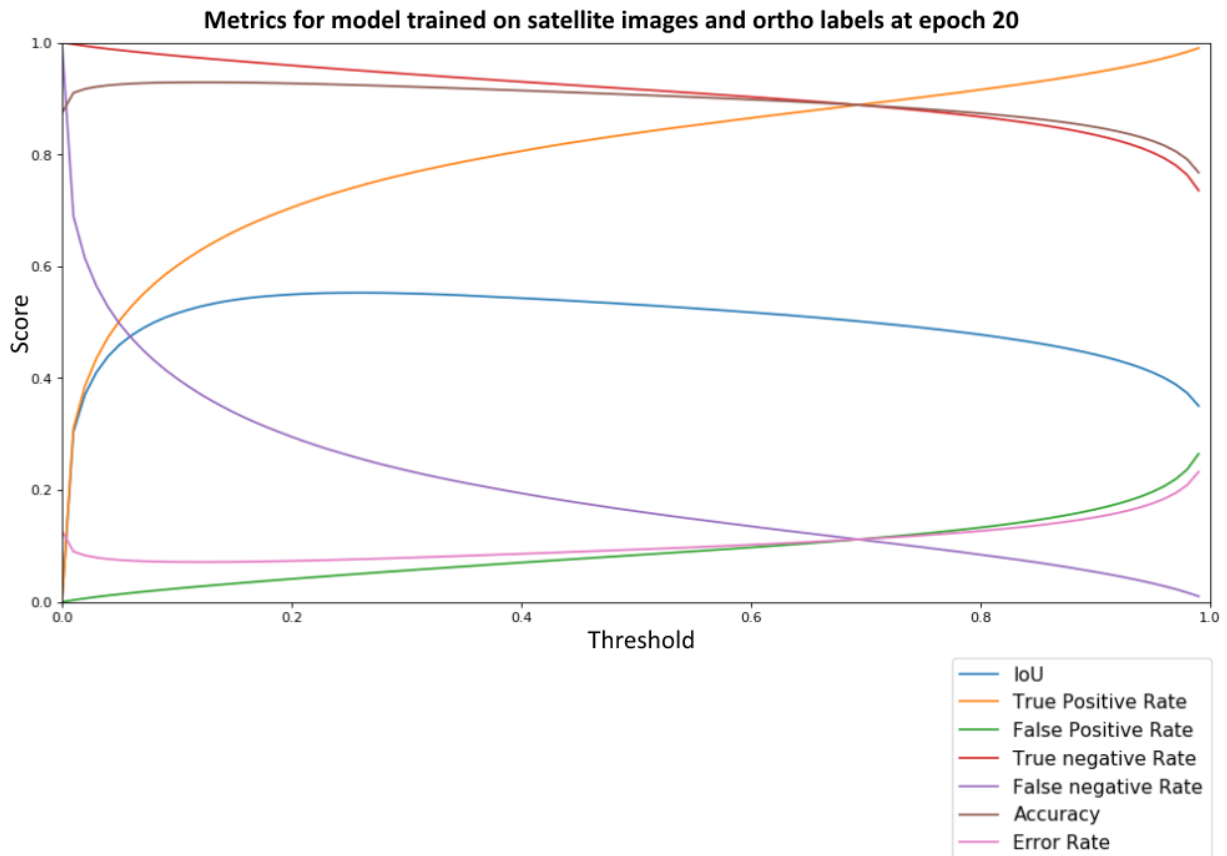


Figure 47: *IoU*, true positive rate, false positive rate, true negative rate, false negative rate, accuracy and error rate for epoch 3 on the ortho test data set

Figure 48 shows the *ROC* curve of the prediction from the model trained on ortho labels with the test performed on ortho labels. The curve is not as steep as the curve in figure 46, suggesting that the prediction is not as good as the first one. It requires a higher false positive rate to achieve the same true positive rate, this is a result of the larger and more detailed labeling which are more prone to errors. The class balancing is also a bit better, which leads to more total pixels that can be detected as true positive.

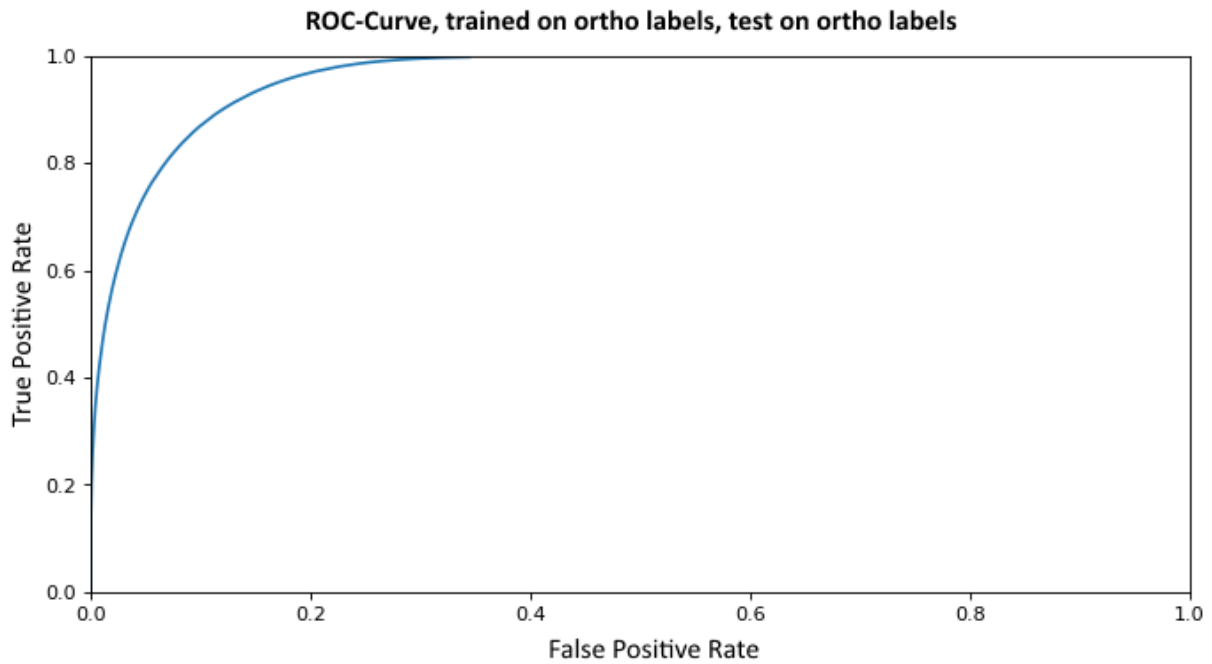


Figure 48: ROC-Curve of the prediction on satellite images. The model was trained on ortho labels, the test performed also on ortho labels

The figures 49 and 50 show a random satellite image tile from the test data set together with the respective satellite labels and the prediction without a threshold and with a threshold of 0.05 applied. It is clearly visible that the raw prediction without a threshold includes too many false positive pixels around the correct damage labels. In places where there is clearly no damage, the prediction does actually give these pixels a score close to 0, this shows that the model is well trained to detect true negative values, which represent everything but storm damage.

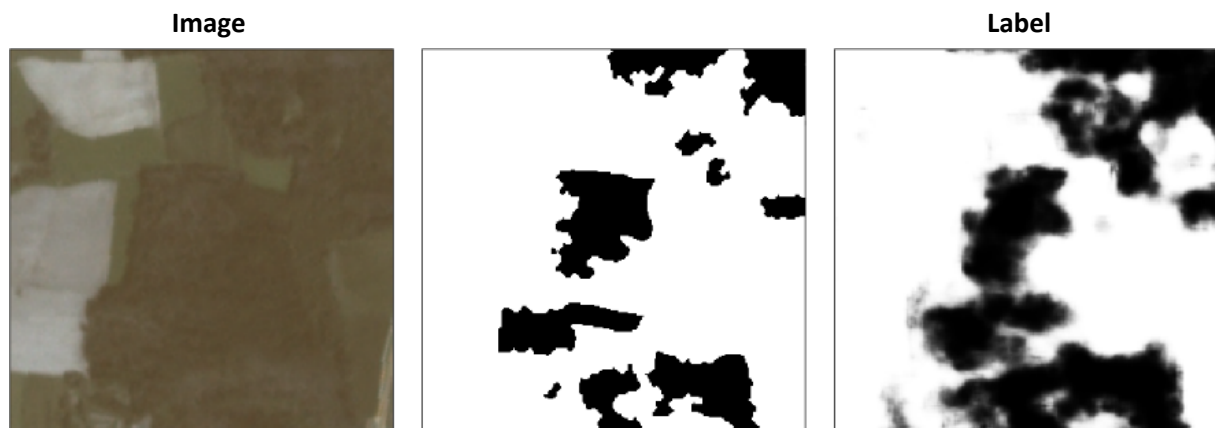


Figure 49: Satellite tile 1 with label and prediction, without threshold. White represents no damage, black represents damage. The prediction shows the probability of damage between 0 (white) and 1 (black)

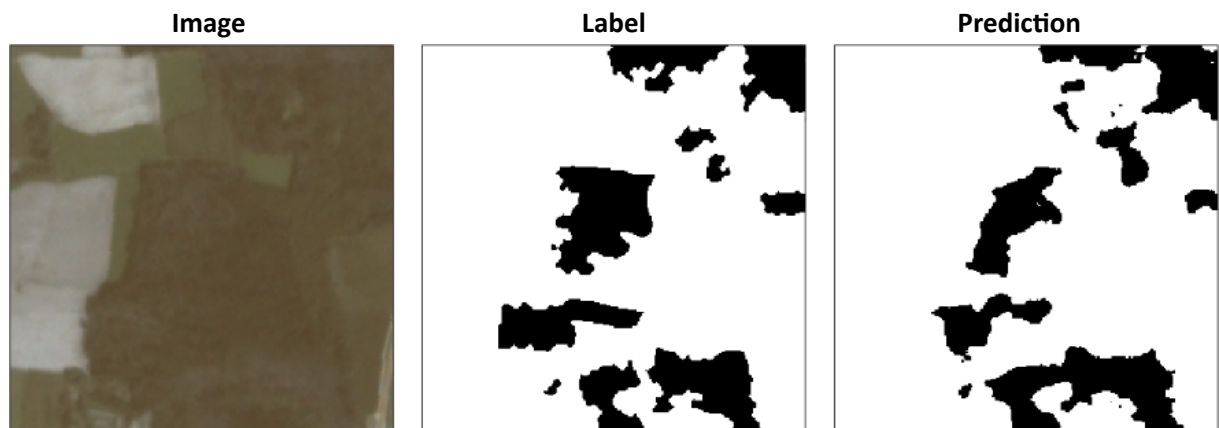


Figure 50: Satellite tile 1 with label and prediction, with threshold of 0.05 applied. White represents no damage, black represents damage

The figures 51 and 52 show another randomly selected tile from the satellite test data set together with the satellite labeling and the prediction, first without a threshold and then with a threshold of 0.05 applied.

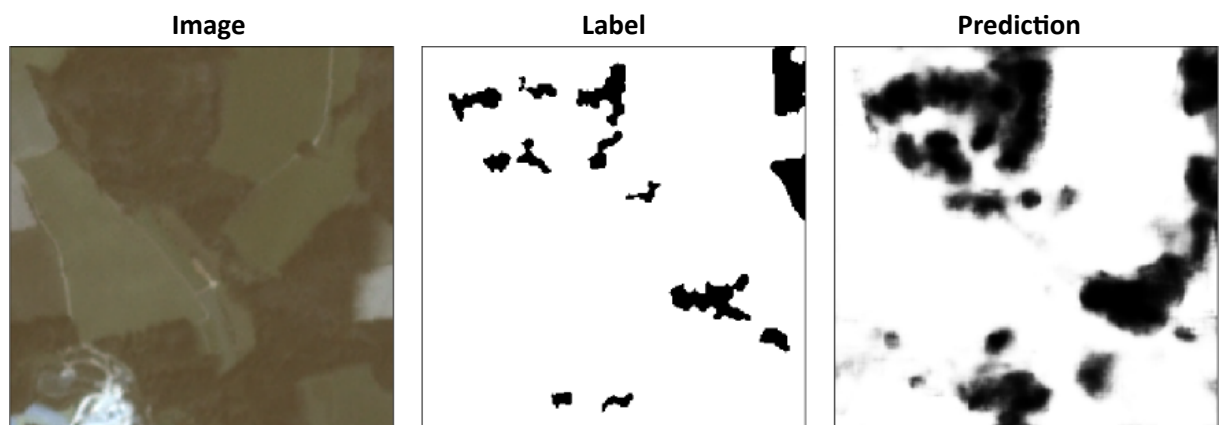


Figure 51: Satellite tile 1 with label and prediction, without threshold. White represents no damage, black represents damage. The prediction shows the probability of damage between 0 (white) and 1 (black)

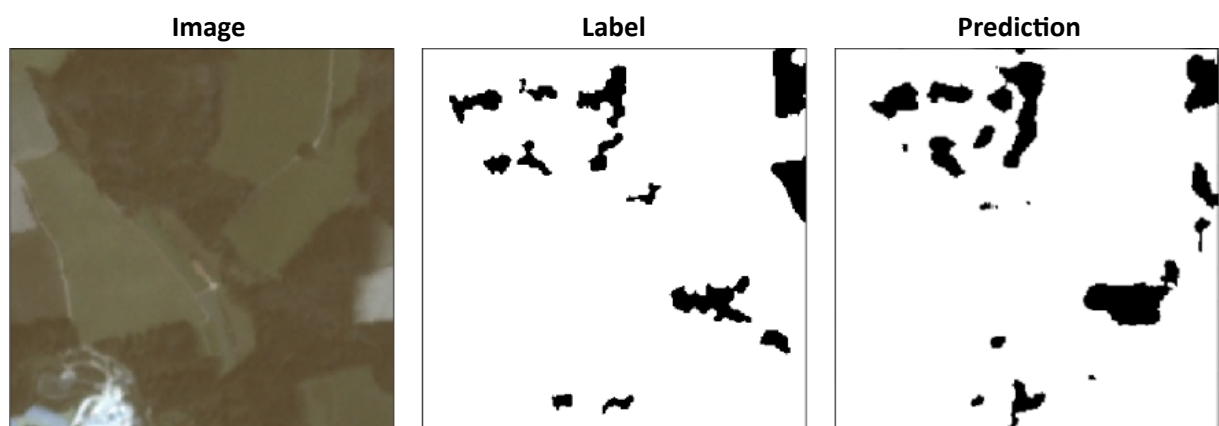


Figure 52: Satellite tile 2 with label and prediction, with threshold of 0.05 applied. White represents no damage, black represents damage.

5.1.2 Performance evaluation for ortho images

Usually an abort criterion during training would be the visible occurrence of overfitting within the training and validation loss curves. During Hamdi's experiments overfitting was clearly visible around the 33rd epoch, so it is surprising that no overfitting is visible during training in this thesis. Figure 53 shows the training and validation loss as well as the training and validation *IoU* curves plotted over all epochs. An explanation can be found in the additional batch normalization layers as well as the extensive data augmentation, which was not performed by Hamdi.

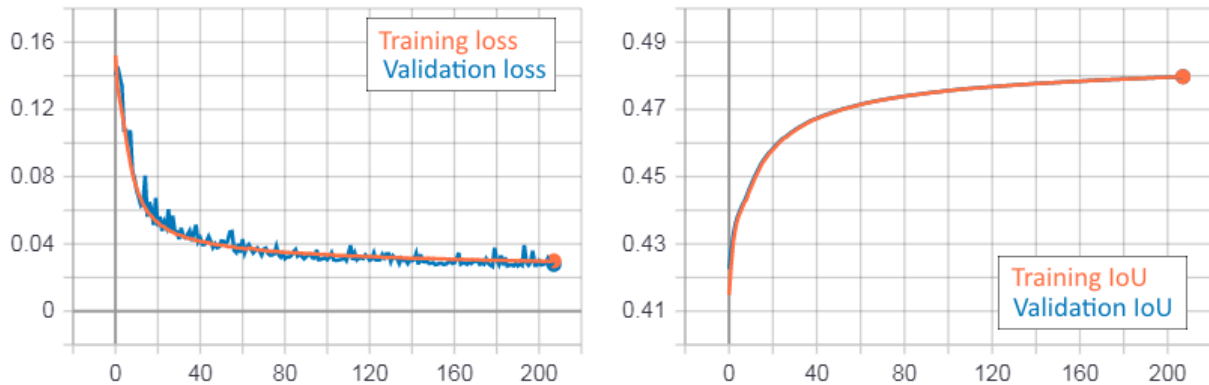


Figure 53: Performance at each epoch during training on ortho data. The left graph shows the values of the loss function for both the training and the validation data. The x axis represents the epochs, the y axis the loss value. The right graph shows the calculated *IoU* score for each epoch, the x axis represents the epochs, the y axis the *IoU* score.

Since the test data set of the ortho data is a lot larger than the test data set of the satellite data, calculating the optimal threshold for each epoch and comparing the results to find the best model iteration is not possible in a reasonable amount of time. Some random samples combined with an iterative search however show that the optimal epoch is actually the third epoch, with all following epochs delivering a lower test *IoU* score. This might be due to the extensive data augmentation and the already quite large training data set, so that three epochs are enough and after that overfitting does actually occur.

