ETH zürich

IGP
Institut für Geodäsie und Photogrammetrie

Change Detection in Tropical Rainforests by Using Optical Satellite Images

Luca Miotti

Interdisciplinary Project

Autumn Semester 2018

Professorship

Prof. Dr. Konrad Schindler

Supervisors

Nico Lang

Andrés Rodríguez

Dr. Jan Dirk Wegner

One of the biggest environmental problems of our time is the deforestation of tropical rainforests. Some of the biggest forest areas of our planet continue to diminish causing impact on climate change in the long run as well as the loss of various species in flora and fauna. A lot of these deforestations are executed illegally which is not always easy to monitor, especially for smaller areas. Luckily there is an abundance of satellite data available, but checking these datasets manually is very time consuming and thus not really practical. So a good way to solve this problem would be an automated detection of change in rainforests by using satellite data.

One way to achieve this could be with the help of deep learning. Deep learning uses a so called artificial neural network, somewhat resembling the neural network of a human brain. These networks can be trained in various different ways to achieve a certain task. So in this project a Convolutional Neural Network (CNN) is implemented to detect deforestation in tropical rainforests between two points in time.

To detect deforestation any type of remote sensing data could be used. Each of them providing certain advantages. In this project the training of the network is done by using optical image data. This data is provided by the Sentinel-2 mission of the European Space Agency (ESA) and is freely available.

Table of Contents

Table of Figures

1. Introduction

1.1. Motivation

Over 31% of the total land mass of our planet is covered by forests. They are home to millions of species of flora and fauna, serve as a vital part in the cycle of the planets' atmosphere and play an important role to balance climate change. Unfortunately in modern times this area is constantly decreasing. Every year over 75'000 km^2 of forest area are lost worldwide. A large amount of these losses occur in tropical rainforests where the space formerly occupied by forests is used to build huge plantations for agriculture crops. This often happens in countries with large land masses and not enough capacity to closely monitor all ongoing legal and illegal deforestation. [WWF, 2018]

Automated systems monitoring deforestation with satellite data could be the answer to that problem. A way to implement such automation could be with the help of deep learning and neural networks. Such a system could track the forests of huge areas in intervals much higher than manually possible and therefore allow to easily detect illegal deforestation.

Figure 1: Deforestation in South East Asia [National Geographic, 2015]

1.2. Goals and Procedure

The goal of this project is to implement a Convolutional Neural Network to detect changes in native rainforests. To detect such changes the network is trained on optical satellite data obtained from the Sentinel-2 mission operated by the European Space Agency (ESA). The dataset includes several optical channels showing an area at two different points in time.

One of the biggest issues when using optical remote sensing data is cloud coverage. It will be tested if the use of optical data is suitable for such a monitoring project and what predictions can be achieved. The influence of cloud coverage is discussed and the achieved accuracies are analyzed.

The whole procedure includes the following steps:

- Obtaining the satellite data and proceeding it
- Transforming the data into a suitable form to serve as input for a neural network
- Creating the network architecture
- Training the network on the data
- Evaluation of the results

2. Used Software

In the context of this project the following software is applied:

QGIS

An free open source geographic information system. Amongst other things it was used to transform all data into the same coordinate reference system.

Python

Python is a high level, interpreted programming language. It allows the import of hundreds of different modules and libraries, each expanding the possibilities of the user. Some of the modules used during this project include:

- Gdal: A translator library for geospatial data formats.
- Tensorflow: An open source dataflow framework often used in the field of machine learning.
- Keras: A neural network library used as backend for Tensorflow.

Sentinelhub

The online platform provided by ESA to browse through and download all available data from the satellite missions Sentinel-1 and Sentinel-2.

Snap

Snap (Sentinel Application Platform) is a collection of several toolboxes provided by ESA to process and export data from the Sentinel Mission. It includes the Sen2Cor plugin. This plugin can be used to apply atmospheric corrections to raw satellite images.

3. The principle of Convolutional Neural Networks

In recent times Convolutional Neural Networks have become increasingly popular and wide spread in the fields of image classification and segmentation. Like already mentioned, such a network is also applied during the course of this project and therefore some theoretical aspects are highlighted in this chapter. This chapter is based on the Stanford course CS231n: Convolutional Neural Networks for Visual Recognition [Stanford, 2018]. This also includes the shown figures if not mentioned otherwise.

3.1. Idea of a Neural Network

An artificial neural network is somewhat inspired by the structure of biological neural networks consisting of a multitude of connected neurons. A neuron receives certain input signals and transform them into an output signal. Such a single neuron can be seen in Fig. 2. The combination of several neurons thus allows to transform an input into a certain output by applying multiple single transformations. This is the basic principle also used in artificial neural networks.

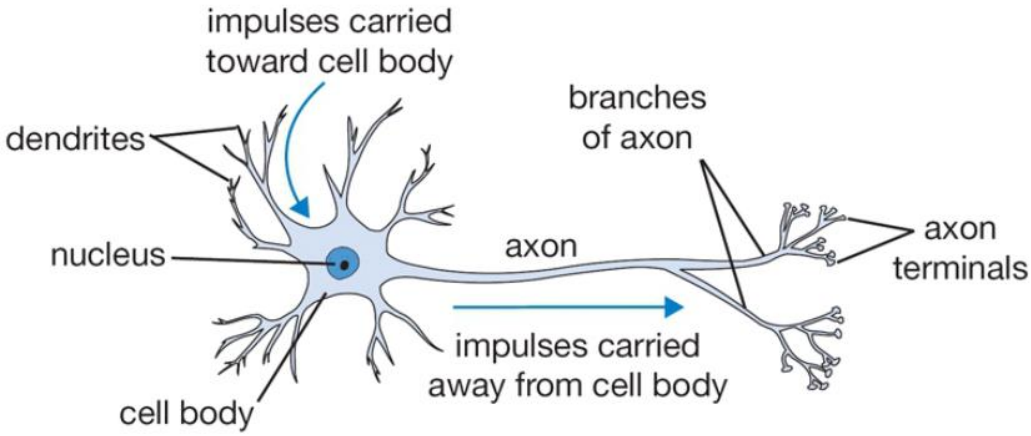

Figure 2: A single biological neuron

In an artificial neural networks several neurons are combined to a layer. A network then consists of one or several layers. The amount of layers is mostly dependent on the task the network is supposed to perform. Adding more layers usually improves the accuracy and the capabilities of the network but also can have its disadvantages. This concept can be seen in the following figure.

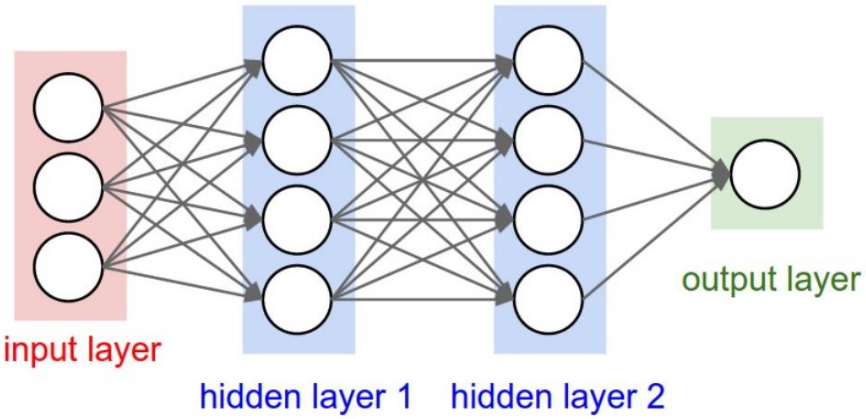

Figure 3: A neural network with 2 hidden layers each consisting of 4 neurons

In this example there are a total of eight neurons split up into two layers. Note that the amount of neurons in each layer does not have to be the same. Each neuron receives multiple input values and calculates a new value which then serves as an input value for the neurons in the following layers. This is repeated until the network concludes at the output layer.