Figure 54 shows various test metrics for the model trained on ortho images. The *IoU* score has its maximum at a threshold of 0.67 with a value of 0.728, which is significantly higher than the *IoU* scores of the models trained on satellite images. This is no surprise given the much higher resolution of the ortho images. With the rather low class unbalance of 65% no damage pixels, the accuracy metric can be taken into account, and with a maximum of 0.858 at the threshold of 0.58 a rather high value is achieved. It is however inferior to the model of [Hamdi et al., 2019], which reached an accuracy 0.92, but was also trained on other labels that might have a higher unbalance. The *IoU* score of 0.728 however is higher than the *IoU* score reached by Hamdi, which is 0.635. The true positive rate has a steep increase at the first few threshold values up to a score of about 0.5, followed by a rather linear inclination. This means that it is easy for the model to detect half of the damaged pixels, with the second half being a lot harder to detect. At the maximum of the *IoU* curve, 88% of the pixels labeled as damaged are correctly predicted and therefore marked as true positive. The false positive rate

increases much faster, with already 18% false positive pixels at the **IoU** maximum. The true negative rate crosses the true positive rate at 0.68 which is about the maximum of the **IoU**, which is a coincidence and not a general rule. For balanced data sets and well trained models it might be expected that this crossing as well as the **IoU** maximum is at a threshold of 0.5. The false negative rate is the opposite of the true positive rate and crosses the false positive rate at the same threshold of 0.68. The error rate has its peaks at the edges of the graph, meaning the highest error can be found with very low and very high thresholds. The lowest error rate of 17% is also at the same threshold of 0.58 as the maximum of the accuracy.

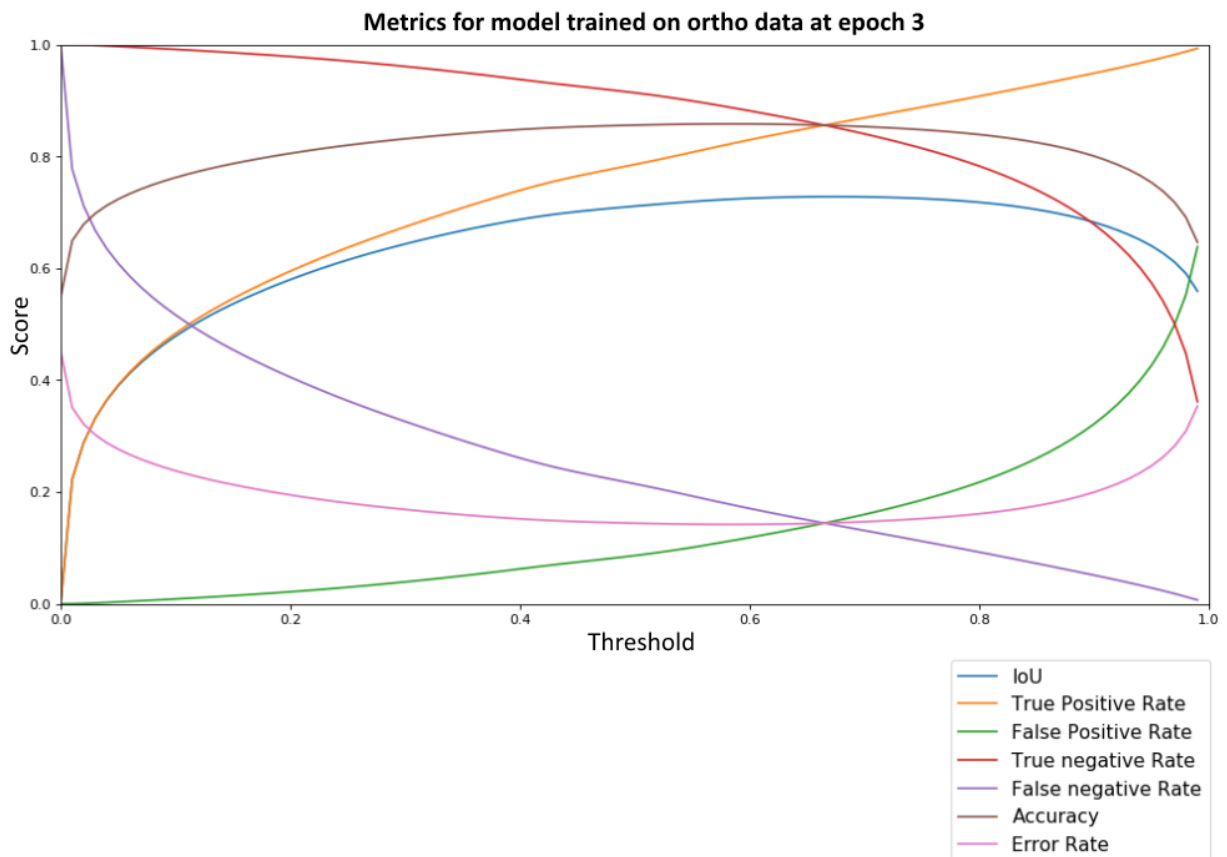


Figure 54: **IoU**, true positive rate, false positive rate, true negative rate, false negative rate, accuracy and error rate for epoch 3 on the ortho test data set

Figure 55 shows the **ROC** curve of the prediction from the model trained on the ortho data set. Although the curve is far from being a straight line, it is still flatter than the curves of the models trained on satellite images, which are shown in the figures 46 and 48. For instance at a true positive rate of 90%, the false positive rate is already at about 50%. It is therefore harder to maximize the true positive rate without getting too many errors. The turning point of the **ROC** curve is at a false positive rate of about 0.15 with a true positive rate of about 0.75. Overall the curve still suggests that the model does perform quite well.

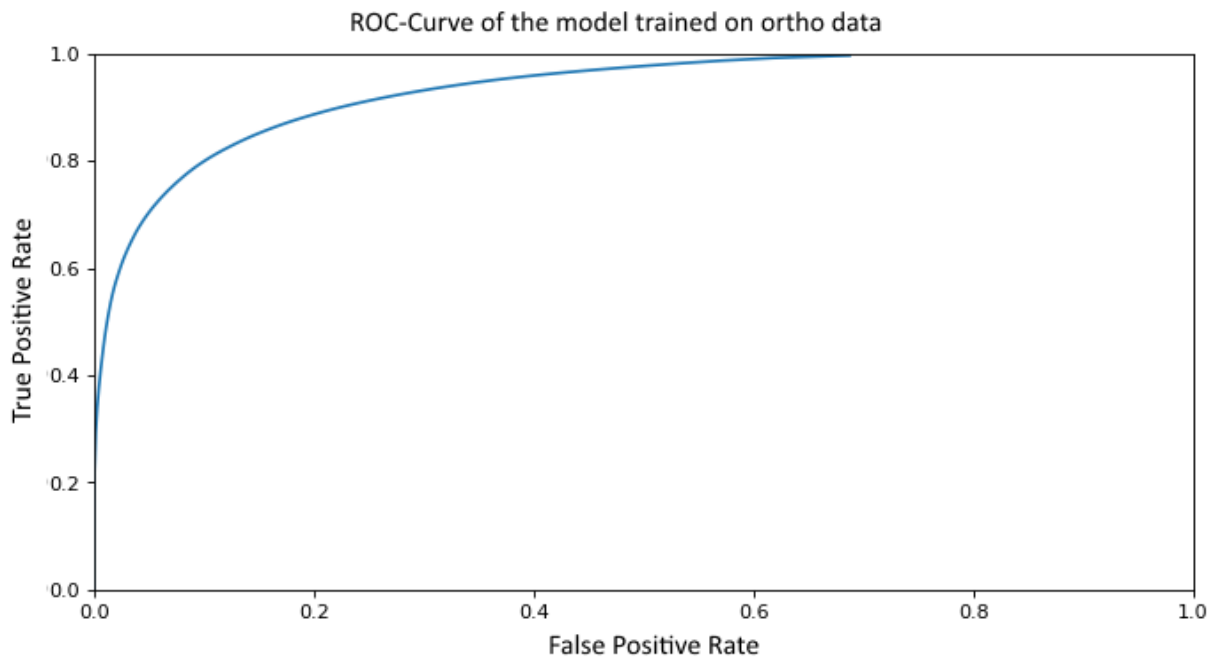


Figure 55: ROC-Curve of the prediction on ortho images. The model was trained on the ortho image and label data set

The figures 56 and 57 show the prediction of a randomly selected ortho tile from the test data set along with the according label and the prediction. Figure 56 shows the prediction without a threshold, figure 57 shows the effect of a threshold of 0.67. It is clearly visible that the model is for instance not trained to solely detect the shape of fallen trees or the brown color of damage, but to take a multitude of factors into account. The dark spot at the bottom for example is clearly distinguished in the prediction from the dark spots at the top, although on their own they look very similar.

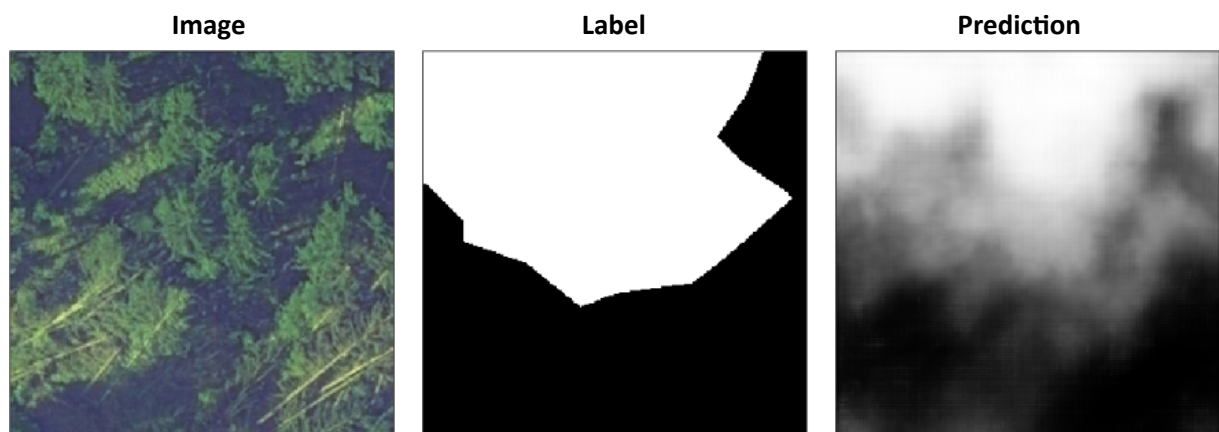


Figure 56: Ortho tile 1 with label and prediction, without threshold. White represents no damage, black represents damage. The prediction shows the probability of damage between 0 (white) and 1 (black)

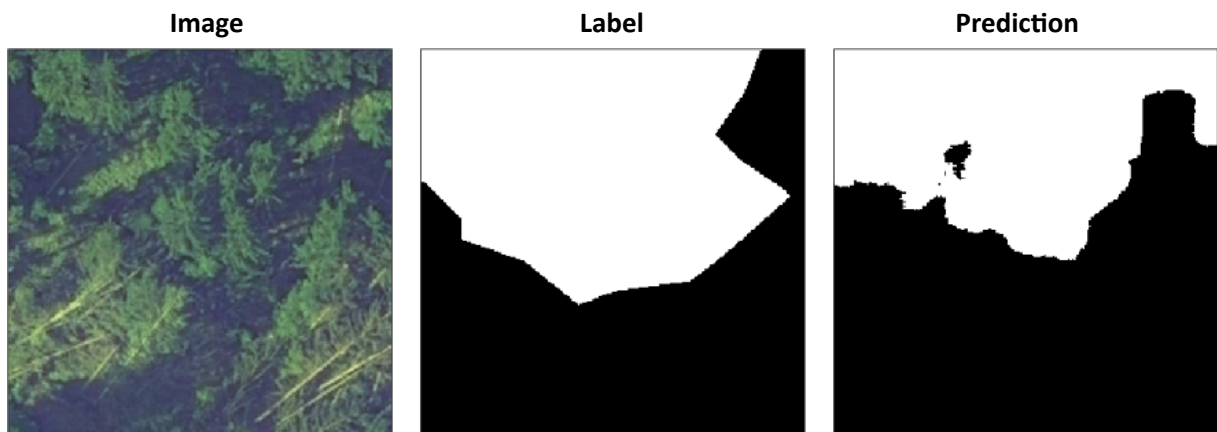


Figure 57: Ortho tile 1 with label and prediction, with threshold of 0.67 applied. White represents no damage, black represents damage.

The figures 58 and 59 show another randomly selected ortho tile from the test data set together with the according label and the prediction, once without a threshold and once with a threshold of 0.67 applied.

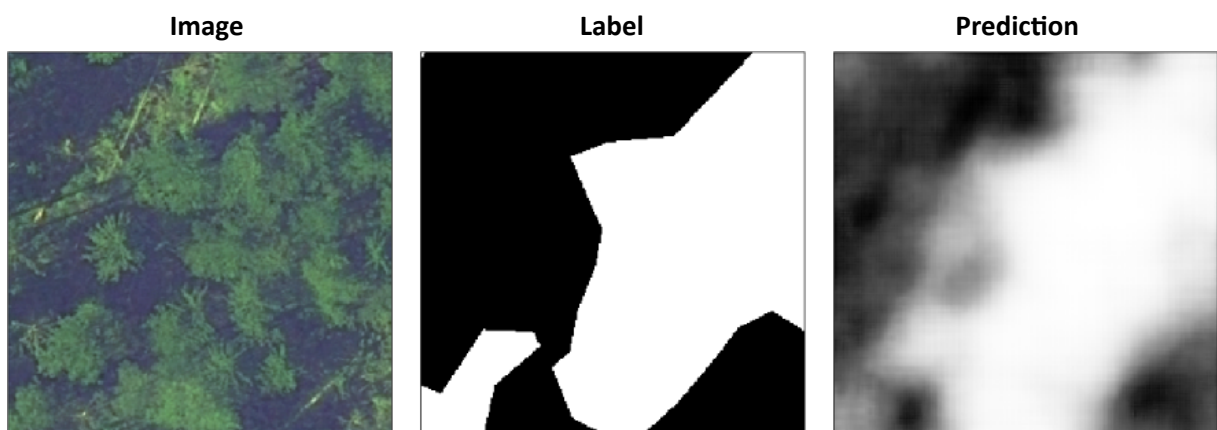


Figure 58: Ortho tile 1 with label and prediction, without threshold. White represents no damage, black represents damage. The prediction shows the probability of damage between 0 (white) and 1 (black)

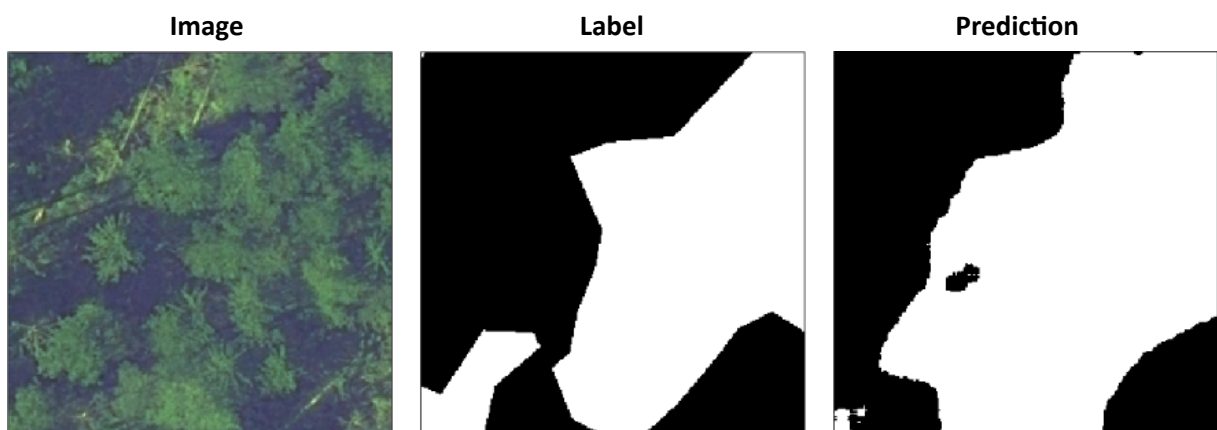


Figure 59: Ortho tile 2 with label and prediction, with threshold of 0.67 applied. White represents no damage, black represents damage.

5.1.3 Performance evaluation for transfer learning

The prediction of the model that utilizes the VGG19 encoder and the U-Net decoder is performing about equally as the U-Net architecture trained on ortho data. Figure 60 shows the evaluation metrics of the model at epoch 15. The **IoU** curve has its maximum at a threshold of 0.51 with a value of 0.755, which is slightly higher than the 0.728 achieved by the U-Net model. The maximum accuracy of 0.841 at the threshold of 0.39 however is slightly lower than the 0.858 of U-Net. The other true positive rate and the false negative rate show again a rather linear inclination compared to the satellite predictions, which means that it is harder for the model to detect every damaged pixel on this data set. It is possible, and also likely, that the epoch 15 is not the ideal epoch at which the model performs best, but a search for the best epoch is very time consuming due to the high inference time. The results at epoch 15 however show that transfer learning in general can be used on the ortho data set.

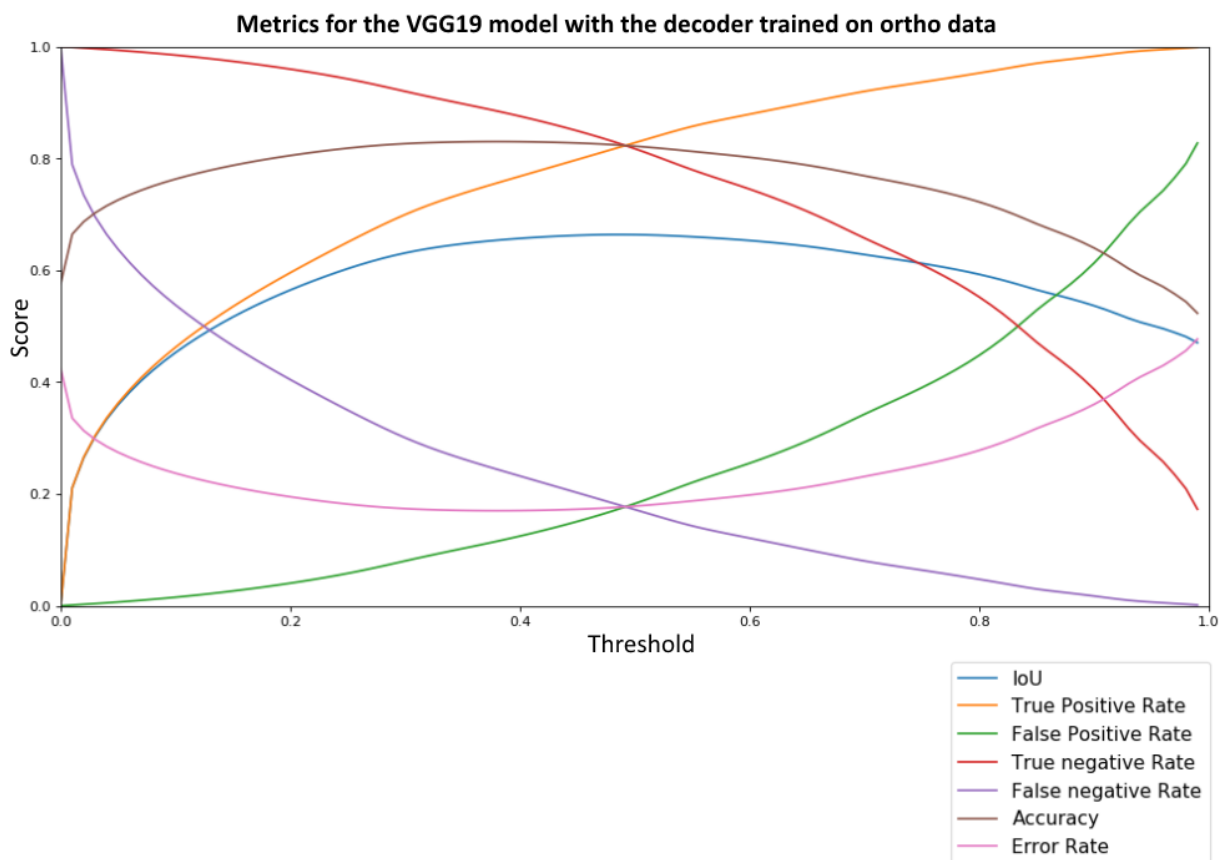


Figure 60: IoU, true positive rate, false positive rate, true negative rate, false negative rate, accuracy and error rate for epoch 15 on the VGG19 model

Figure 61 shows the **ROC** curve of the prediction from the VGG19 model that was trained a few epochs on the ortho data set. The curve is fairly similar to the **ROC** curve of the U-Net model trained on ortho data. The turning point is located at a false positive rate of about 5% and a true positive rate of about 70%. Increasing the true positive rate above 80% comes with a drastic increase of the false positive rate.

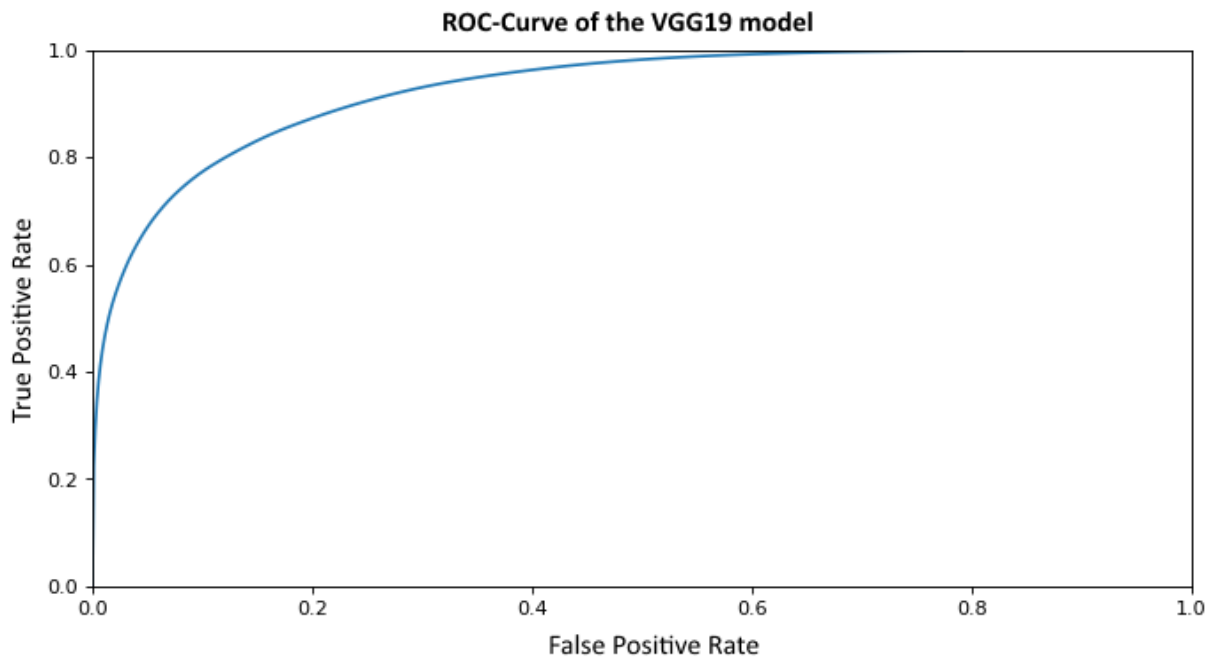


Figure 61: ROC-Curve of the VGG19 model prediction on ortho images. The model decoder was trained on the ortho image and label data set, the encoder was trained using the ImageNet database [ImageNet, 2016]

5.1.4 Test of the impact of partial reduction of the training data set

For most deep learning applications, a very large set of data is required with numbers of individual tiles going up to 100000 or even a million or more. The satellite data set is in this regards very limited, since it offers with a horizontal and vertical overlap of 50% during tiling a maximum of just over 2000 individual tiles that can be used for training, not including data augmentation. To counter this flaw, the U-Net architecture is used which is suitable for training on small data sets. A relevant question however is, if 2000 tiles are enough, and how much the result could be improved by adding more data. A possible answer can be found by artificially decreasing the available training data in several steps and training the same model architecture on the gradually decreasing amount of data. The prediction are performed on an independent test data set, which is the exact same for every test iteration. Starting with 10% and going up to 100% in 10% steps, the *IoU* scores at the optimal threshold for each respective data set are collected. The threshold is chosen in each step by sliding the threshold value from 0 to 1 in 0.01 steps and taking the maximum of all values. The tiles that are used for each step during training are selected randomly from the pool of available tiles. Figure 62 shows the *IoU* scores at each increasing data step along with an approximation using the least squares method with a maximum power of 2. Table 7 lists the prediction values for every test iteration.

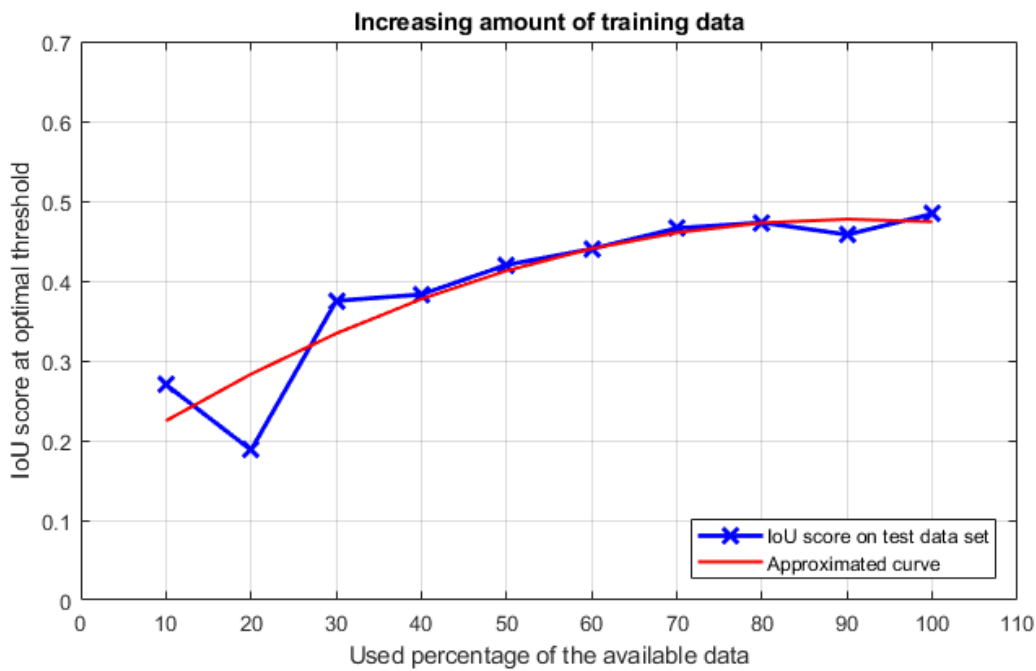


Figure 62: Different percentages of the data used for training, with an approximated curve using the least squares method in red.

Percentage of used data	IoU
10%	0.270
20%	0.188
30%	0.375
40%	0.383
50%	0.420
60%	0.440
70%	0.466
80%	0.473
90%	0.458
100%	0.484

Table 7: IoU values with different amounts of training data

As expected, the **IoU** score is decreasing the less training data is used, with some outliers at 20% and 90% used data. Since the tiles that are used are selected randomly, some test steps might contain more suitable tiles than other steps, which can be an explanation for these outliers. The decrease in the **IoU** score however is not linear, but more in the shape of a logarithm function. At 70% percent, the incline is already very slow, which indicates that at this point the complexity of the given data set is almost fully reached. Adding more tiles of the same data set does likely not benefit the model very much, however adding data from a different data set would improve the robustness of the model still greatly.

5.2 Evaluation of the prediction in ArcGIS Pro

A custom toolbox for ArcGIS Pro enables a prediction on large scale data without any further programming. Splitting the data into tiles and combining the prediction into a single image is done automatically, as well as the conversion to polygons and the elimination of small artifacts. The following test cases, if not further specified, are showing a small excerpt of the complete data set, which is within the dedicated test area. This means, the data in this area was never used during the training process and the various models are seeing it the first time when creating the prediction. A sample image of the scene is presented in figure 63 to show the test area as it is without any overlaying polygons. It contains the satellite image taken on the 30th of October 2017.



Figure 63: Test area without damage masks for comparing predictions on satellite data. The area is fully included in the test data set, which was not used during training.

5.2.1 Prediction of models trained on satellite images

The original satellite labels are less detailed and generally smaller than their ortho label counterparts, due to the fact that it is difficult for human supervisors to reliably detect damage based on the 3 m resolution of the satellite images. Figure 64 shows the satellite labels in green as well as the prediction that was created with a model trained on satellite labels in red. It is clearly visible that the model is able to distinguish between damaged grey forest and grey crop fields, as well as damaged grey forest and healthy green forest. The model seems to have some trouble with small patches on the right side, they could be ignored due to their somewhat rectangular form and similarity to crop fields. Some small false positive patches can be seen on the left and on the right side, those are larger than the given threshold of 10 m² below which all features are automatically removed. The inference for the complete data set for one of the satellite images takes about 40 seconds on a moderately fast computer with a Nvidia Quadro K2100M GPU and an Intel i7-4810MQ processing unit. For the given excerpt in figure 64, the inference would theoretically be about one second, but necessary initializations raise that number a bit. The short inference time of roughly 40 seconds enables a quick damage assessment and allows for multiple predictions with various thresholds in short succession to find the optimal threshold value.

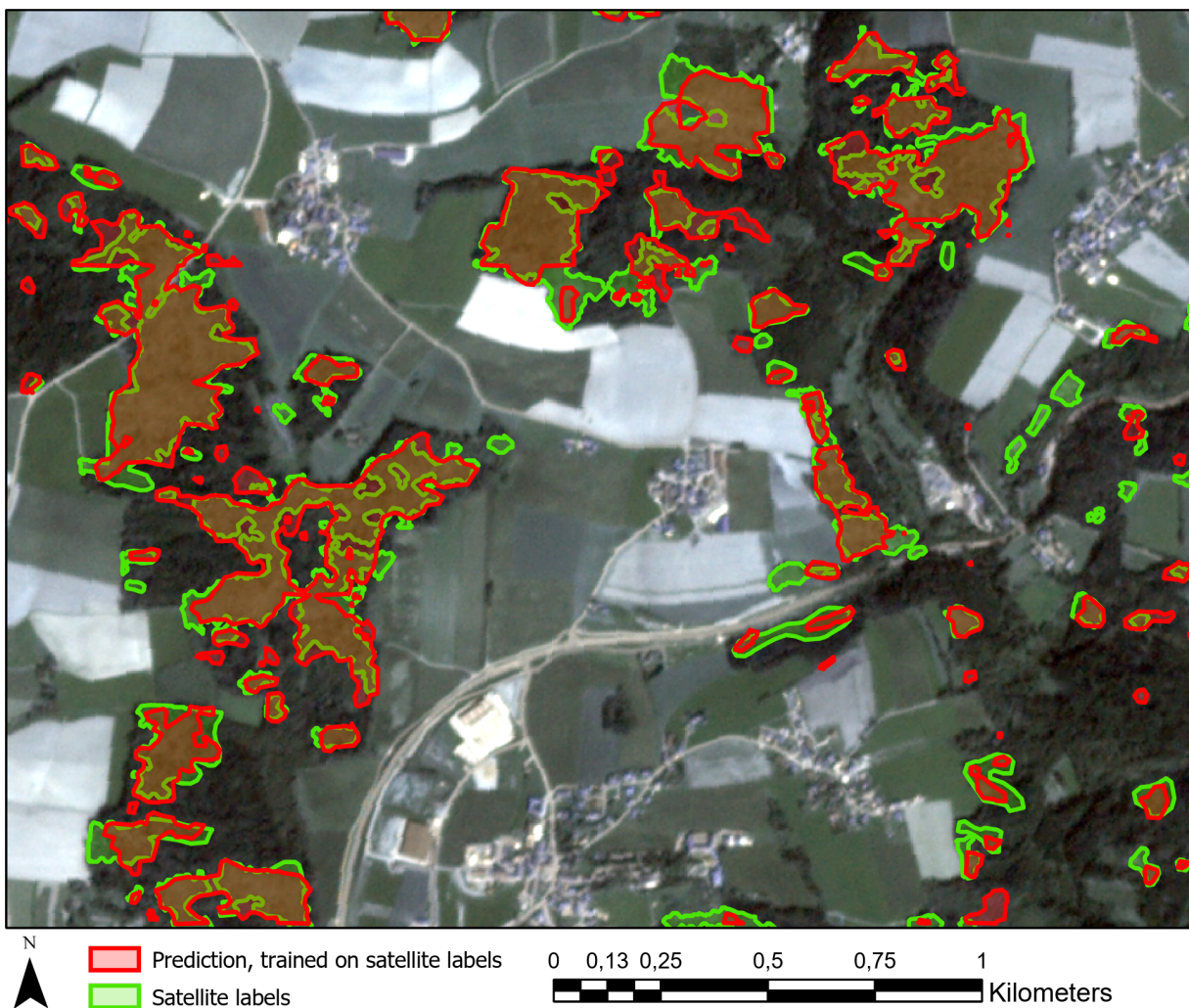


Figure 64: Comparison of the original satellite labels in green against the prediction of a model which was trained on the same satellite labels in red

The ortho labels are regarded as the ground truth and should ideally be used to assess the performance of predictions. The question remains, how well the ground truth can be predicted with a model that is trained on satellite labels only. The achieved **IoU** score of 0.4951 shows that in theory the prediction is only slightly worse than the prediction of the model trained on ortho labels with an **IoU** score of 0.5114. Figure 65 shows the ground truth in green together with the prediction of the model trained on satellite labels in red. When looking at the results in ArcGIS Pro, it appears that the model has a harder time to follow the pattern of the ground truth compared to the pattern of the satellite labels. Especially at small and filigree structures on the right side the prediction does have a lot of false negative area. On the other hand at larger structures for instance on the left side the prediction seems to fit quite well. On the lower right side there are some straight line artifacts visible in the prediction, these occur during the inference when some area in a tile is marked as damage, but in the neighboring tile it is marked as no damage. This can happen if the decision between damage and no damage is quite difficult and the according probability is close to the given threshold. It is rather surprising that the model does perform so well, for it was trained on satellite labels only and requires a high threshold of 0.8 to extend its prediction area enough to include so much damaged area, and still does not include any crop fields.