For the computation of the output of a neuron each input value receives a certain weight. These weighted inputs are then summed up and an neuron specific bias is added:

$$x = \sum input_i * weight_i + bias$$

For a neuron in the second layer in the above mentioned figure for example, there are four weights and one bias resulting in five unknown parameters. Because Neural Networks are often used to perform complicated tasks, a system of such linear equations mostly is not capable enough. That is why an activation function is used for every neuron. This function computes the output of a neuron from the above mentioned value. It introduces non-linearity and thus allows for more difficult problems to be solved. Often used activation functions include ReLu, Softmax or Sigmoid.

The idea of a neural network is now to determine all of the parameters of a network in such a way, that a certain input results in a certain output. If we have a network that should distinguish between dogs and cats for example, we want that the network recognizes a dog regardless if it had seen before. To achieve this the network is trained on a certain data set. Once the training period is complete it can perform its assigned task pretty fast. The challenging part of using a neural network to perform a certain task is thus the training.

3.2. Training of the Network

For applications in image recognition the method used to do the training is mostly called gradient descent. There are other methods including unsupervised learning and reinforcement learning, but these are generally used in other fields. Gradient descent is a supervised learning method meaning that there is a need for labeled training data to complete the training.

All the weights and biases of all neurons are combined in what is a called a loss function. This loss function determines a value for each input of the network, indicating how correct the network transform the input into the desired output. If we go back to the cats and dogs example, each input would be a picture of either a cat or a dog and the output the probability of the species:

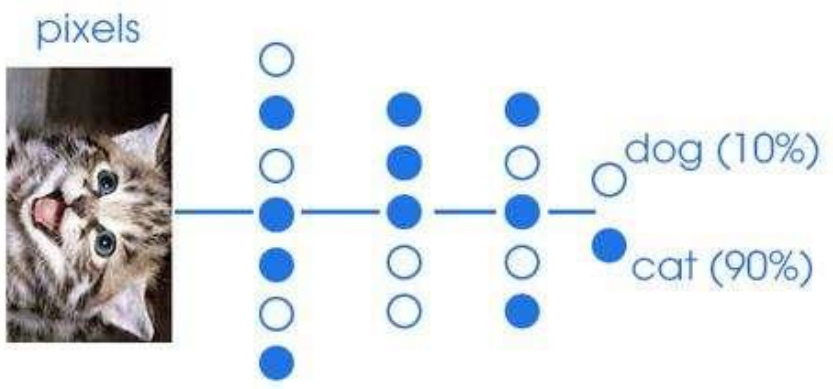

Figure 4: Graph of the cats vs dogs network [https://www.eetimes.com/document.asp?doc_id=1322696]

The correct label in Fig. 4 would obviously be cat. But since the network predicts cat with a probability of 90% and not 100%, there is a certain loss or penalty. How big this loss exactly is depends on the used loss function. The goal of the training is therefore to continuously adapt the weights and biases to minimize the loss for each input without significantly increasing the losses of earlier inputs.

This is done by analyzing the gradient of the loss function and adapting the parameters in the direction of the most negative gradient. Hence the name gradient descent. The size of the step in that direction is called the learning rate and is not always easy to define, since it often depends on the specific task of the network. If it is chosen too large the minimum of the gradient can easily be missed and possibly be never achieved during the training. If it is too small the function can be stuck in a local minimum. There are numerous variations of the gradient descent optimization, tweaking the algorithm and improving its results. But the overall principle does not change a lot. Such variations include stochastic gradient descent, Adam or Adagrad.

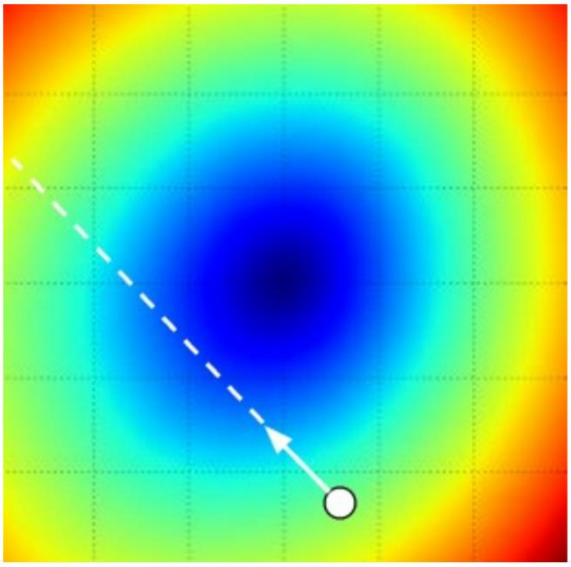

Figure 6: Graphical Example of a Gradient

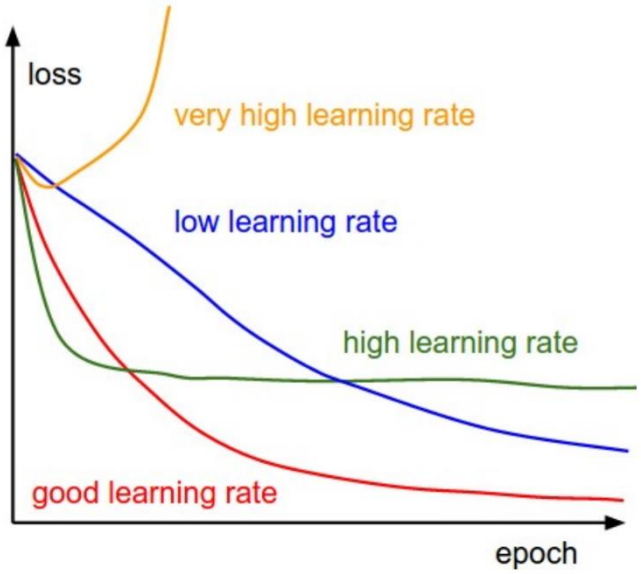

Figure 5: Different learning rates

A complete training phase is performed over several epochs. In each epoch every input is used once, often at an random order. The number of epochs can be chosen freely but once again is depending on the task of the network and the characteristics of the data. Due to computational reason each epoch is further split into different batches, each containing a small part of the whole data set.

For more details and mathematical background please refer to the stated Stanford Course.

3.3. Convolutional Neural Networks

Convolutional Neural Networks are very similar to conventional Neural Networks. In fact all the already described properties also apply here. The main problem for image classification with regular neural networks is that with growing image size the networks become less and less useable. So Convolutional Neural Networks are specifically designed for image classification regardless of the image size. The biggest difference between the networks is the connection of the neurons with each other. Where in normal neural networks each neuron is connected to all neurons from the previous and the following layer, in convolutional networks this is not the case. This is compensated by the introduction of a third dimension in each layer: the depth or receptive field, which would relate to the three colour channels of an image for the input layer.

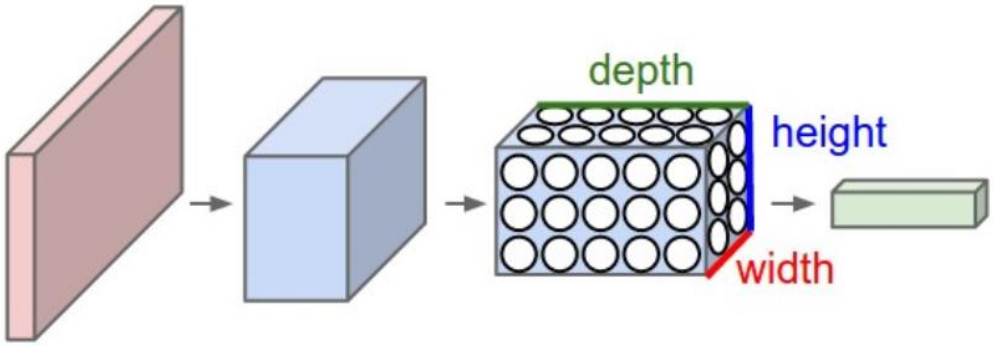

Figure 7: The layers of a Convolutional Neural Network

A neuron is now connected only to a smaller area of the previous layer in the two spatial dimensions, but always along all depth dimensions. The size of the depth dimension varies from layer to layer depending on the amount of filters chosen in the previous layer. For the last layer the depth corresponds to the number of classes.

A Convolutional Neural Network is made up by the following types of layers.

- Conv layer
- ReLu layer
- Pool layer
- Fully connected layer

Note that in this project not all of the layers are used, since the goal is an image segmentation instead of a classification and therefore the spatial dimension of the images are not allowed to change.

3.3.1. Conv layer

The convolutional layer is the most import layer in a CNN. It is the basic building block and applies, as the name suggest, a convolution on the input layer. To be fully described it requires 4 hyperparameters.

- The number of applied filters F: This amount is equal to the depth of the output layer
- The spatial extent of the convolution kernel K
- The size of the stride S: Describes by how many pixels the kernel is shifted
- Amount of zero padding P: Can be used to control the spatial dimension of the output layer

A simple convolution with a kernel size of three and one filter can be seen in Fig. 8. It is important to note that this convolution only includes the size of the kernel in the spatial dimensions but all depth layers.

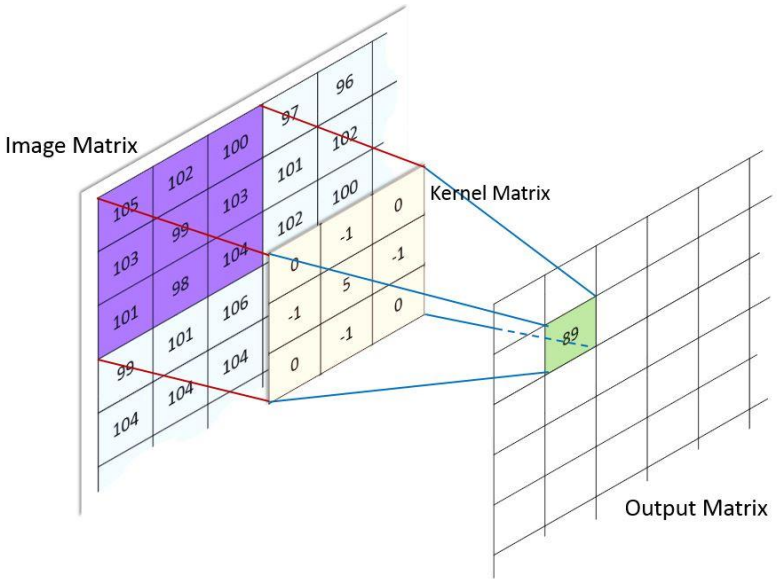

Figure 8: A simple example of a convolution

The dimensions of the output layer can now be calculated like this:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1$$

$$D_{out} = F$$

3.3.2. ReLu layer

ReLu is an activation function. As already mentioned an activation function introduces the concept of nonlinearity into neural networks and transforms the values of an output layer into a certain range. It is therefore always and only applied after a convolutional layer. In the case of ReLu it just sets the negative values to zero.

$$f(x) = \max(0, x)$$

3.3.3. Pool layer

A pool layer is used to perform a down sampling along the spatial dimensions. It is used to reduce the amounts of parameters needed, allowing for advantages regarding computational time and overfitting. Since it is performed for every channel separately the depth of the layer will not be altered. The usual pooling method is called max pooling, which just adopts the maximum value inside the kernel and requests two parameters.

- The spatial extent of the kernel
- The size of the stride

The size of the following layer can be calculated equivalent to the already described convolution by just ignoring the pooling.

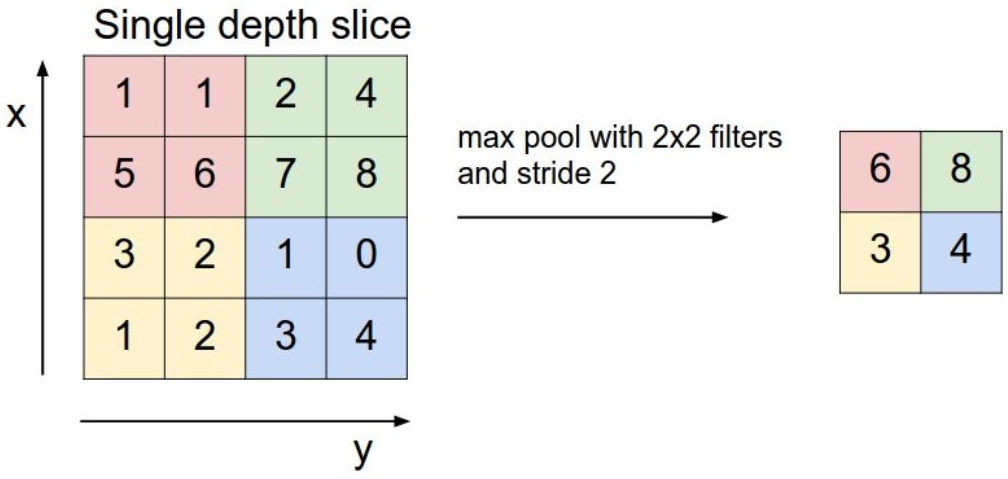

Figure 9: Concept of max pooling

3.3.4. Fully connected layer

Usually the last layer in a Convolutional Neural Network and not used anywhere else. Unlike in the convolutional layer every neuron is connected to every neuron from the previous layer. A normal procedure would include flattening the last used convolutional layer and then adding a fully connected layer. This fully connected layer has the same amount of neurons as there are classes. By flattening the convolutional layer and connecting it to the last layer, we can distribute its values into the class probabilities and thus let the network terminate.

4. Data and Processing

4.1. Raw data

All the data used in this project originates from the Sentinel-2 mission. This mission is composed of a pair of optical earth observation satellites and is operated by the European Space Agency (ESA). It has started in 2015 and is planned to last for seven years. All the collected data is publicly available at their web archive browser. It is delivered in the xml format and contains numerical values expressed in 16 bit, allowing a much higher range of values and therefore a highly precise numerical representation. [https://www.sentinel-hub.com/, 2018].

All in all the data consists of 13 bands or channels distributed over the whole spectrum of optical wavelengths. Due to several bands in the infrared channel, Sentinel-2 data is especially suitable for remote sense application with objects regarding vegetation. Fig. 10 gives an overview over all the bands, their respective purpose, wavelength and ground resolution.

Sentinel-2 Bands	Central Wavelength (µm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.490	10
Band 3 - Green	0.560	10
Band 4 - Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.740	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A - Vegetation Red Edge	0.865	20
Band 9 - Water vapour	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.610	20
Band 12 - SWIR	2.190	20

Figure 10: All the Sentinel-2 Bands [Satellite Imaging Corporation, 2017]

For this project all channels with a ground resolution of 10 and 20 meters are utilized, resulting in a total of 10 used channels.

4.2. Region of Interest and Labels

The used region of interest lays in central Malaysia. As already mentioned above, the changes in native rainforests between two points in time are analyzed. These two points are in December 2015 and December 2016 respectively. This specific area and times were chosen because labeled changes for these points in time were already available. Furthermore because of the existing native rainforests and intensive ongoing deforestation in this area, its properties agree well with the goal of this project. In fact Malaysia has one of the highest deforestation rates in the world. [www.climatechangenews.com, 2013].

Fig. 11 shows the location of the region at a larger scale.

Figure 11: Location of the region of interest in Malaysia [Google Earth Pro, 2018]

In Fig. 12 we can see closer images of the area at the two points in time. These images are generated from the Sentinel-2 data. The data used in this project was recorded on 01.12.2015 and 15.11.2016. At both points in time a significant part of the area is covered by clouds. This large cloud coverage is obviously not ideal when using optical satellite data. But since the distribution of the coverage is quite similar and better data relating to the labels is simply not available, this dataset is used.

The whole area constitutes of roughly 12'000 km^2. It consists of 10980x10980 pixels at the highest resolution, with each pixel corresponding to a size of about 100 km^2.