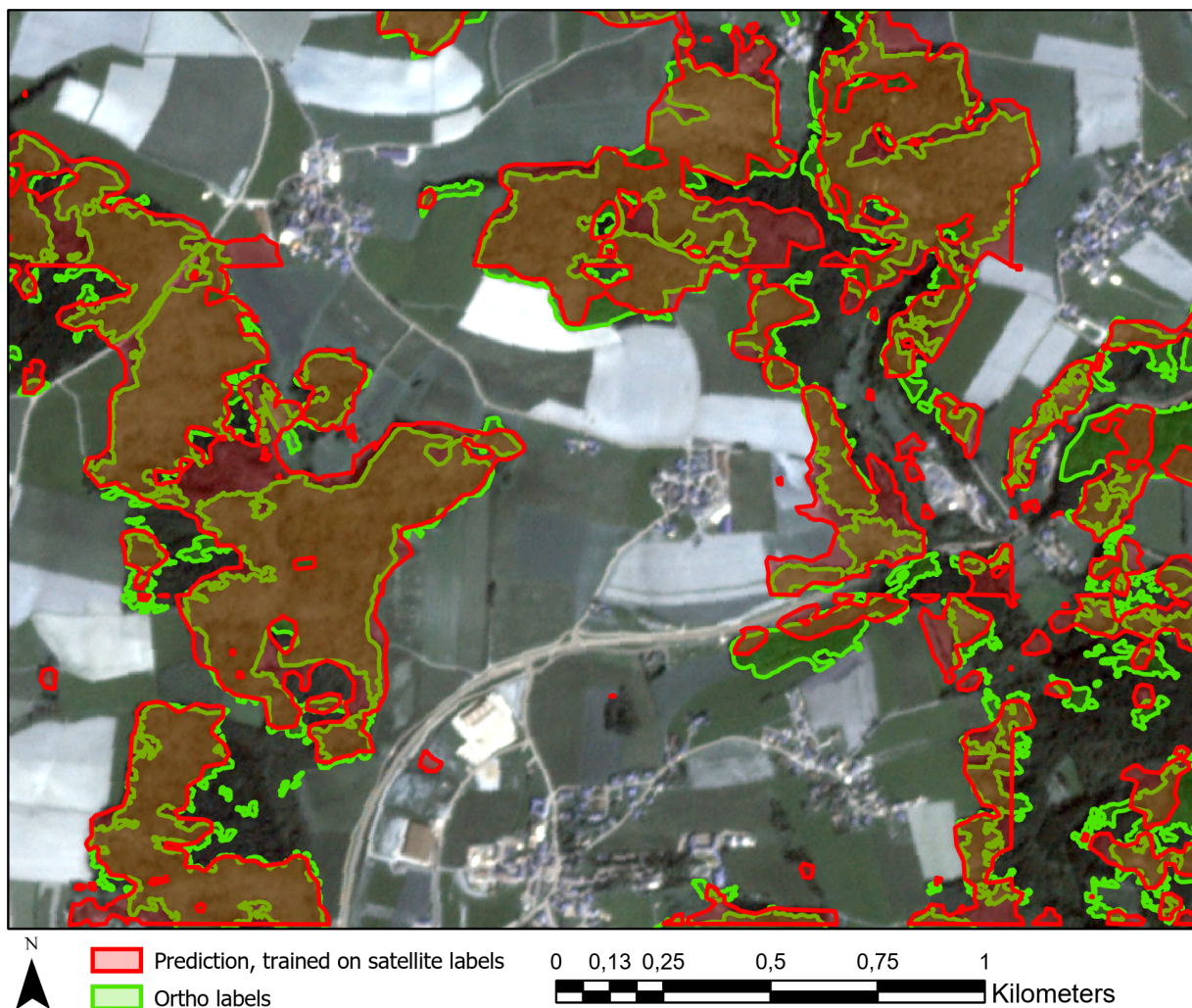


Figure 65: Comparison of ortho labels in green against the prediction of a model which was trained on satellite labels in red. The prediction is still performed on satellite image data

By training the model on satellite images and ortho labels, the model learns to recognize features which the human supervisors were not able to detect. This gives this particular model instance an advantage to the previous one which was only trained on satellite labels. This advantage is clearly visible in the more filigree surface of the prediction, as it can be seen in figure 66 as red polygons, with the ground truth marked as green polygons. On the right side there are still a lot of small features missing and labeled as false negative, but especially at the forest edges the prediction is a lot more accurate. However in some areas, especially with small ground truth features, the prediction of the model trained on satellite labels seems to be actually more accurate.

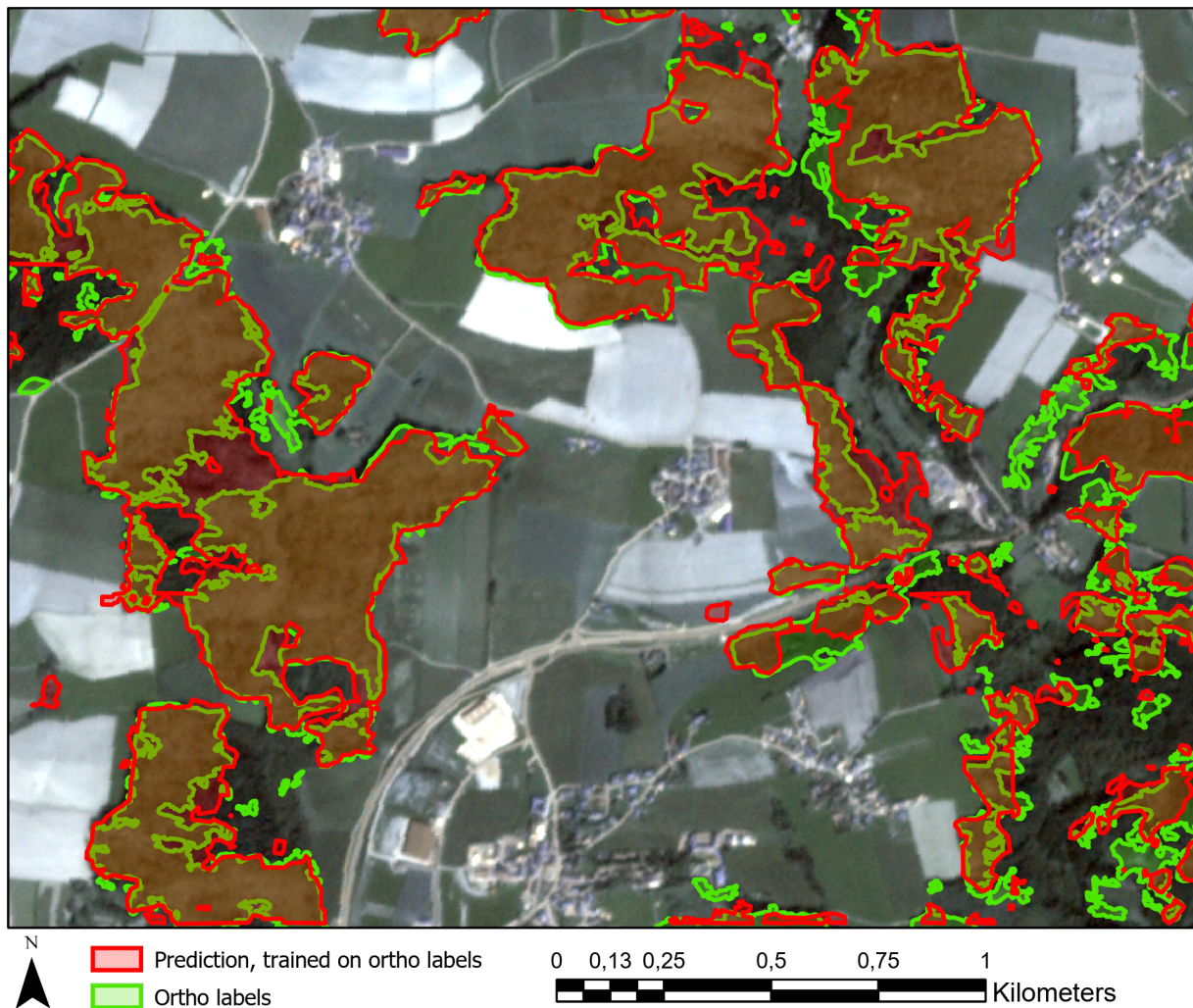


Figure 66: Comparison of ortho labels in green against the prediction of a model which was trained on ortho labels in red. The prediction is still performed on satellite image data

The previous scenes contain a lot of damaged forest but not much healthy forest. To verify if the model is not only able to differ between forest and crop fields, but also between damaged and undamaged forest, another image scene is presented in figure 67. It is also included in the test data set, some kilometers to the east of the previous scenes. The model is trained on satellite images and ortho labels, the prediction is performed so that it tries to match the ortho labels. For comparison the original satellite labels are also included in blue, they cover considerably less area than the ortho labels. The damaged forest areas in the left part of the image are detected well by the model, whereas the small areas in the right part

are almost entirely missed. The undamaged forest however is also almost entirely correctly predicted as true negative, which is also a crucial requirement in storm damage assessment.

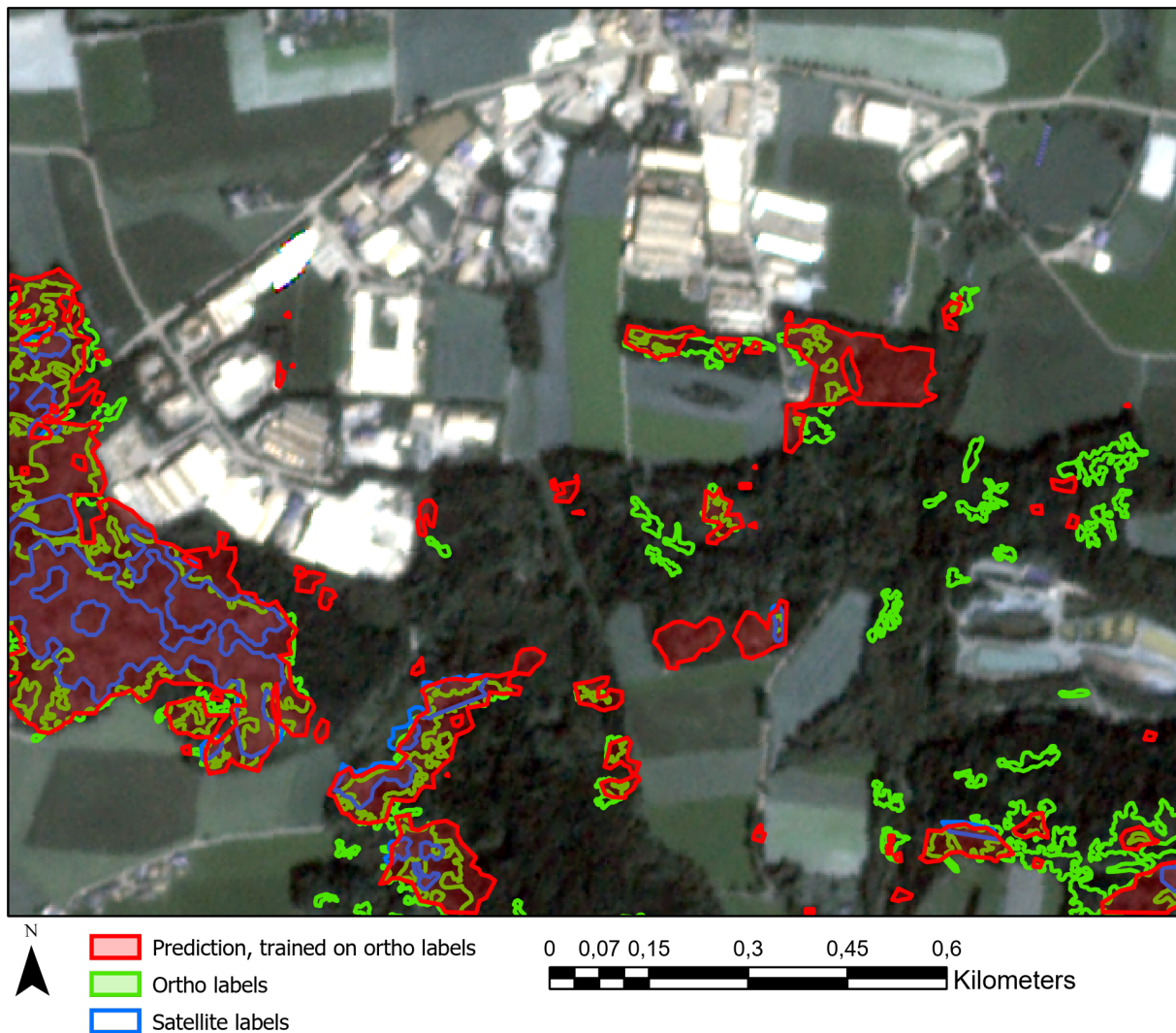


Figure 67: Comparison of ortho labels in green and satellite labels in blue against the prediction of a model which was trained on the same ortho labels in red. The scene shows a lot of undamaged forest along with a village. The prediction is performed on satellite image data

The total area of all damage included in the original satellite labeling is 7.85 km^2 , the area predicted by the model trained on satellite images and labels is 7.76 km^2 and thus slightly smaller. The predicted area depends however heavily on the chosen threshold value, which was in this case picked to maximize the *IoU* score, but could also be chosen in a way to minimize the discrepancy between the areas. The total area of all damage in the original ortho labels, which are considered as ground truth, is 17.35 km^2 , the area predicted from the model trained on satellite images and labels is 24.11 km^2 , the area predicted from the model trained on satellite images and ortho labels is 23.41 km^2 . Both predictions are considerably larger than the ground truth, but again the chosen threshold is selected for an optimal *IoU* score and not for an optimal match in the areas.

5.2.2 Prediction of models trained on ortho images

Inference for the same area with ortho data takes considerably more time, due to the higher image resolution, and the more tiles that are being created. For each of the 45 ortho images inference is about 5 minutes, which would sum up in total to about 4 hours for the complete data set. Figure 68 shows in an almost complete extent one of the eight test images together with the ground truth as green and the prediction as red polygons. In comparison to the satellite predictions, the ortho prediction manages to align the damage borders and the borders between forest and fields in most parts quite well. Noticeable are the patches of false positive damage in the middle of the scene. Since the tiles are so small with 50×50 meters each, it can be challenging for the model to determine whether there is damage or not by observing only one tile at a time. Especially the large false positive patch that is completely included in a crop field is interesting. The field looks about the same in its full extent, but only a part of it is incorrectly labeled. In the left side some areas of false negative inside the large damage area are present, the trees in this region might have not been as effected by the storm as the surrounding area.

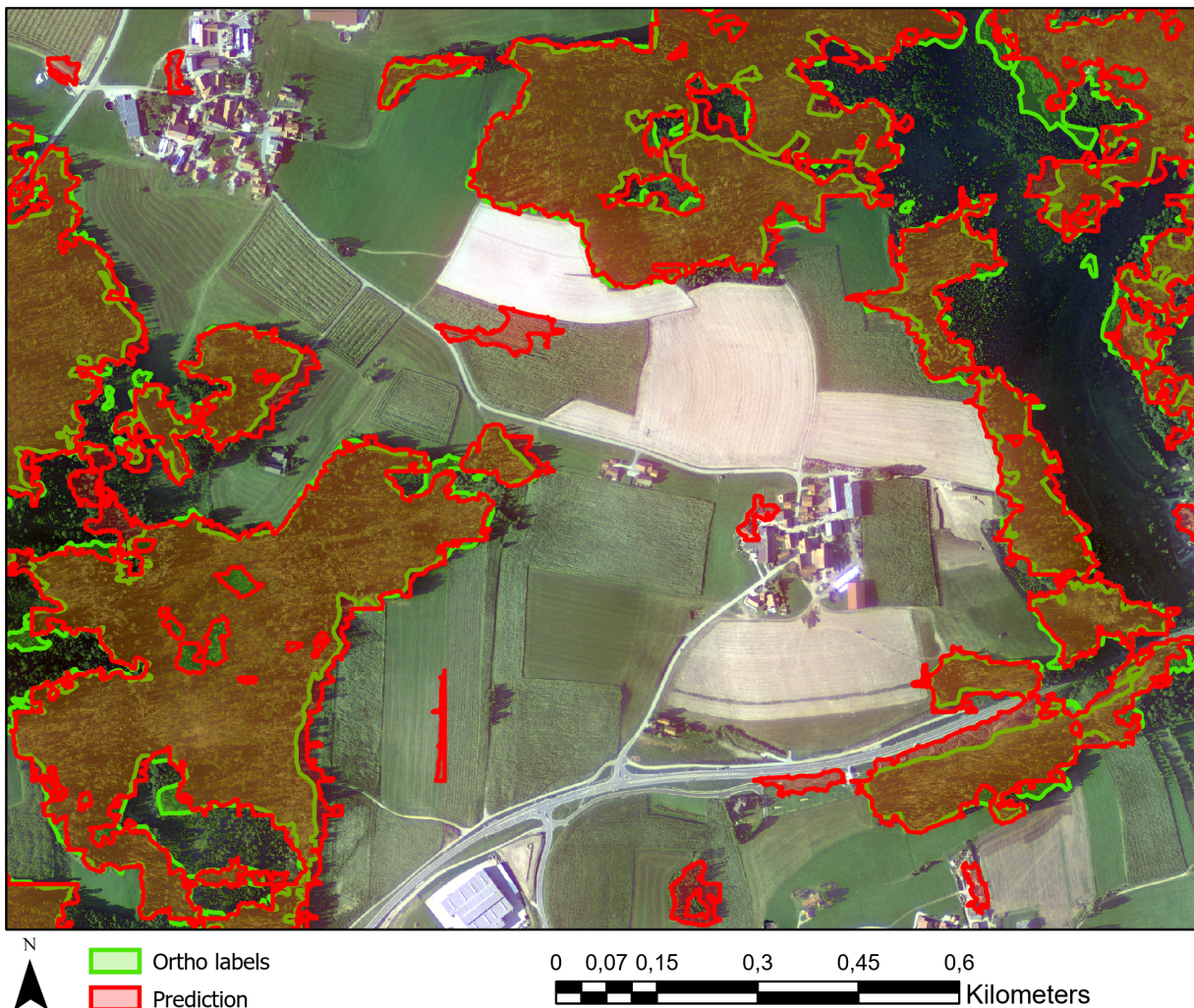


Figure 68: Comparison of the ground truth in green against the prediction of a model which was trained on the ortho images and labels in red

When taking a closer look at the prediction in the right part of figure 68, it appears that the model is performing with different accuracy at differently orientated edges of the forest. A small excerpt of the scene is shown in figure 69 as a zoomed in version. In the scene the sun is standing in the south west, which causes the shadows to cast in the direction of north east. At the western, northern and southern edge of the forest the prediction is very well aligned with the ground truth, whereas on the eastern edge an offset of up to 30 *m* is visible. The shadows in this part create distortions which cause a false positive prediction. A simple way to compensate this flaw is to take the images when the sun is at its highest point, minimizing the total shadows it casts. Training the model to specifically ignore shadows is a difficult task, the data could be prepared in a way so that shadowy regions have more weight in the loss function, but the U-Net architecture is likely not suitable for this kind of implementation.