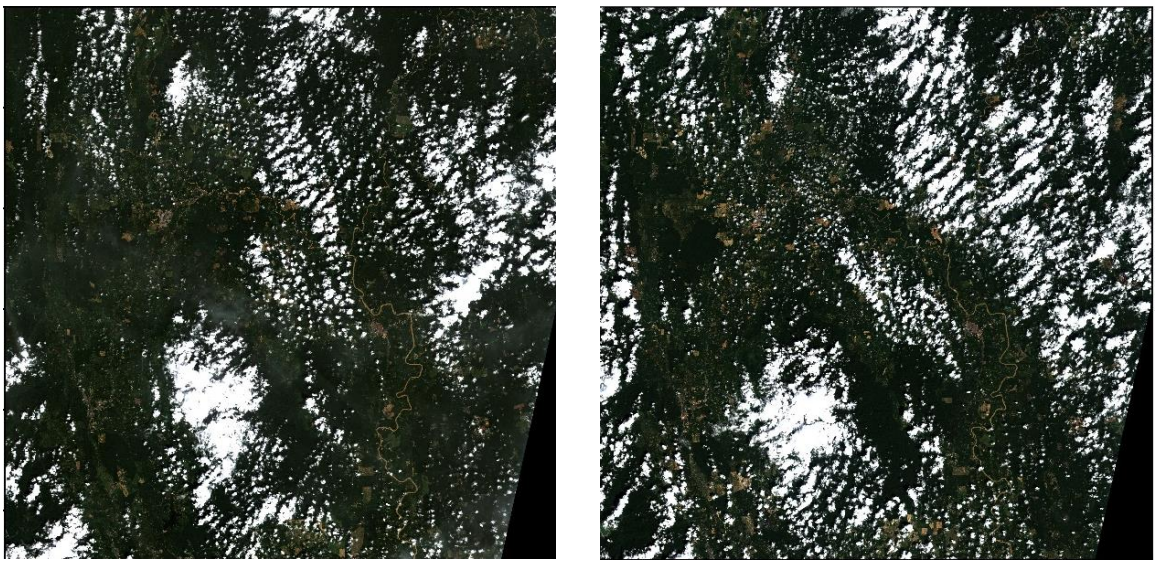

Figure 12: Region of interest at both points in time

In Fig. 13 the already mentioned labeled changes can be seen. This is a binary map indicating an occurring deforestation between December 2015 and December 2016.

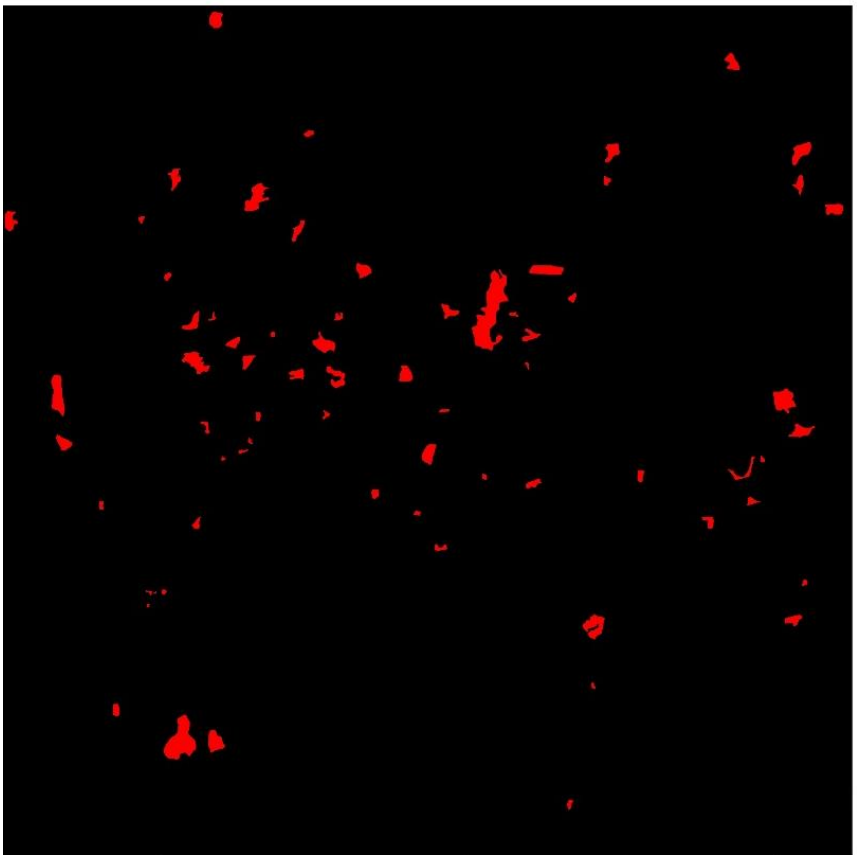

Figure 13: Labels of deforestation

4.3. Processing

To make the satellite data usable as an input for a neural network, several processing steps have to be performed. The whole workflow includes:

- Applying atmospheric corrections and creating GeoTiffs
- Transforming the labeled changes into the coordinate system of the optical data
- Resampling all the data and crop it to a certain bounding box
- Stacking all the channels
- Splitting the data into training, validation and test data
- Applying image pre processing
- Creating random patches with the training and validation data
- Creating sorted patches with the test data

Applying atmospheric corrections

In this step all atmospheric influences on the raw data are removed and a cloud mask indicating the probability of a pixel being covered is created. This is done with the help of sen2cor. Sen2cor is a plugin for ESA's software Snap but can also be applied without this very program. Furthermore the data is exported into GeoTiffs. Each band is exported separately resulting in 20 different files.

Transforming the labels into the coordinate system of the optical data

Since the labels are created in the global WGS 84 reference system and the Sentinel-2 dataset in the locally adapted system WGS 84/ UTM zone 47N, the labels have to be transformed into this local system. This is done by using the already implemented function of the geographical information system QGIS.

Resampling of the data

Because of the different spatial resolutions of some of the bands (see chapter 4.1.), some of those bands and the labels have to be resampled to correspond to the highest spatial resolution of 10 meters. Additionally the labels are cropped to the exact same size as the used Sentinel-2 dataset. To achieve this empty Tiffs with the same resolution and projection as the high resolution files are being created. The lower resolution files are then reprojected onto these files by using a bilinear interpolation.

Stacking the channels

All the channels as well as the labels do now have the same resolution and spatial dimensions and can be converted into numerical arrays. To make them usable as an input for the network, the band arrays get stacked on each other resulting in a single matrix. This matrix has the size of 10980x10980x20 pixels. Corresponding to the spatial dimensions and all the channels at both points in time. As seen in chapter 3.3. a matrix with two spatial and a depth dimension can be easily used as an input for a convolutional neural network.

Split the data into training, validation and test sets

To validate the accuracy and quality of a neural network during and once the training is complete, the dataset is split up into three subsets.

- Training set: This is the data set the network is actually trained on. It is by far the biggest one and normally makes up about 70% of the whole dataset.
- Validation set: The validation set is primarily used to prevent overfitting during the training. In the event of overfitting the network adapts itself too much to the training data and is not able to generalize. So the validation data is used to monitor how accurate the network can recognize other inputs during the training.
- Test set: As the name suggest this dataset is used to test the accuracy of the prediction of the network, once the training is finished

Fig. 14 shows how the whole dataset is split up in this project.

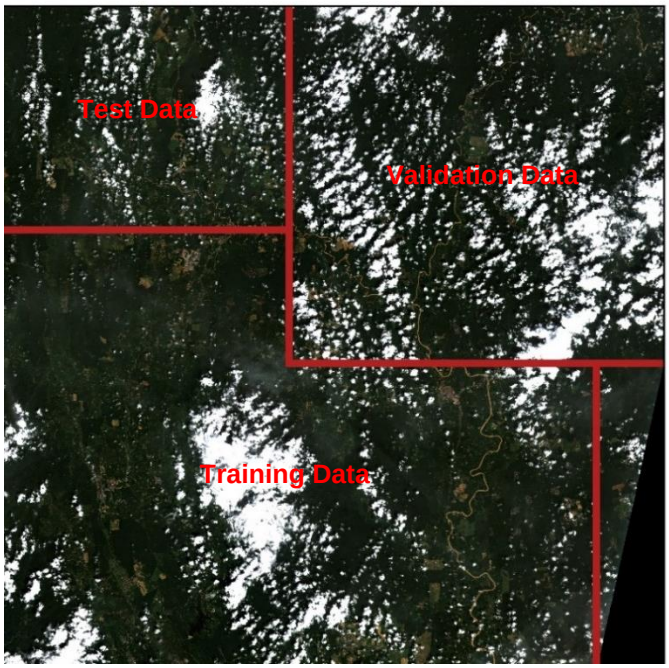

Figure 14: Partitioning of the whole dataset into training, validation and test data

Applying image pre processing

There are multiple ways to preprocess image data before using them in a neural network. The most popular one is called zero mean and is applied in most cases. It combines subtracting the mean and dividing by the standard deviation (Normalization). This is done for every channel separately, though mean and standard deviation are calculated from the two corresponding channels from both points in time. It is also important to note that this is done for all three subsets of the data, but mean and standard deviation are computed using only the training set.

$$channel_{norm} = \frac{channel - mean(channel_{t1}, channel_{t2})}{std(channel_{t1}, channel_{t2})}$$

By subtracting the mean, the data gets centered around the origin of every dimension. Normalization allows to reduce the scale of the data values. Since the used data consists of the unit16 format, as opposed to normal images which are often in uint8, there is a much wider range of values. By dividing through the standard deviation this scale is much smaller, which should lead to more accurate results. This whole process is shown in Fig. 15.

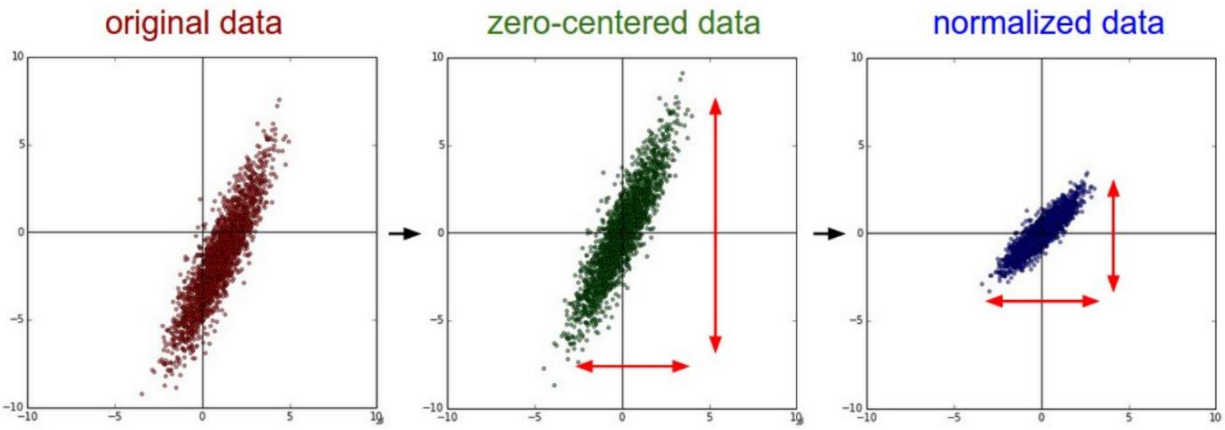

Figure 15: Concept of zero mean pre processing

Creating training and validation patches

Since the network always expects the same input size, as well as computational reasons, all three data sets get cut into smaller patches. In this project a patch size of 32x32 pixels is chosen.

If we analyze the data set we can see that there are two big problems. Firstly there is a lot of cloud coverage and secondly only a very small part of the ground, which is not covered by clouds, is actually labeled change. This amount lies around 1% for all three data sets. These problems result in a very unbalanced data set and could prove not to be enough data for the network to successfully detect changes.

It was tried to compensate this by two steps:

- First one is to create the patches completely random. To do so an random starting point inside the image is generated and the patch is built around it. If such a random patch would be made up by at least 1% of it pixels as labeled change and not more than 5% as covered by clouds, then it would be accepted and saved. This is then repeated until the desired amount of patches is achieved. Additionally random patches containing no labeled changes at all were added. By doing this the amount of labeled change rose from 1% to around 20%. In this way 4000 training patches and 1000 validation patches are being created.
- Second one is to use the concept of data augmentation. Data augmentation gets applied in the case of small datasets and allows to enhance the dataset artificially. Each picture or patch is slightly changed. For a human being it might be almost the same picture but for the network it is something completely different. In this case the changes include randomly flipping or rotating half of the earlier generated patches.

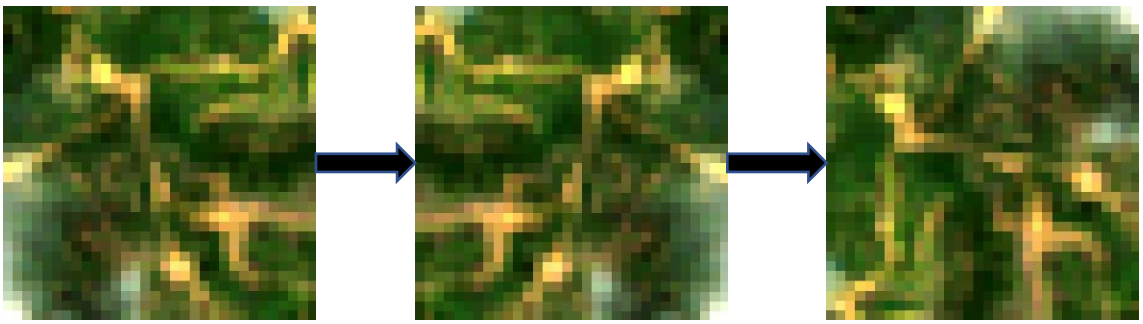

Figure 16: A single patch that is flipped and then rotated by 90 degree

Creating test patches

To create the test patches, the whole test area is cut consistently into smaller parts of width and height containing 32 pixels. Of course patches containing too much cloud coverage would be sorted out and not used as network input. The tolerance is once again set at 5%.

Such a partitioning of the test set leads to the advantage that the predictions of the network for each patch can then be reassembled into an image showing the whole area. Additionally this is much more similar to a real life application. The amount of available patches on the other hand is very low. That is why the tolerance for clouds was set at 5%. With a better dataset it would be of advantage to set it lower.

Fig. 17 shows how the test area is cut up and which patches were ignored due to cloud coverage at either the first or second point in time.