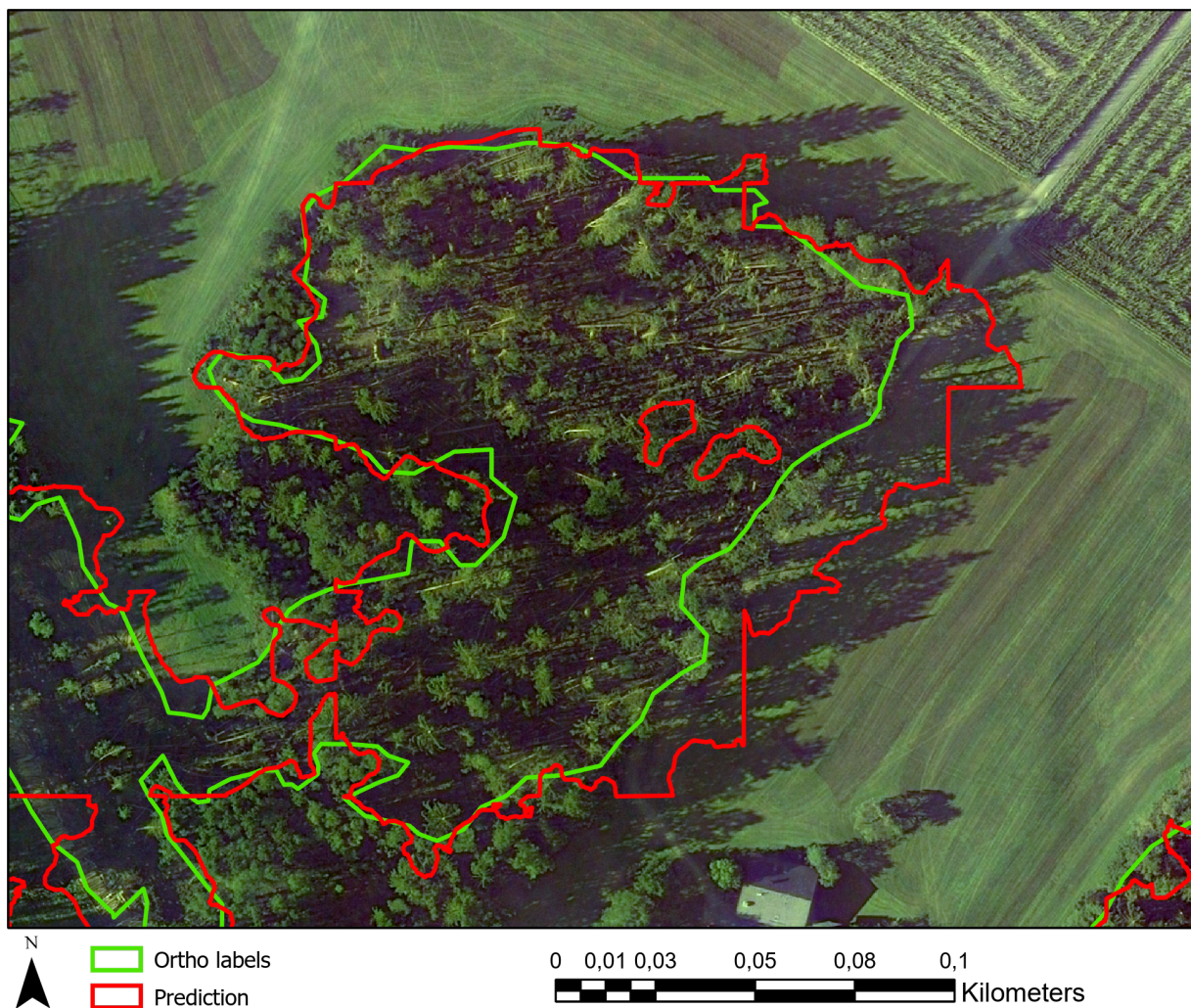


Figure 69: Zoomed in comparison of the ground truth in green against the prediction of a model which was trained on the ortho labels and images in red. The sun is shining in the scene from the south west, casting shadows in the north east

Figure 70 shows another excerpt of figure 68, this time at the bottom left with a road dividing two forest damage areas. Particularly interesting is the false positive stripe right below the street, where there appears to be no storm damage but some sort of smaller vegetation. The predicted damage that is crossing the street appears to be a connection between two larger damage patches, the model is apparently not trained well enough to avoid roads entirely. In the south eastern part of the scene some false negative areas are visible, in this region the prediction is especially tricky due to the trees looking almost like not damaged, which might actually be an error in the labeling.

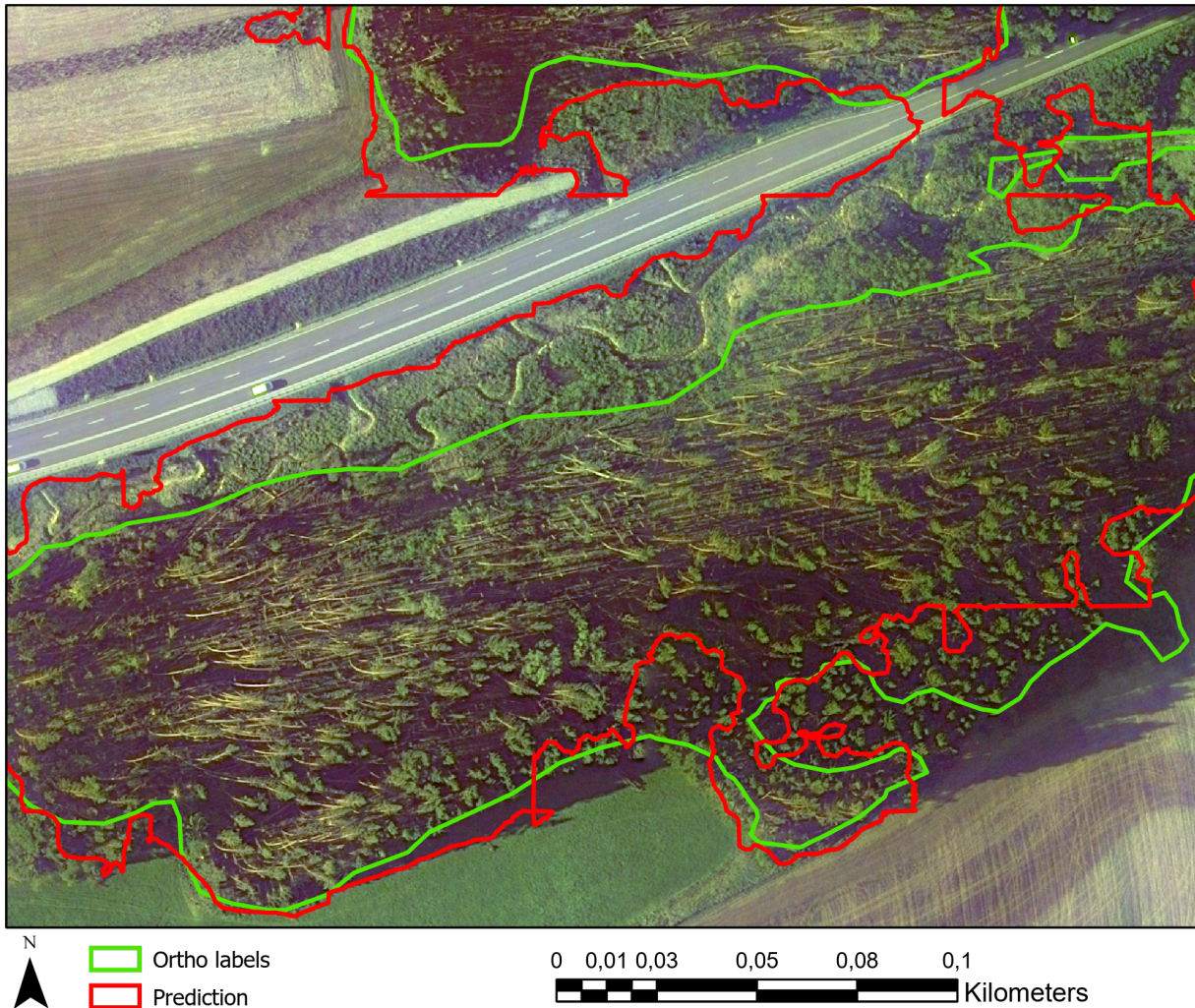


Figure 70: Zoomed in comparison of the ground truth in green against the prediction of a model which was trained on the ortho labels and images in red.

To verify if the model is well trained to avoid false positive pixels in healthy forest, an image scene with a lot of undamaged forest is required. Figure 71 shows such a scene from another of the eight test ortho images. Although a lot of the smaller damage predictions are a bit off from the labels or are entirely composed of false positive, the general prediction matches the labels quite well. There is always the possibility of human errors in the ground truth, especially in large healthy forest with some individual dead trees. These might as well also be caused by regular harvesting operations or simply be trees that have died of old age. Most of the healthy forest is correctly labeled as true negative, which shows that the model is also well trained in this regard.

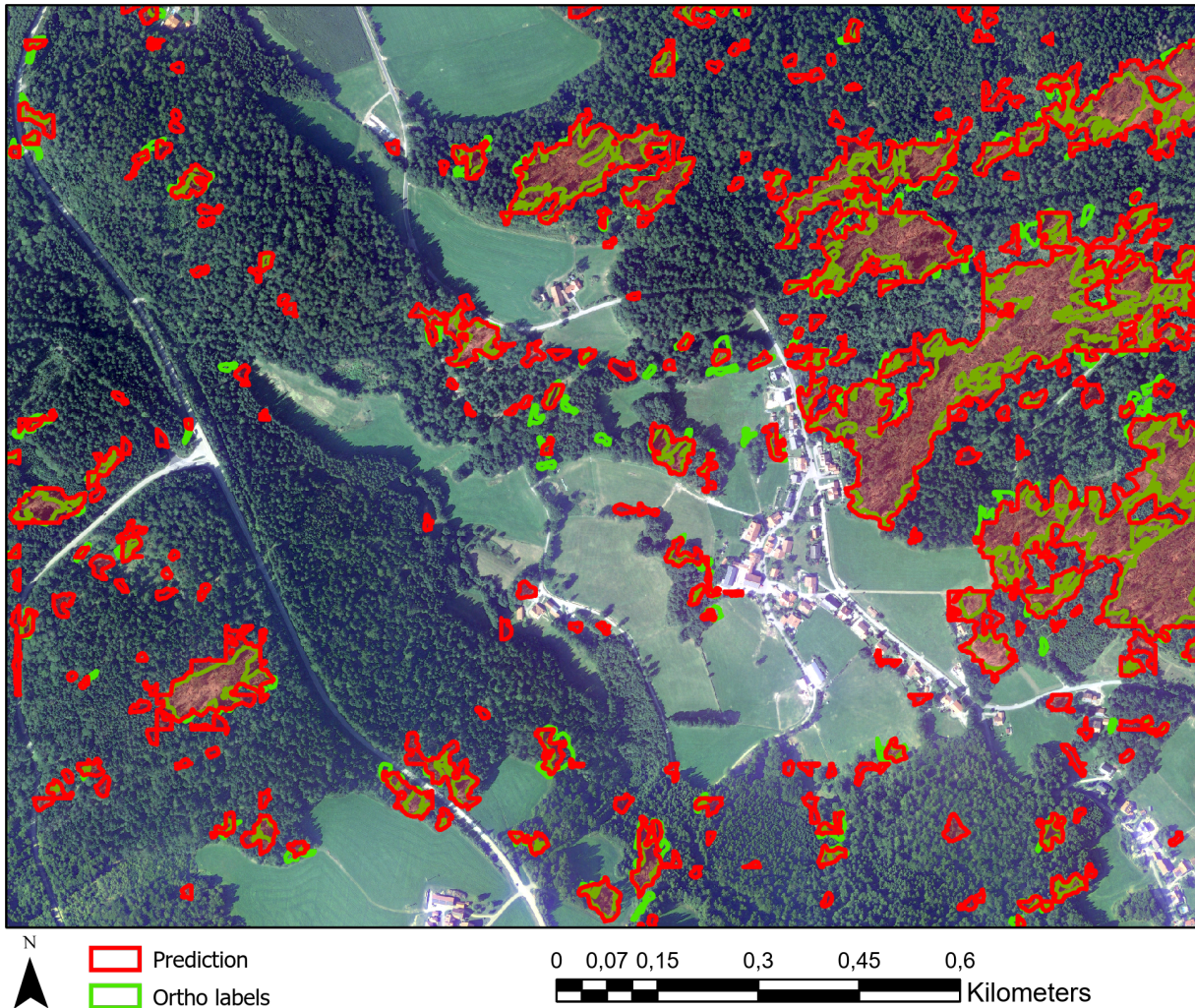


Figure 71: Comparison of the ground truth in green against the prediction of a model which was trained on the ortho labels and images in red

The total area of all damage included in the original ortho labeling which is considered as ground truth is 17.35 km^2 , the ground truth area in the scene in figure 68, which is one of the 45 ortho images, is 0.97 km^2 . The prediction in that scene has a total area of 1.04 km^2 , which is slightly larger than the ground truth. The threshold in the prediction is again chosen to maximize the *IoU* value and could be adjusted so the total area matches better with the area of the ground truth.

5.2.3 Prediction of the transfer learning model

As already mentioned, the *IoU* scores and the accuracy of the VGG19 and the U-Net model are about the same. When looking at the predictions on a larger scale however there are some differences visible. Figure 72 shows an ortho image scene together with the ground truth as green polygons and the VGG19 prediction as red polygons. Especially at the eastern edges of the forest, the prediction consists of small artifacts with mostly false positive pixels, and larger areas are not as consistently filled but contain some false negative holes in them. Also there seem to be more small false positive patches around the villages and in the fields. Noticeable are also the large amount of straight line artifacts, especially in the lower left corner. The inference time for the entire scene is about 1 hour, which is considerably longer than the 5 minutes of the U-Net model.