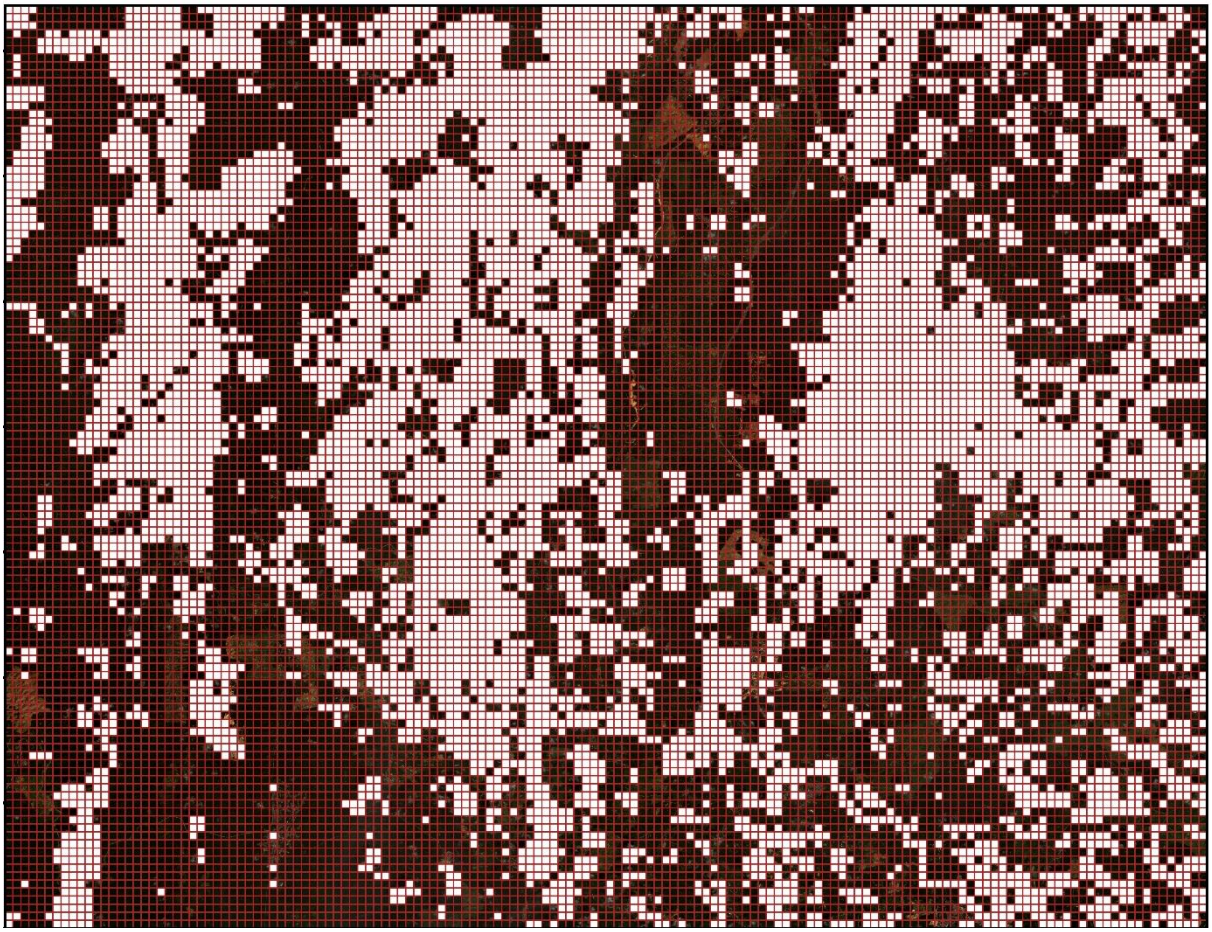

Figure 17: Partitioning of the test area into separate patches

5. Network

5.1. Applied network

As mentioned earlier Convolutional Neural Networks have become the standard Networks for tasks related to image recognition. They allow for large images used as input, without having to determine the huge amount of parameters a normal neural network would have to.

In this project a special kind of Convolutional Neural Network was used: ResNet-50. This stands for a residual network containing 50 layers. The following information and graphs are based on the paper [He et al., Deep Residual Learning for Image Recognition, 2015].

Residual Network tackle the problem that networks experience when adding too many layers. Adding more layers generally increases the nonlinearity of the network, allows for more complex problems to be solved and improves the accuracy of its predictions. Hence where the often used name “Deep Learning” originates. There is however a limit to the benefit of increasing the amount of layers and at a certain point the network’s accuracy stagnates or even gets worse. As described, optimizing the network is done by minimizing the gradient of the loss function. Through higher nonlinearity this gradient gets more complex though and less deeper networks are often able to converge earlier. Additionally by the time the deeper network is converging, a degradation of accuracy can happen. An example of this case is demonstrated in Fig. 18.

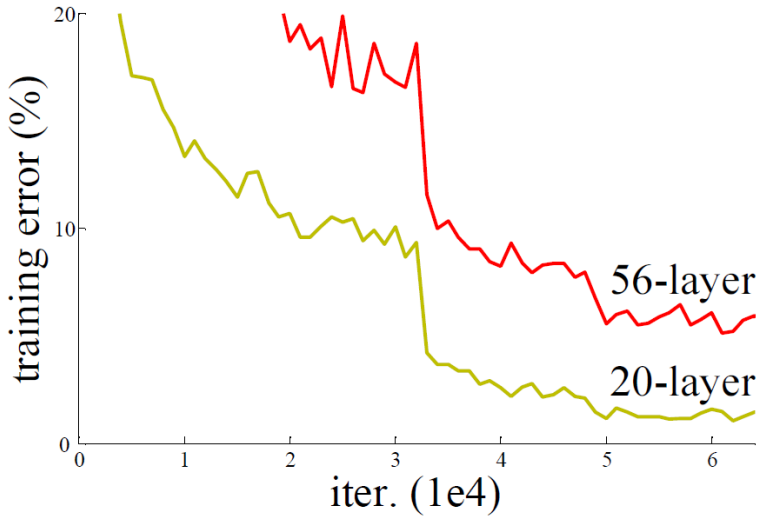

Figure 18: Example of the training errors of two networks with different depths

This problem can be solved with the introduction of Residual Networks. Such networks can achieve this by allowing layers to be shortcut. If we assume that multiple layers can be represented by a function mapping the input to the output, those layers can be represented like in Fig. 19.

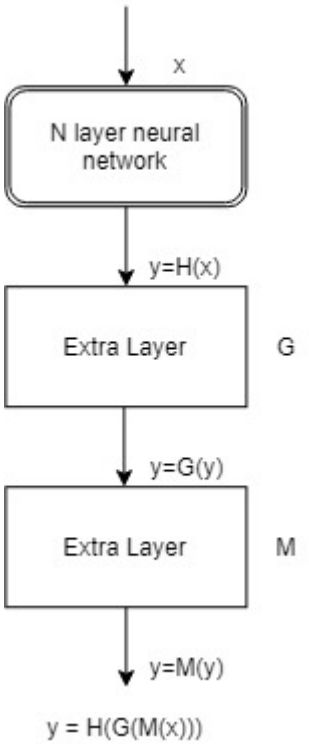

Figure 19: Representation of a mapping function for several layers

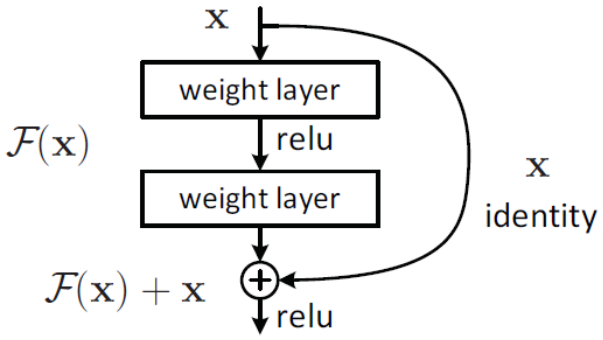

Figure 20: A residual building block

Instead of using the direct function, describing a few stacked nonlinear layers like in Fig. 19 ,a Residual Network introduces a residual function.

$$F(x) = H(x) - x$$

Where x represents the identity function. This leads to a residual block as can be seen in Fig. 20. According to the paper by [He et al.] it is much easier to optimize the residual function than the original unreferenced mapping function. Note that this represents of course not the whole network, but only a small amount of layers. If the dimension of the input equals that of the output, the identity shortcut can be used directly. If not, additional parameters have to be calculated adjusting the change in dimensions.

A complete Resnet usually consists of four stages, each containing several residual blocks. Inside a stage the dimensionality remains the same, so that identity blocks can be used. Furthermore each stage uses the same amount of kernels, but the amount of filters doubles every stage.

By adding a normal convolutional layer at the very beginning and a fully connected layer at the end, the respective amount of layer results. Those architectural principles can be seen for different amounts of layers in Fig. 21.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Figure 21: Architectural principle of different Resnets

5.2. Adaption of the network

The residual network just described is used for image classification. It takes an image as input and delivers a class for the whole image as output. Since the goal of this project is to perform an image segmentation by determining the class for each pixel of the input patch, it had to be adapted slightly.

In the case of image classification the spatial dimension of the input gets continuously reduced until it equals one. As opposed to the case of a segmentation, where the spatial dimension must not change. Thus all layers causing a reduction of dimensionality are either removed or adapted:

- All pooling layers are removed
- The strides in all layers are set to one
- Zero padding is always applied

Additionally the fully connected layer at the end of the network is also removed. This layer normally transform the flattened and pooled last layer of the last stage into a vector containing the probabilities for each class. But since we want a probability for every pixel of our input, the fully connected layer is replaced by a convolutional layer with a filter size of two, representing the two classes change or no change respectively. This whole procedure is summarized by Fig. 22.

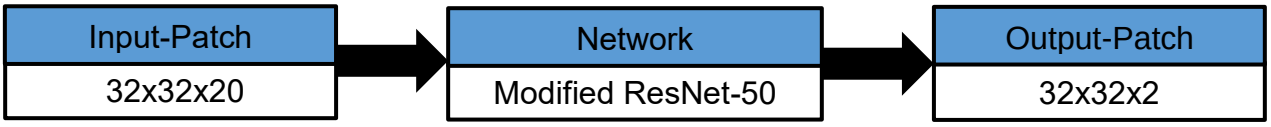

Figure 22: Summary of the whole process

The rest of the network architecture corresponds to the usual structure of ResNet-50 as can be seen in Fig. 21. In total the complete network consists of 23'645'122 parameters.