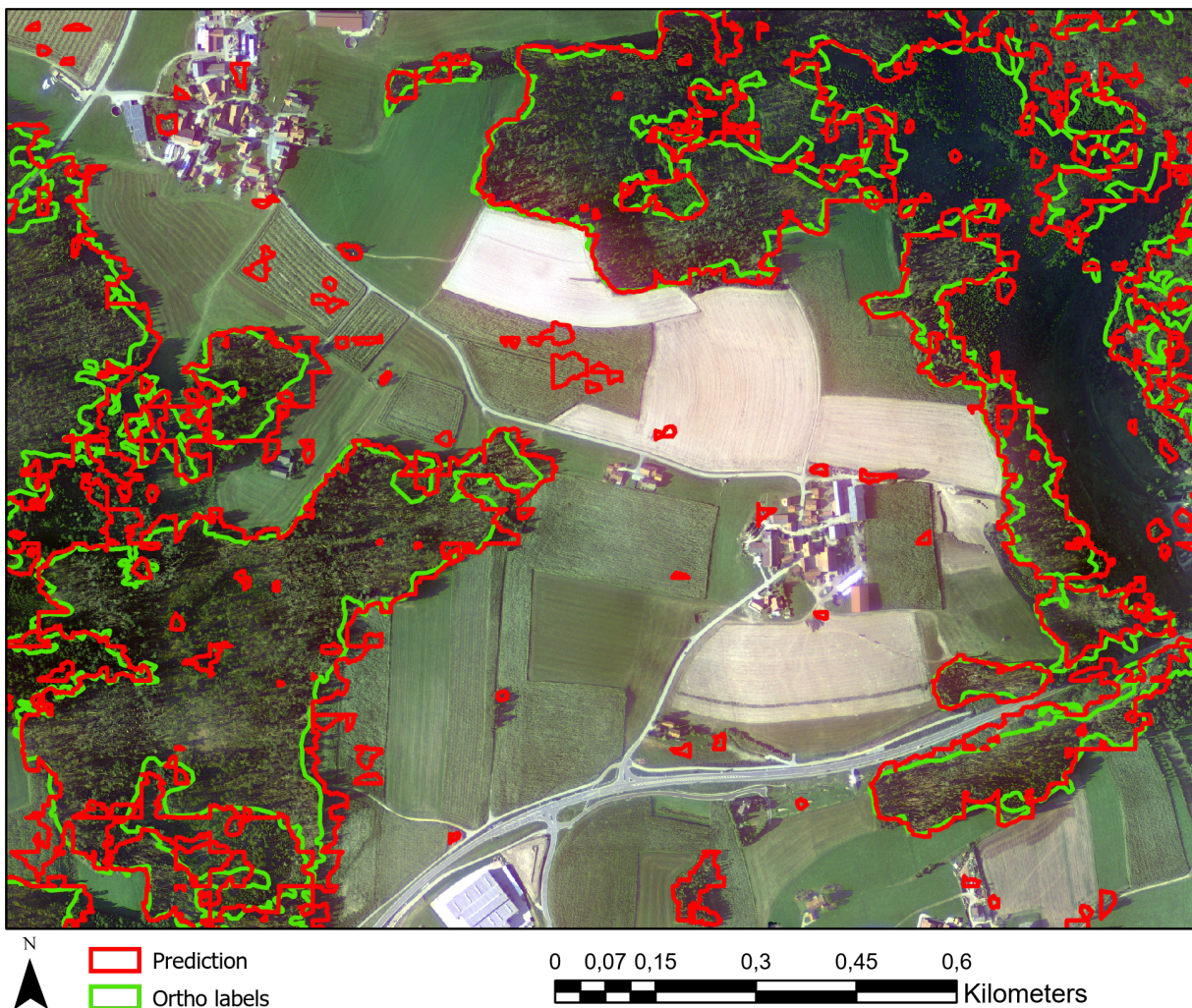


Figure 72: Comparison of the ground truth in green against the prediction of the VGG19 pre-trained model with a U-Net decoding part in red

5.2.4 Transfer ability study using another data set

The satellite images provided by HessenForst include the five image bands blue, green, red, red edge and near infrared. Since the U-Net model is trained on four image bands, only four bands can be used for inference. Since the near infrared band is basically a binary image, the red edge band is used instead. Tests using the near infrared band have shown that the prediction is more or less useless when using the near infrared band. Figure 73 shows the prediction as red polygons together with the ground truth as green polygons for one of the satellite images included in the HessenForst data set. In comparison to the LWF images, a lot more false positive areas are included in the prediction. Especially in the crop fields, a lot of incorrect predictions can be found. Inside the forest itself are many small false negative areas, where the damage was not detected. In the bottom middle part and the top right part at larger damage patches however the prediction seems to be somewhat accurate. The used threshold has a value of 0.1, by reducing it a lot of the false positive areas can be eliminated, but most of the true positive areas do also disappear.

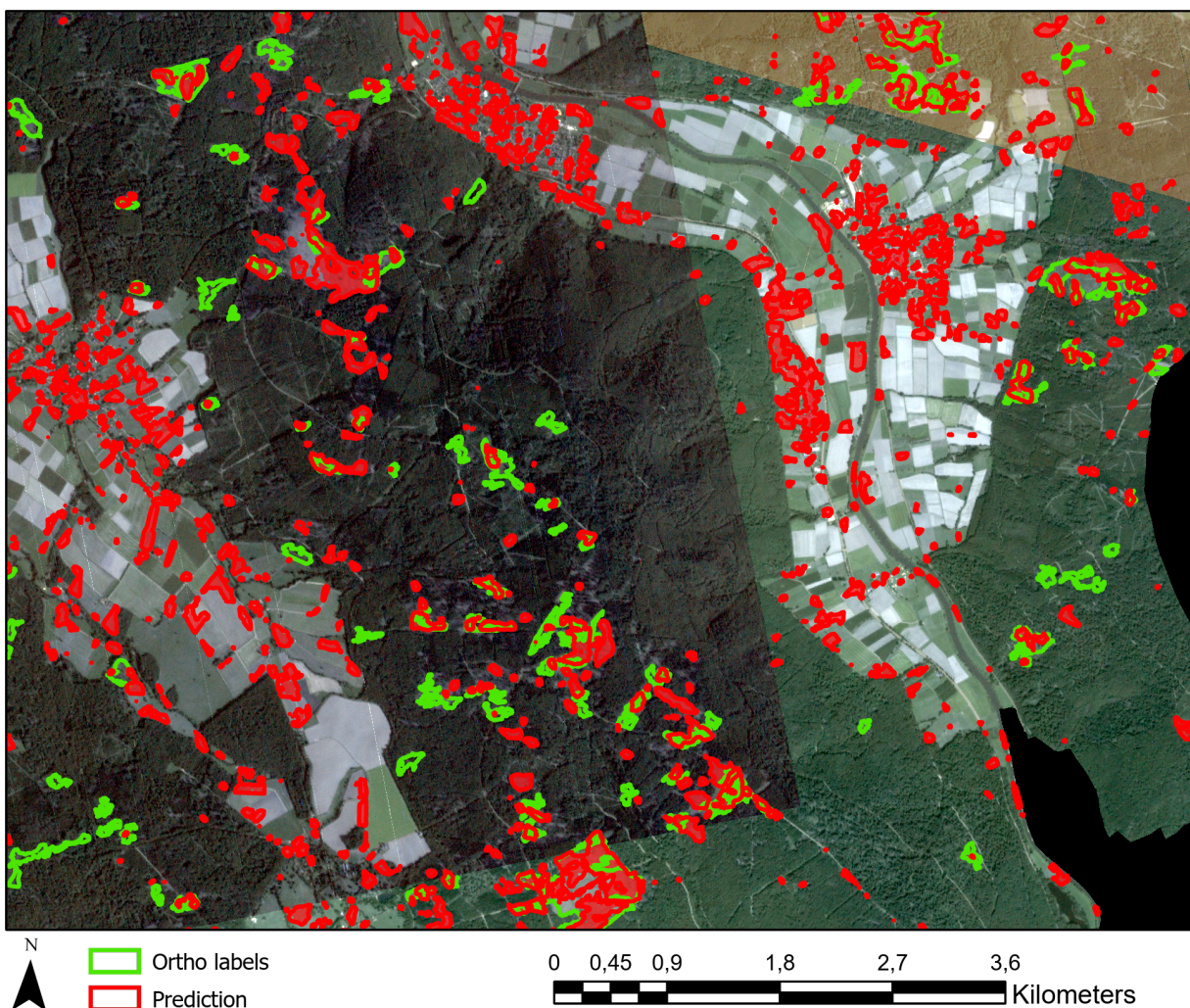


Figure 73: Comparison of the ground truth in green against the prediction of the U-Net model trained on satellite images and labels with a threshold of 0.1 in red. The image is from a data set provided by HessenForst and was not used during training

Figure 74 shows a small area of figure 73 as zoomed in version. The prediction in red is clearly different from the labeling in green. The damage is predicted as false positive in areas that might resemble damage to humans, but are according to the labeling not damaged. The grey areas however can also consist of forest that got damaged before the storm event and are therefore not included in the labeling. Also noticeable are the vertical stripes in the image, these might be results of errors in the satellite sensor.

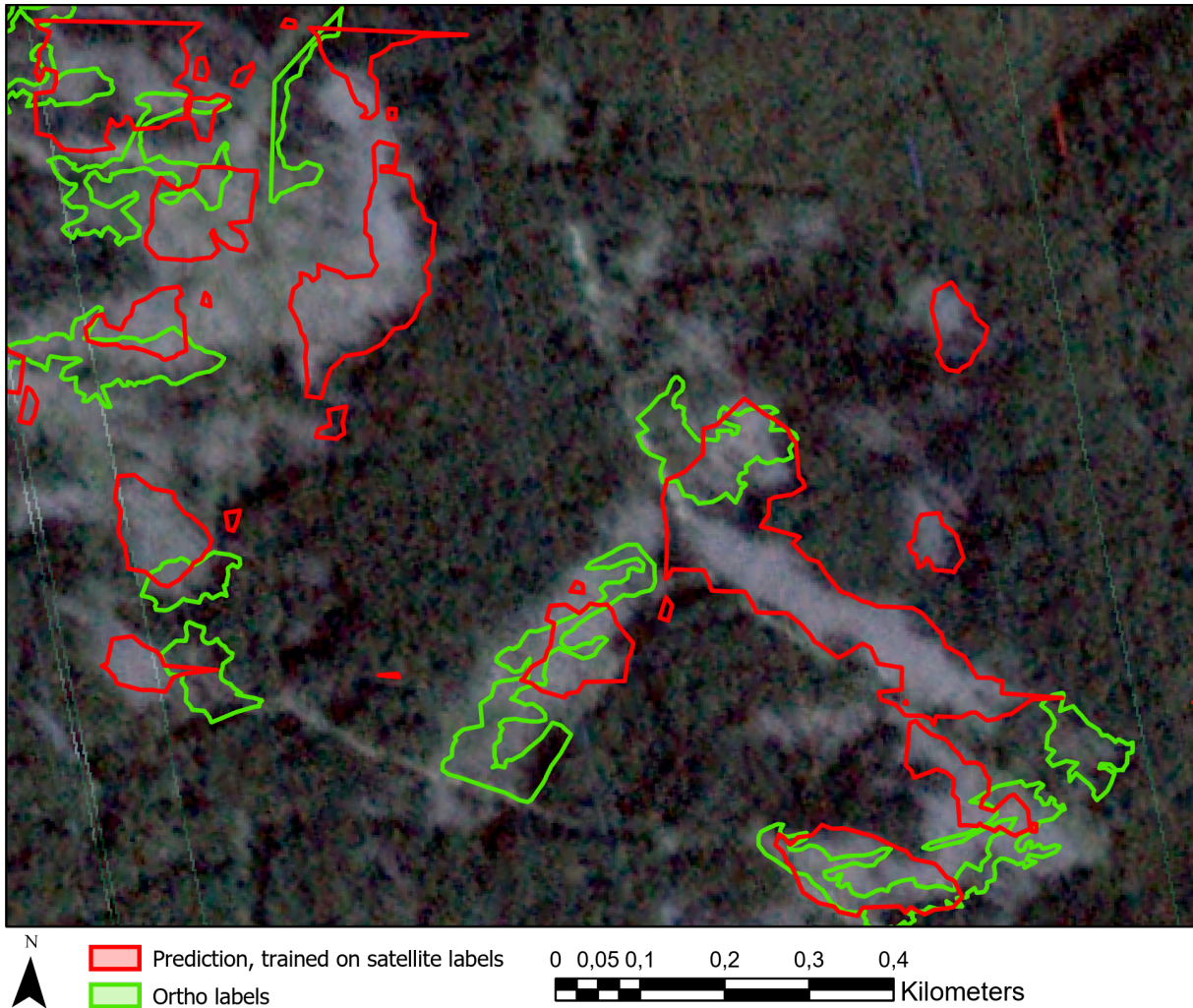


Figure 74: Zoomed in version of figure 73. Comparison of the ground truth in green against the prediction of the U-Net model trained on satellite images and labels in red. The image is from a data set provided by HessenForst and was not used during training

When using the U-Net architecture trained on ortho data on the HessenForst ortho images, the prediction is quite bad and unfortunately not usable. It fails to distinguish between healthy and damaged forest, and even between forest and fields. The damaged areas are a bit more noticeable than rest of the prediction, but over the entire scene patches of damage are detected seemingly randomly. Altering the threshold does not benefit the prediction much, since the true positive areas also tend to disappear along with the false positive areas. The VGG19 model however proves to be more suited for this data set. Figure 75 shows one of the images of the HessenForst data set with the original labels as green polygons and the prediction as red polygons. The true positive rate is quite high with almost no false negative pixels, however there is also a lot of false positive in the prediction. The threshold used for the prediction is 0.3. With the greater number of training data, the VGG19 model is a lot more robust and therefore is able to handle also completely new image scenes. A solution for both models to boost their performance is to train the models a few epochs on the new data to make them accustom to the new color theme and the possible different tree types. This could potentially also make the U-Net architecture viable on the data set.

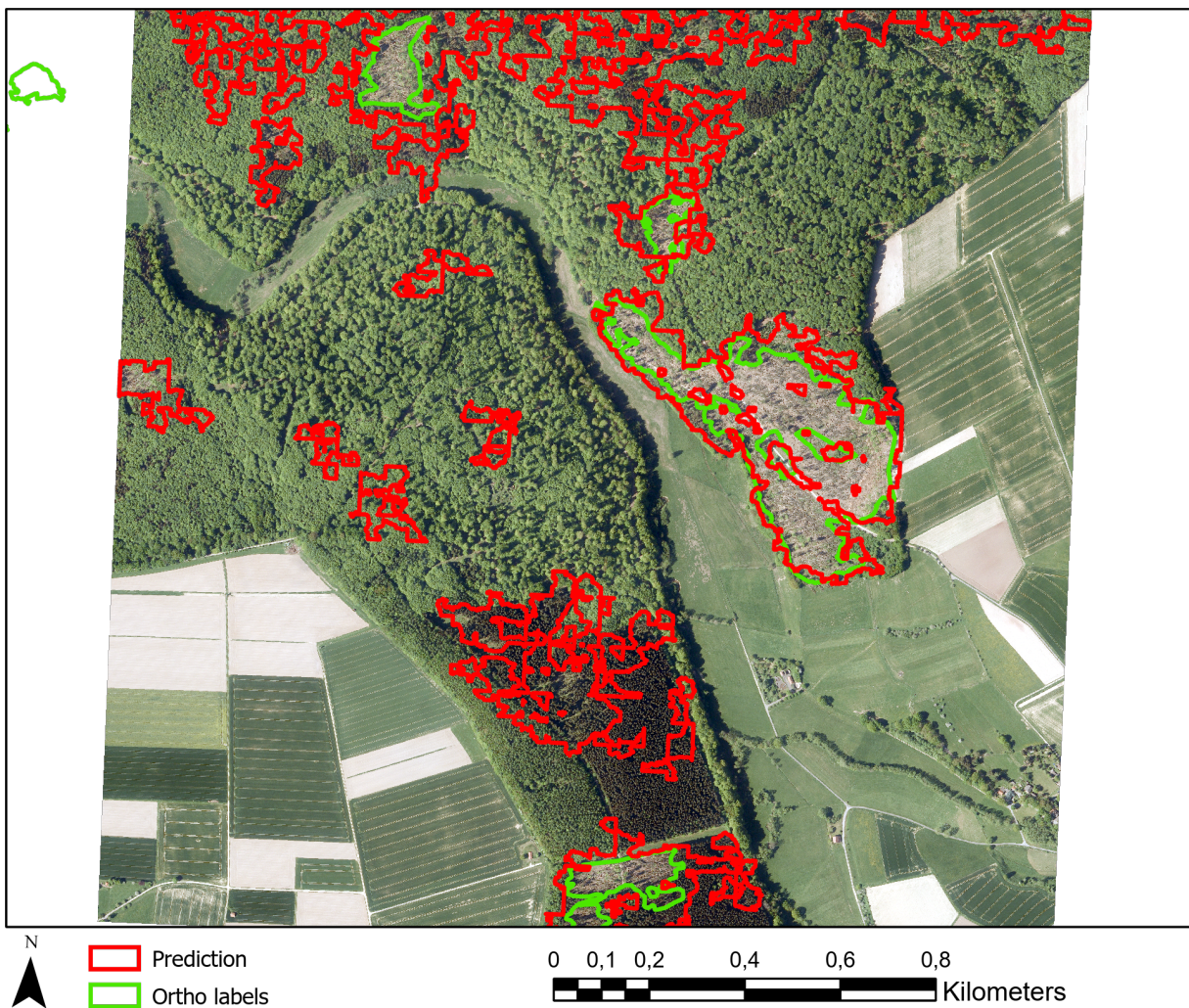


Figure 75: Comparison of the ground truth in green against the prediction of the VGG19 pre-trained model with a U-Net decoding part in red. The image is from a data set provided by HessenForst and was not used during training

6 Discussion

One of the major research questions for this thesis is whether the satellite data set is sufficient enough to train a CNN from scratch in order to acquire rapid predictions of damage caused by future storm events. With their availability of only one or a few days after a storm, satellite images can prove to be a valuable source of information. The downsides of the available satellite images are their small size as well as their rather blurry appearance. The chosen model architecture U-Net however is well suited to be trained on a limited amount of training data, and with the achieved *IoU* score of 0.55 on satellite images and ortho labeling, most of the larger damage areas are successfully detected. The model does have some difficulties on detecting small damage patches, where it often times detects too much or too less, and the exact shape of filigree damage borders can most of the time not be predicted accurately. The total predicted area is larger than the ground truth when setting the threshold in a way so that the *IoU* score is at its maximum, but as the difference between the predictions in the figures 64 and 65 shows, by adjusting the threshold the total area of the prediction can be drastically changed. An interesting finding is that the model trained on satellite images seems to perform better on predicting damage than the human operators that created the satellite labels in the first place, since the prediction is much closer to the actual ground truth. In terms of the meaningfulness of the *IoU* score metric it has to be always considered that the score is a result of the entire test data set, while the predictions in ArcGIS Pro were only performed on one of the three available after storm satellite images. The metric score might be in fact a bit higher when using only the best image, but not enough test data is available to leave out the other two images and still get a meaningful score. Taking also into account the fact that the model trained on satellite images is not only able to distinct between damaged forest and fields or villages, but also between damaged forest and healthy forest, the assumption can be made that it is in fact possible to apply Deep Learning onto satellite images to predict storm damage in forests.