5.3. Optimization

The loss function applied for the optimization of this network is called categorical cross entropy loss. The total loss can be written as:

$$L = \sum_{i=0}^{k} -\log(pi)$$

Where pi corresponds to the probability of a class. Since the cross entropy loss expects probabilities as input, the ReLu activation function obviously does not work here. It is replaced in the last layer with the softmax activation function, which transforms the input values into the desired range. Additionally the probabilities of all classes sum up to one.

$$pi = F(xi) = \frac{e^{xi}}{\sum_{j=0}^{k} e^{xj}}$$

It divides the exponential power of every class value by sum of all the exponential powers of every class value

The applied optimization algorithm is a variation of the stochastic gradient descent called Adam.

6. Training

For every training session two models are saved.

- The model of the last epoch
- The model with lowest validation loss

No pre trained model is used and the all weights are initialized with a method called he_normal. This method draws samples from a truncated normal distribution centered around 0, with the standard deviation depending on the number of input units [Keras documentation, 2018].

The network was trained multiple times with varying learning rates. Each of those training session resulted in very similar results. The best result was achieved with a learning rate of 0.00001 and a batch size of 25 patches. This training was done over 65 epochs with the lowest validation loss of 0.36 in epoch 57. Fig. 23 shows the development of the loss.

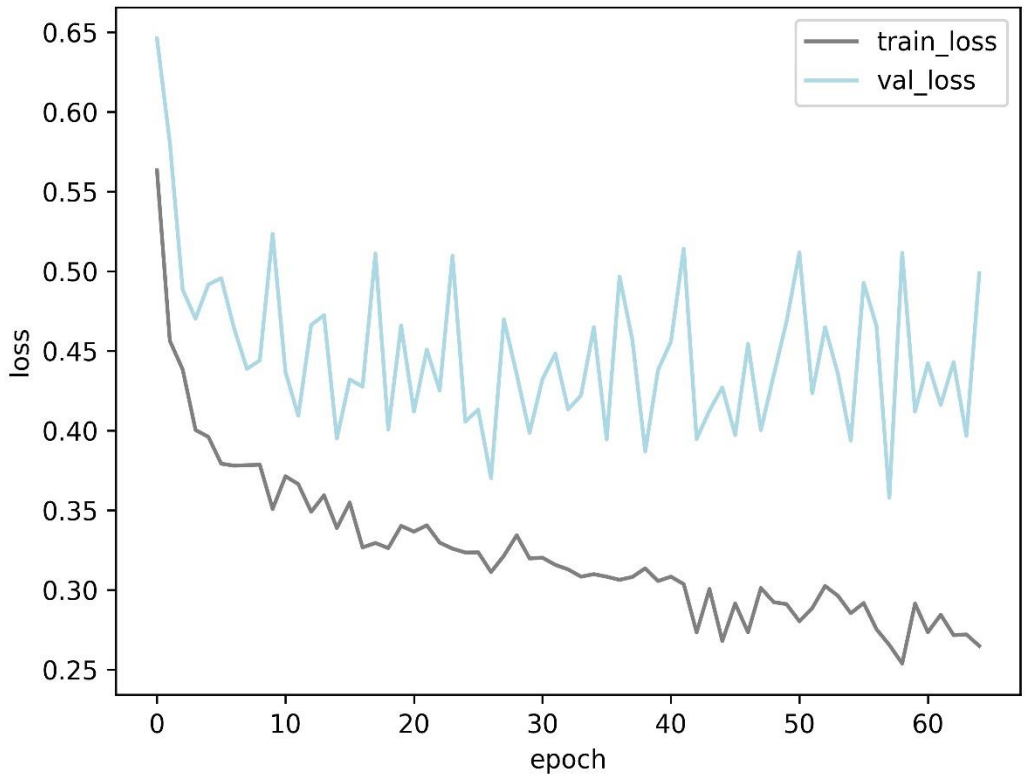

Figure 23: Training and validation loss in every epoch

As expected, both training and validation loss quickly decrease in the beginning. The training loss continues to decrease slowly with smaller exceptions from time to time. But this can be expected, due to the not always necessary improving gradient descent. The trend of the trainings loss curve is surely converging. The trend of the validation loss however, stops decreasing inside the first 10 epochs and starts oscillating between 0.53 and 0.36. These large oscillations always came to pass, regardless of the amounts of epochs in a training session. The curve never started to converge.

The history of the accuracy during the training shows similar properties.

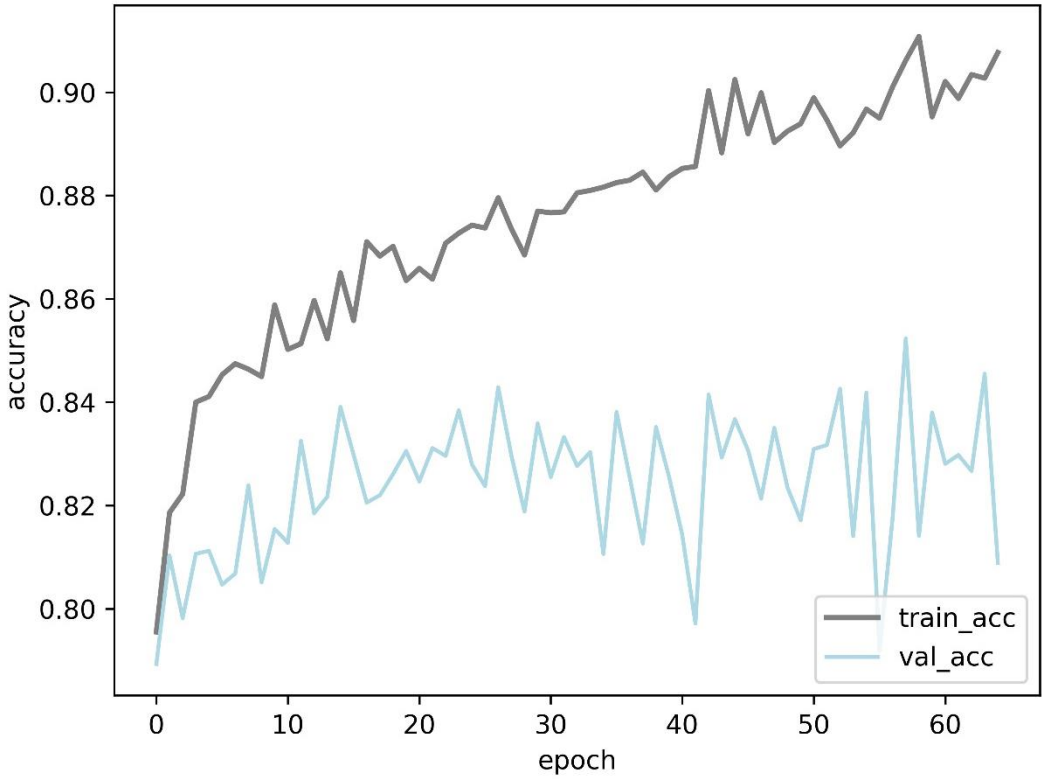

Figure 24: Training and validation accuracy in every epoch

Once again we can see fast improvements in the beginning for both curves and a converging trainings curve whereas the validation accuracy starts oscillating.

For every training that was performed those curves looked alike. And since the best result, although not by a big margin, for both the validation and test accuracy resulted from the training shown above the other training session are not further mentioned here.

7. Evaluation

Once the training of the network is complete, the trained model can be applied to the test patches. This results in a binary map of the patches, containing the two classes change or no change respectively. To provide an adequate visual representation of the whole test area, those patches are then merged once again. The patches that were sorted out before because of too much cloud coverage, are just replaced with a third class cloud. The complete prediction for the test area and the labeled changes can be seen in Fig. 25 and Fig. 26. Change is coloured in red, no change in black and the clouds in white.