It was already shown by [Hamdi et al., 2019] that aerial ortho images can also be used with great results. The prediction of a model trained on ortho data is far more detailed due to the higher resolution of the images, especially at the edges of the forest. At regions with shadows however the model seems to have difficulties with distinguishing between the shadow and actual damage. The difference of the model in this thesis compared to Hamdi is the greatly increased training data set due to the new labeling provided by the LWF as well as several data augmentation steps, a weighted loss function and the introduction of additional regularization layers in the model architecture. The *IoU* score could be increased from 0.64 to 0.73, which is a rather large step, but the accuracy of 0.92 could not be matched and is with the current model implementation at 0.86. A direct comparison between the two models can only be made if the exact same test data is used, which is not the case. Some data augmentation steps like rotations that create different sun and shadow angles which do not occur in the test data set may even lower the accuracy and *IoU* score of the prediction. It is nevertheless still reasonable to use these augmentation steps, since the robustness of the model is increased, and the idea is not only to reach a perfect score but to also create a useful model for future storm damage predictions.

Testing the model trained on satellite images on satellite images of a different data set, it is shown that the model is in fact able to predict damage, although the accuracy of the prediction is by far lower than the accuracy on the original test data set. Larger damaged areas are detected, but smaller ones are missed, and there are a lot of areas that are incorrectly predicted as damaged. Especially crop fields are

vastly recognized, which is interesting given the fact that the model should take their rectangular shape into account and not label them. Perhaps the big difference in the histogram compositions between the training and the new test data set is too large for the model to handle. The ground truth labeling is also not as accurate as the labeling of the training data, since some areas that actually contain damage are not included because the damage did not originate from the last storm, or the area is not in governmental property. Training the model a few epochs on the new data would probably benefit the prediction greatly and should eliminate most of the incorrect predictions in fields. The U-Net model trained on ortho data however completely fails to create a useful prediction on a different data set, it labels basically every shape like fields, individual trees, houses and roads as damaged areas. In terms of the histograms, the two data sets look rather similar, although the near infrared band curve does not have the same shape. This might be the reason why the prediction is so bad, since the resolution and even the sun angle are the same. This might also be solvable by training the model a few epochs on the new data, but there are probably more epochs necessary than with the model trained on satellite images. Increasing the additional noise during data augmentation would most likely also benefit the model greatly in terms of a higher robustness.

The main issue with tests on a different data set is the lack of robustness in the models. This can either be compensated by additional training data, or by usage of a model that is already trained extensively on a huge database. One such already trained model with the code name VGG19 coupled with an U-Net decoder that is trained from scratch shows that the prediction on the new ortho test data is in fact possible. The decoder of the model was trained on the original data set, which shows that the encoder is robust enough to handle both sets of data. The prediction is however less accurate than the prediction on the original data set with a lot of incorrectly predicted damage, but the main damage areas of the ground truth are discovered. The discrepancy between the prediction and the labeling can to some degree be traced back to the less accurate ground truth labeling. In terms of accuracy on the same data set and in terms of prediction speed, the U-net architecture is superior to the implemented VGG19 model, when it comes to new data without additional training the opposite is the case. A major disadvantage of pre trained models is also their training data set being oriented on 8 bit RGB images and not multispectral 16 bit satellite images that use only a fraction of each band. This makes the VGG19 model only usable for ortho images, for satellite only the implemented U-Net model can be used as of now.

Compared to other state of the art storm damage detection methods that do not involve deep learning, the implemented U-Net model does achieve about equal accuracy. A pixel based change detection method for wind throw in forests in France developed by [Chehata et al., 2014] reaches an accuracy of 87.8%, which is slightly higher than the 0.86 achieved by U-net, but the data of France comes in a resolution of 5 to 10 meters per pixel, which is not easily comparable with the 0.2 meter resolution of the ortho images. The PlanetScope satellite images would be better suited for a comparison, but due to the heavy class unbalancing the achieved accuracy score of 0.93 is not a really reliable value. [Ekstrand, 1996] achieved an accuracy of 80% using Landsat Thematic Mapper data with a resolution of 25 m by using a change detection based approach. The image resolution is unfortunately also too large for a good comparison, but the score is at least in the same range. Compared to UAS data as used by [Mokroš et al., 2017] from a flight height of 700 m with an accuracy of 0.82, the U-net architecture performs slightly better. The described method is based on change detection and can be enhanced using ALS data, which

increases the accuracy to 0.88. Perhaps the most suitable state of the art method to compare the U-Net satellite image prediction with is an approach using IKONOS image data in Switzerland to perform change detection in forests [Schwarz et al., 2001]. The resolution of the data is 4 m, which is not that much larger than the PlanetScope satellite image resolution of 3 m. The achieved accuracy of 0.74 is considerably lower, but the study area in this approach is also a lot more diverse which makes a general damage assessment more difficult. Overall it can be said that both the U-Net model trained on satellite images as well as the model trained on ortho images is comparable in their accuracy to current state of the art wind throw detection methods.

One major advantage however compared to all other described methods is the fact that only one image after the storm is required to create a prediction. This eliminates the need to constantly map and store data for potential storms in the future. With their rapid availability, the PlanetScope images are ideal for a quick and first damage assessment before aerial images are ordered. The inference time of U-Net is also very low, which can reduce the required time for labeling from weeks to a few hours. Even if new data is too inconsistent to the used training data, new models can be trained in a matter of hours or days by using the optimal hyperparameters presented in this thesis in A. This is the greatest advantage of U-Net compared to VGG19, since training such a complex model requires extensive hardware and inference is quite slow.

The successful integration into a modern GIS environment enables an application of the trained models on large scale data. With an easy to use graphical user interface, no programming skills are required to apply Deep Learning on raster data. A processing chain converts the raw prediction into a polygon based data format, that is free of tiny artifacts and that can easily be further processed. For instance the total area of the prediction can directly be read from the data, and intersection with other forest masks can be performed. With the help of cloud based computing, the predictions can be calculated for machines that are by themselves not suitable for Deep Learning for example due to a missing GPU. The ArcGIS environment offers additionally the option to port the predicted damage to other devices like mobile phones, which can be used in the field by operators to verify the prediction. Most state of the art methods lack this kind of integration and ease of usage.

7 Conclusions & Outlook

It has been successfully shown that PlanetScope satellite data can be used in combination with Deep Learning to create rapid predictions of wind throw in forests. The accuracy of the implemented U-Net architecture on satellite images surpasses that of human operators with the prediction being considerably closer to the ground truth compared to the original satellite labeling. Using high resolution aerial images, the prediction can be enhanced further if necessary to match the ground truth even better. By adjusting the threshold value on the prediction, the total area of the prediction can be increased or decreased to suit individual requirements.

The implemented models do not deliver the same accuracy on a different data set, with the satellite prediction being moderately good, while the ortho prediction completely fails. This is compensated for by using transfer learning, with the disadvantage of higher inference time and the incompatibility with satellite images.

With the integration into the modern GIS environment ArcGIS Pro, a tool is created that can easily be accessed and used to create damage predictions. The results are processed using GIS based methods and converted into vector format. The ability of outsourcing the inference to servers and to share the prediction with other devices makes it a powerful GIS tool that may find application in future storm events.

Originally when first conducting this thesis, the idea of labeling individual tree stems to boost the accuracy of the prediction came up as a solution if the performance of the implemented U-Net architecture would not reach the desired level. The proposed method includes the detection of individual trees via a second model specifically trained for this task, and the combination of both predictions inside ArcGIS Pro with for example intersections of buffer zones. The individual tree prediction could be used to fill gaps in the pixel wise area prediction. The detection of individual tree stems would result in a coordinate for every tree, for this task image segmentation is not a suitable approach, an CNN developed for object detection would be required. An example could be the implementation of an R-CNN based model [He, Gkioxari, et al., 2017], which uses region proposals to determine the most likely positions of trees and then detects each individually, or a single shot detector [Liu et al., 2016] which works by adjusting bounding boxes around proposed objects. Since the ortho prediction does not show mayor gaps in regions where individual trees are clearly visible, and the weaknesses of the implemented model are shadowy regions and regions of alternating types of vegetation, it seemed not necessary to implement individual stem detection. For other applications and data sets this might however be a valid option to increase the accuracy of the prediction. Figure 76 shows a small excerpt of the ortho data set with individual trees visibly lying on the ground. The orange polygons on the right show example labeling that could be used for training.

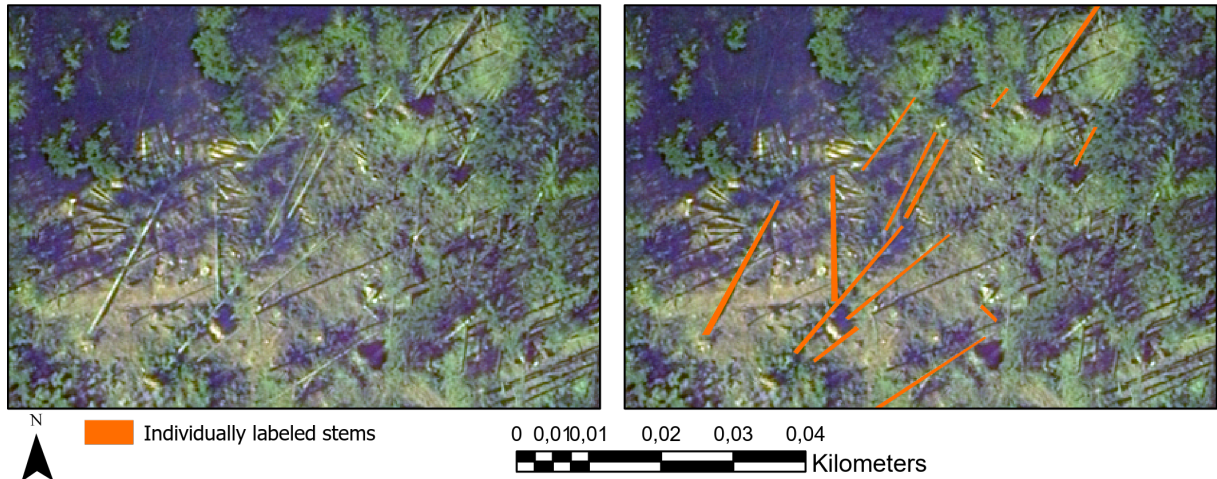


Figure 76: Small excerpt of the ortho data set without labeling (left) and labeling of individual fallen tree stems (right)

By using transfer learning techniques, the U-Net models could also be used to detect other types of forest damage, for instance fire damage. New training data of the effected area would be required, with which the models are trained for a few epochs. This approach saves time and eliminates the need of training new models from scratch.

8 Appendices

8.1 Appendix A: Hyperparameters used during training

The following parameters and hyperparameters are used during data augmentation and training.

- Tile size: 256×256 pixel
- Number of bands: 4
- Additional noise: $\textit{Sigma} = 1.5$ (divided by 256 for normalization)
- Augmentation: Rotations + Mirroring
- Number of filters and blocks: [8, 16, 32, 64, 128]
- Filter size: 3×3 pixel
- Activation function: *ReLU*
- Filter initializer: *LeCun Normal*
- Batch Normalization momentum: 0.9
- Learning rate: 0.002
- Loss function: Weighted binary crossentropy
- Optimizer: Adam
- Metric: Intersection over Union

8.2 Appendix B: Intersection over Union and accuracy of the predictions

The trained models achieve the following **IoU** and accuracy scores at the given threshold.

Image type	Training labels	Test labels	IoU	Accuracy
Satellite	Satellite	Satellite	0.51 (Threshold: 0.05)	0.97 (Threshold: 0.02)
Satellite	Satellite	Ortho	0.50 (Threshold: 0.80)	0.92 (Threshold: 0.44)
Satellite	Ortho	Satellite	0.46 (Threshold: 0.05)	0.97 (Threshold: 0.02)
Satellite	Ortho	Ortho	0.55 (Threshold: 0.26)	0.93 (Threshold: 0.12)
Ortho	Ortho	Ortho	0.73 (Threshold: 0.67)	0.86 (Threshold: 0.58)

8.3 Appendix C: Summary of U-Net

The following table shows the model summary of the implemented U-Net architecture.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 256, 256, 4)	0	
conv2d_1 (Conv2D)	(None, 256, 256, 8)	296	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 256, 256, 8)	32	conv2d_1[0][0]
activation_1 (Activation)	(None, 256, 256, 8)	0	batch_normalization_1[0][0]
alpha_dropout_1 (AlphaDropout)	(None, 256, 256, 8)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 8)	584	alpha_dropout_1[0][0]
batch_normalization_2 (BatchNor	(None, 256, 256, 8)	32	conv2d_2[0][0]
activation_2 (Activation)	(None, 256, 256, 8)	0	batch_normalization_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 8)	0	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 16)	1168	max_pooling2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 128, 128, 16)	64	conv2d_3[0][0]
activation_3 (Activation)	(None, 128, 128, 16)	0	batch_normalization_3[0][0]
alpha_dropout_2 (AlphaDropout)	(None, 128, 128, 16)	0	activation_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 16)	2320	alpha_dropout_2[0][0]
batch_normalization_4 (BatchNor	(None, 128, 128, 16)	64	conv2d_4[0][0]
activation_4 (Activation)	(None, 128, 128, 16)	0	batch_normalization_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 16)	0	activation_4[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 32)	4640	max_pooling2d_2[0][0]
batch_normalization_5 (BatchNor	(None, 64, 64, 32)	128	conv2d_5[0][0]
activation_5 (Activation)	(None, 64, 64, 32)	0	batch_normalization_5[0][0]
alpha_dropout_3 (AlphaDropout)	(None, 64, 64, 32)	0	activation_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 32)	9248	alpha_dropout_3[0][0]
batch_normalization_6 (BatchNor	(None, 64, 64, 32)	128	conv2d_6[0][0]
activation_6 (Activation)	(None, 64, 64, 32)	0	batch_normalization_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 32)	0	activation_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 64)	18496	max_pooling2d_3[0][0]

batch_normalization_7 (BatchNor	(None, 32, 32, 64)	256	conv2d_7 [0] [0]
activation_7 (Activation)	(None, 32, 32, 64)	0	batch_normalization_7 [0] [0]
alpha_dropout_4 (AlphaDropout)	(None, 32, 32, 64)	0	activation_7 [0] [0]
conv2d_8 (Conv2D)	(None, 32, 32, 64)	36928	alpha_dropout_4 [0] [0]
batch_normalization_8 (BatchNor	(None, 32, 32, 64)	256	conv2d_8 [0] [0]
activation_8 (Activation)	(None, 32, 32, 64)	0	batch_normalization_8 [0] [0]
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 64)	0	activation_8 [0] [0]
conv2d_9 (Conv2D)	(None, 16, 16, 128)	73856	max_pooling2d_4 [0] [0]
batch_normalization_9 (BatchNor	(None, 16, 16, 128)	512	conv2d_9 [0] [0]
activation_9 (Activation)	(None, 16, 16, 128)	0	batch_normalization_9 [0] [0]
alpha_dropout_5 (AlphaDropout)	(None, 16, 16, 128)	0	activation_9 [0] [0]
conv2d_10 (Conv2D)	(None, 16, 16, 128)	147584	alpha_dropout_5 [0] [0]
batch_normalization_10 (BatchNo	(None, 16, 16, 128)	512	conv2d_10 [0] [0]
activation_10 (Activation)	(None, 16, 16, 128)	0	batch_normalization_10 [0] [0]
concatenate_1 (Concatenate)	(None, 16, 16, 256)	0	activation_10 [0] [0] activation_10 [0] [0]
alpha_dropout_6 (AlphaDropout)	(None, 16, 16, 256)	0	concatenate_1 [0] [0]
conv2d_11 (Conv2D)	(None, 16, 16, 16)	36880	alpha_dropout_6 [0] [0]
activation_11 (Activation)	(None, 16, 16, 16)	0	conv2d_11 [0] [0]
conv2d_transpose_1 (Conv2DTrans	(None, 32, 32, 16)	2320	activation_11 [0] [0]
concatenate_2 (Concatenate)	(None, 32, 32, 80)	0	conv2d_transpose_1 [0] [0] activation_8 [0] [0]
alpha_dropout_7 (AlphaDropout)	(None, 32, 32, 80)	0	concatenate_2 [0] [0]
conv2d_12 (Conv2D)	(None, 32, 32, 32)	23072	alpha_dropout_7 [0] [0]
activation_12 (Activation)	(None, 32, 32, 32)	0	conv2d_12 [0] [0]
conv2d_transpose_2 (Conv2DTrans	(None, 64, 64, 32)	9248	activation_12 [0] [0]
concatenate_3 (Concatenate)	(None, 64, 64, 64)	0	conv2d_transpose_2 [0] [0] activation_6 [0] [0]
alpha_dropout_8 (AlphaDropout)	(None, 64, 64, 64)	0	concatenate_3 [0] [0]
conv2d_13 (Conv2D)	(None, 64, 64, 64)	36928	alpha_dropout_8 [0] [0]

```

-----
activation_13 (Activation)      (None, 64, 64, 64)  0          conv2d_13[0][0]
-----
conv2d_transpose_3 (Conv2DTrans (None, 128, 128, 64) 36928      activation_13[0][0]
-----
concatenate_4 (Concatenate)    (None, 128, 128, 80) 0          conv2d_transpose_3[0][0]
activation_4[0][0]
-----
alpha_dropout_9 (AlphaDropout) (None, 128, 128, 80) 0          concatenate_4[0][0]
-----
conv2d_14 (Conv2D)             (None, 128, 128, 128) 92288      alpha_dropout_9[0][0]
-----
activation_14 (Activation)      (None, 128, 128, 128) 0          conv2d_14[0][0]
-----
conv2d_transpose_4 (Conv2DTrans (None, 256, 256, 128) 147584     activation_14[0][0]
-----
conv2d_transpose_5 (Conv2DTrans (None, 256, 256, 1) 1153       conv2d_transpose_4[0][0]
=====
Total params: 683,505
Trainable params: 682,513
Non-trainable params: 992

```

8.4 Appendix D: Code implementation

The implemented code, the model definition files, the toolbox for ArcGIS Pro as well as the weights and models themselves are available under the following address:

<https://github.com/WolfgangDeigele/PlanetDeepLearning>

List of acronyms

ALS Airborne Laser Scanning

ANN Artificial Neural Network

CNN Convolutional Neural Network

DSM Digital Surface Model

ESRI Environmental Systems Research Institute

GAN Generative Adversarial Network

GIS Geoinformation System

GNSS Global Navigation Satellite System

GPU Graphic Processing Unit

IoU Intersection over Union

JSON Javascript Object Notation

LRZ Leibniz Supercomputing Centre

LWF Bavarian State Institute of Forestry

ROC Receiver Operating Characteristic

SAR Synthetic Aperture Radar

UAS Unmanned Aircraft system

List of Figures

1	Passive detection method: Change detection via DSMs	3
2	Location of the study area	7
3	Overview of the study area	7
4	Satellite image, August 7 th	9
5	Satellite image, August 15 th	9
6	Satellite image, August 23 th	9
7	Satellite image, August 27 th	9
8	Satellite image, August 30 th	9
9	Histogram of the satellite images	10
10	Ortho image, August 29/30 th	11
11	Ortho image close up, August 29/30 th	11
12	Histogram of the ortho images	11
13	Labels for the satellite data set	12
14	Labels for the ortho data set	12
15	Location of the independent test data set	13
16	Example of the HessenForst data	13
17	Histogram of the satellite images from HessenForst	14
18	Histogram of the ortho images from HessenForst	14
19	Road map of the complete training and prediction process	16
20	Example tile of the satellite data set	18
21	Example tile of the ortho data set	18
22	Split of satellite data set into training and test data	19
23	Split of the ortho data set into training and test sections	19
24	Different sun angles that occur when rotating an image	20
25	Normal distribution of the additional random noise	21
26	Biological and artificial neuron	23
27	2D convolution filter	24
28	Max Pooling operation	26
29	Visual representation of the AlexNet model architecture	27
30	Simple visual representation of overfitting	28
31	The optimum point at which to stop training on order to avoid overfitting	28
32	Diagram of the U-Net architecture	30
33	Calculation of the intersection over union score	32
34	Receiver operating characteristic	34
35	Different IoU scores over time for comparing tile sizes	36
36	Different IoU scores over time for comparing tile sizes, zoomed in	36
37	Different IoU scores over time for comparing block configurations	38
38	Different IoU scores over time for comparing learning rates	39
39	Different IoU scores plotted against the learning rate	40
40	Different IoU scores over time for comparing the effect of additional noise intensities	41

41	Composition of the custom toolbox in ArcGIS Pro	45
42	Graphical user interface of the toolbox	46
43	IoU scores of different training and test data and different epochs	47
44	IoU scores of the best performing epoch with different thresholds	48
45	Test metrics on satellite data set, trained on satellite labels	49
46	ROC-Curve of the satellite prediction, satellite labels for training and test	50
47	Test metrics on satellite data set, trained on ortho labels	51
48	ROC-Curve of the satellite prediction, ortho labels for training and test	52
49	Satellite tile prediction without threshold	52
50	Satellite tile prediction with threshold applied	53
51	Satellite tile prediction without threshold	53
52	Satellite tile prediction with threshold applied	53
53	Training performance on ortho data	54
54	Test metrics on ortho data set	55
55	ROC-Curve of the ortho prediction	56
56	Ortho tile prediction without threshold	56
57	Ortho tile prediction with threshold applied	57
58	Ortho tile prediction without threshold	57
59	Ortho tile prediction with threshold applied	57
60	Test metrics on transfer learning model	58
61	ROC-Curve of the VGG19 ortho prediction	59
62	Different percentages of the data used for training	60
63	Test area for comparing satellite predictions	61
64	Comparison of satellite labels against prediction from model trained on satellite labels	62
65	Comparison of ortho labels against prediction from model trained on satellite labels	63
66	Comparison of ortho labels against prediction from model trained on ortho labels	64
67	Comparison of ortho labels against prediction from model trained on ortho labels in an area of mostly healthy forest	65
68	Comparison of ground truth against prediction from model trained on ortho data	66
69	Zoomed in scene to compare the prediction to different shadow situations	67
70	Zoomed in scene to compare the prediction to roads and forest border	68
71	Comparison of ground truth against prediction from a model trained on ortho data in an area of mostly healthy forest	69
72	Comparison of ground truth against prediction from a transfer learning model	70
73	Comparison of the ground truth of a different data set to the U-Net model	71
74	Comparison of the ground truth of a different data set to the U-Net model, zoomed in	72
75	Comparison of the ground truth of a different data set to the VGG19 model	73
76	Labeling of individual tree stems	78

List of Tables

- 1 Confusion matrix for two classes 32
- 2 IoU scores at epoch 15 for comparing tile sizes 37
- 3 IoU scores at epoch 15 for comparing block configurations 38
- 4 IoU scores at epoch 15 for comparing learning rates 40
- 5 IoU scores at epoch 15 for comparing different noise intensities 42
- 6 IoU scores, thresholds and epochs for comparing different satellite training and test labels 48
- 7 IoU values with different amounts of training data 60

List of Equations

1 Sigmoid function 22

2 Tangens hyperbolicus function 22

3 ReLu function 22

4 Activation function 22

5 Mean Squared Error 27

6 Binary Cross Entropy 27

7 Weighted Binary Cross Entropy 28

8 Adam optimizer 29

9 Accuracy 33

10 Precision 33

11 Recall 33

12 Specificity 33

References

- Beam, Andrew (2017). *Deep Learning 101 - Part 1: History and Background*. URL: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html (visited on 12/20/2019).
- Chehata, N., Orny, C., Boukir, S., Guyon, D., and Wigneron, J.P. (2014). "Object-based change detection in wind storm-damaged forest using high-resolution multispectral images". In: *International Journal of Remote Sensing* 35.13, pp. 4758–4777. DOI: [10.1080/01431161.2014.930199](https://doi.org/10.1080/01431161.2014.930199).
- Chollet, François (2017). *Deep Learning with Python*. Vol. 1. New York: Manning Publications. ISBN: 9781617294433.
- (2019). *Keras: The Python Deep Learning library*. URL: <https://keras.io/> (visited on 12/20/2019).
- Dorland, C., Tol, R. S. J., and Palutikof, J. P. (Nov. 1999). "Vulnerability of the Netherlands and Northwest Europe to Storm Damage under Climate Change". In: *Climatic Change* 43.3, pp. 513–535. ISSN: 1573-1480. DOI: [10.1023/A:1005492126814](https://doi.org/10.1023/A:1005492126814).
- Duda, Richard, Hart, Peter, and Stork, David (2001). *Pattern Classification*. Vol. 2nd ed. USA: Wiley Interscience.
- Einzmann, Kathrin, Immitzer, Markus, Böck, Sebastian, Bauer, Oliver, Schmitt, Andreas, and Atzberger, Clemens (2017). "Windthrow Detection in European Forests with Very High-Resolution Optical Data". In: *Forests* 8.1, p. 21.
- Ekstrand, Sam (1996). "Landsat TM-Based Forest Damage Assessment: Correction for Topographic Effects". In: *Photogrammetric Engineering and Remote Sensing* 62, pp. 151–161.
- Esri (2019). URL: <https://www.esri.de/arcgis/produkte/arcgis-pro> (visited on 12/20/2019).
- Fogg, Andrew (2018). *A History of Machine Learning and Deep Learning*. URL: <https://www.import.io/post/history-of-deep-learning/> (visited on 12/20/2019).
- Foote, Keith (2017). *A brief history of deep learning*. URL: <https://www.dataversity.net/brief-history-deep-learning/> (visited on 12/20/2019).
- Fransson, J. E. S., Walter, F., Blennow, K., Gustavsson, A., and Ulander, L. M. H. (Oct. 2002). "Detection of storm-damaged forested areas using airborne CARABAS-II VHF SAR image data". In: *IEEE Transactions on Geoscience and Remote Sensing* 40.10, pp. 2170–2175. ISSN: 1558-0644. DOI: [10.1109/TGRS.2002.804913](https://doi.org/10.1109/TGRS.2002.804913).
- Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4, pp. 193–202.
- Glorot, Xavier and Bengio, Yoshua (2010). *Understanding the difficulty of training deep feedforward neural networks*. URL: <http://proceedings.mlr.press/v9/glorot10a.html> (visited on 12/20/2019).

- Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron (2017). *Deep Learning (Adaptive Computation and Machine Learning)*. Cambridge: The MIT Press. ISBN: 0262035618.
- Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.
- Google (2019a). *Machine Learning Classification: Accuracy*. URL: <https://developers.google.com/machine-learning/crash-course/classification/accuracy> (visited on 12/20/2019).
- (2019b). *TensorBoard: TensorFlow's visualization toolkit*. URL: <https://www.tensorflow.org/tensorboard> (visited on 12/20/2019).
- (2019c). *TensorFlow, An end-to-end open source machine learning platform*. URL: <https://www.tensorflow.org/> (visited on 12/20/2019).
- Hamdi, Zayd Mahmoud, Brandmeier, Melanie, and Straub, Christoph (2019). "Forest Damage Assessment Using Deep Learning On High Resolution Remote Sensing Data". In: *Remote Sensing* 11.17, p. 1976.
- He, Kaiming, Gkioxari, Georgia, Dollar, Piotr, and Girshick, Ross (Oct. 2017). "Mask R-CNN". In: *The IEEE International Conference on Computer Vision (ICCV)*, pp. 2961–2969. arXiv: [1703.06870](https://arxiv.org/abs/1703.06870).
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian (June 2016). "Deep Residual Learning for Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.
- HessenForst (2019). *Mehr Wald mehr Mensch*. URL: <https://www.hessen-forst.de/> (visited on 12/20/2019).
- Hinton, Geoffrey (2019). *Neural Networks for Machine Learning*. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 12/20/2019).
- Honkavaara, Eija, Litkey, Paula, and Nurminen, Kimmo (2013). "Automatic Storm Damage Detection in Forests Using High-Altitude Photogrammetric Imagery". In: *Remote Sensing* 5.3, pp. 1405–1424.
- ImageNet (2016). *ImageNet*. URL: <http://www.image-net.org/> (visited on 12/20/2019).
- Ioffe, Sergey and Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: [1502.03167](https://arxiv.org/abs/1502.03167). URL: <http://arxiv.org/abs/1502.03167>.
- ISBI 2015 (2019). *International Symposium on Biomedical Imaging*. URL: <https://biomedicalimaging.org/2015/> (visited on 12/20/2019).
- Ivakhnenko, Aleksei Grigorevich and Lapa, Valentin Grigorevich (1966). *Cybernetic predicting devices*. Tech. rep. PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING.
- Kelley, Henry J (1960). "Gradient theory of optimal flight paths". In: *Ars Journal* 30.10, pp. 947–954.

- Kingma, Diederik P. and Ba, Jimmy (2014). "Adam: A Method for Stochastic Optimization". In: arXiv: [1412.6980](https://arxiv.org/abs/1412.6980) [cs.LG].
- Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- LeCun, Yann A., Bottou, Léon, Orr, Genevieve B., and Müller, Klaus-Robert (2012). "Efficient BackProp". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: [10.1007/978-3-642-35289-8_3](https://doi.org/10.1007/978-3-642-35289-8_3). URL: https://doi.org/10.1007/978-3-642-35289-8_3.
- LeCun, Yann, Boser, Bernhard, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne, and Jackel, Lawrence D (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.
- LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.
- Liu, Wei, Anguelov, Dragomir, Erhan, Dumitru, Szegedy, Christian, Reed, Scott, Fu, Cheng-Yang, and Berg, Alexander C. (2016). "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, pp. 21–37. ISBN: 978-3-319-46448-0.
- McCulloch, Warren S and Pitts, Walter (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.
- Mokroš, Martin, Výbošťok, Jozef, Merganič, Ján, Hollaus, Markus, Barton, Iván, Koreň, Milan, Tomašík, Julián, and Čerňava, Juraj (2017). "Early Stage Forest Windthrow Estimation Based on Unmanned Aircraft System Imagery". In: *Forests* 8.9, p. 306.
- Nyström, Mattias, Holmgren, Johan, Fransson, Johan E.S., and Olsson, Håkan (2014). "Detection of windthrown trees using airborne laser scanning". In: *International Journal of Applied Earth Observation and Geoinformation* 30, pp. 21–29. ISSN: 0303-2434. DOI: <https://doi.org/10.1016/j.jag.2014.01.012>.
- Olivas, Emilio Soria (2009). *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*. IGI Global.
- PlanetLabs (2019). URL: <https://www.planet.com/> (visited on 12/20/2019).
- Rezatofghi, Hamid, Tsoi, Nathan, Gwak, JunYoung, Sadeghian, Amir, Reid, Ian, and Savarese, Silvio (June 2019). "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 658–666.

- Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, pp. 234–241.
- Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.
- Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. arXiv: 1609.04747. URL: <http://arxiv.org/abs/1609.04747>.
- Rüetschi, Marius, Small, David, and Wasser, Lars T. (2019). "Rapid Detection of Windthrows Using Sentinel-1 C-Band SAR Data". In: vol. 11. *Remote Sensing* 2, p. 115.
- Rumelhart, David, Hinton, Geoffrey, and Williams, Ronald (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.
- Samuel, Arthur L (1959). "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3, pp. 210–229.
- Schmoeckel, J. and Kottmeier, C. (2008). "Storm damage in the Black Forest caused by the winter storm "Lothar" – Part 1: Airborne damage assessment". In: *Natural Hazards and Earth System Sciences* 8, pp. 795–803.
- Schwarz, M., Steinmeier, Ch., and Waser, L. (May 2001). "Detection of storm losses in alpine forest areas by different methodic approaches using high-resolution satellite data". In: *Proceedings of the 21st EARSeL Symposium: Observing our Environment from Space: New Solutions for a New Millenium*, pp. 251–257.
- Seitz, Rudolf and Straub, Christoph (2017). "Neue Horizonte für die Fernerkundung". In: *LWF aktuell* 115. URL: https://www.lwf.bayern.de/service/publikationen/lwf_aktuell/173144/index.php.
- Sharma, Avinash (2017). *Understanding Activation Functions in Neural Networks*. URL: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0> (visited on 12/20/2019).
- Smith, Gary S. (2019). *Digital Orthophotography and GIS*. URL: <http://proceedings.esri.com/library/userconf/proc95/to150/p124.html> (visited on 12/20/2019).
- Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew (June 2015). "Going Deeper With Convolutions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9.
- Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior (2014). "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708.
- Tayyebi, Amin (2019). *High Resolution Land Cover Mapping using Deep learning*. URL: <https://medium.com/geoai/high-resolution-land-cover-mapping-using-deep-learning-7126fee571dd> (visited on 12/20/2019).

Turing, Alan Mathison (1948). *Intelligent machinery*.

Wong, S. C., Gatt, A., Stamatescu, V., and McDonnell, M. D. (Nov. 2016). "Understanding Data Augmentation for Classification: When to Warp?" In: *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–6. DOI: [10.1109/DICTA.2016.7797091](https://doi.org/10.1109/DICTA.2016.7797091).