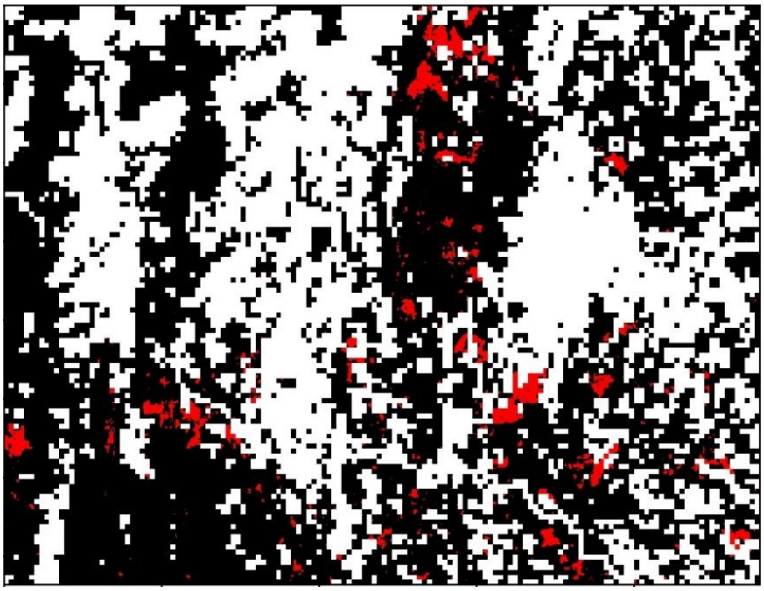

Figure 25: Predicted changes in the test area

Figure 26: Actual labeled changes in the test area

We can see that there is a significant amount of difference between prediction and labels. Every polygon that was labeled also seems to be detected more or less correctly by the network. However the network also predicts much more deforestation that was actually labeled.

This assumption seems to be confirmed by magnifying an area with particularly big differences between prediction and labels which is shown in Fig. 27.

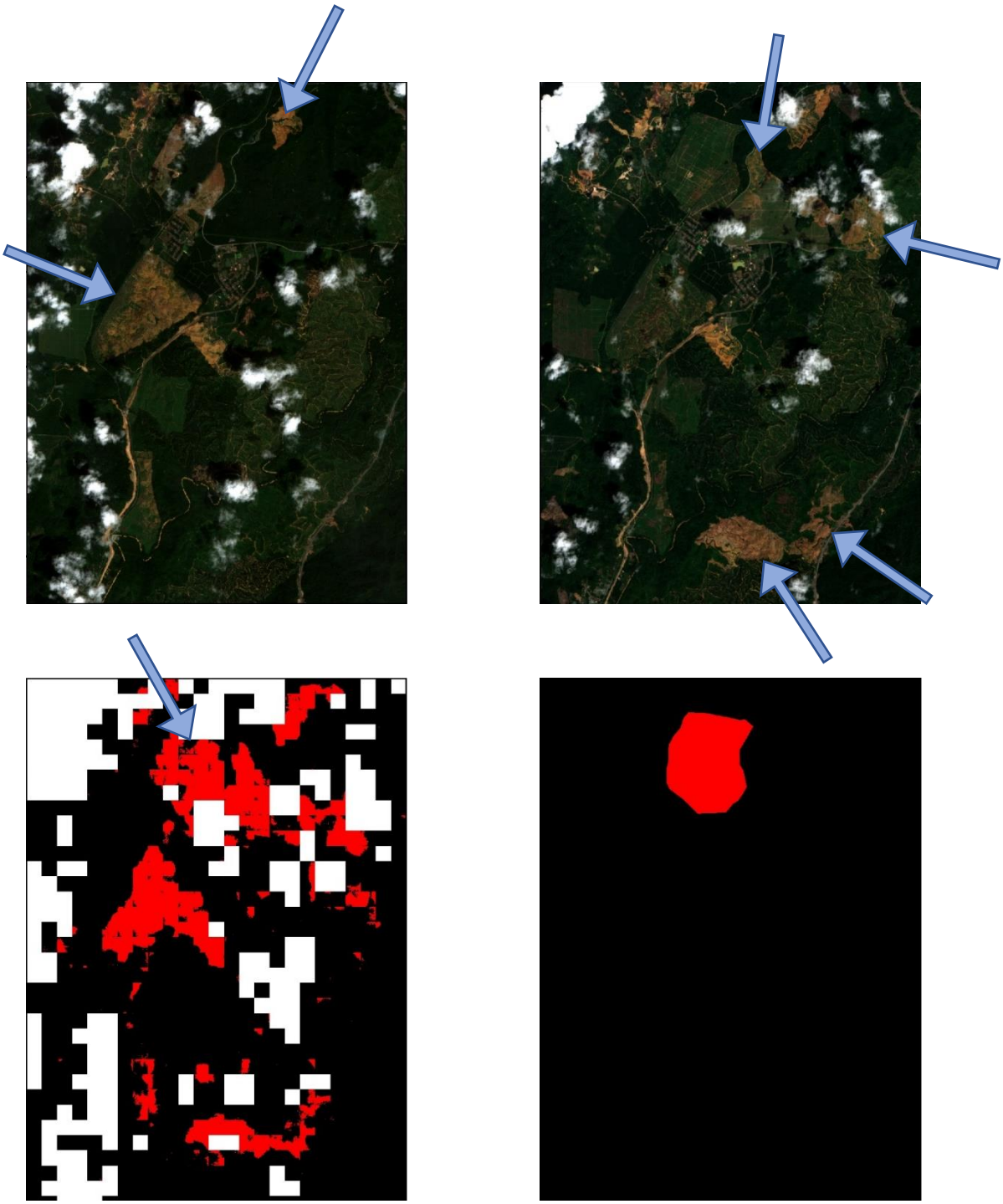

Figure 27: A small part of the test area at both points in time, the prediction and the labels

By analyzing these close ups several properties can be distinguished.

- The network also seems to include all agriculture changes in its prediction. Especially the growth of plantations. Those areas are marked in the first sub image.
- The labels are very incomplete. There is a substantial amount of deforestation occurring which is not included in the labels. But luckily all of those areas are predicted by the network anyhow. These are marked in the second sub image.
- The only polygon modeled in the labels is also included in the prediction. Although the nearby clouds seem to have affected the shape. That polygon is marked in the third sub image.

These observations are also confirmed by the confusion matrix of the whole test area and the therefrom derived values

Predictions \ Labels	No Change	Change
No Change	9'287'911	29'310
Change	322'819	101'272

$$Overall\ Accuracy = 96.39\%$$

$$Recall = 77.55\%$$

$$Precision = 23.88\%$$

$$F1 - Score = 36.52\%$$

As already addressed there occurred significantly more deforestation then included in the labels. Additionally did the network also include agriculture changes in its prediction. This leads to a very high rate of false positives in the statistic and a very low precision score. But since the labels are very incomplete this is not necessarily a negative outcome.

The deforestation, which is actually labeled, on the other hand is predicted quite reliable. 77.55 % of all labeled pixels are actually included in the prediction. If we go by the only polygon contained in the close up, it is clear that the shape of the prediction was influenced by nearby clouds or marginal cloud coverage inside the test patches. This would lead to the idea, that the recall rate could be even better by lowering the amount of clouds allowed in the test patches. But since the amount of suitable test patches would have become very small, this is not executed here.

8. Conclusion

The network has been successfully trained to detect changes in native rainforests. It seems that all the deforestation which can be identified on the aerial images is included in the prediction of the network. The deforestation which was actually labeled is also predicted at a decent percentage of about 78%. It is very likely that this rate would be even higher if the tolerance for clouds in the test patches would be lower, since it seems that small remaining cloud coverage can easily confuse the network.

Unfortunately however, does the prediction also include changes in agriculture. Especially already deforested areas in December 2015, which have been planted with trees during the following year. Indeed is the growing of plantations heavily linked with deforestation but these detections are not intended.

The most obvious solution to most of these problems would be to complete and refine the labels, so that they clearly show the difference between deforestation and planting. It could be expected that the network used in this project would then easily be able to differentiate. Additionally the fluctuation of the validation loss during the training could possible also be solved by this. If that would not be the case, the labels could be expanded by a third class. This third class would indicate the change in agriculture. It should however be mentioned, that manual labeling is not always so easy and that errors in the labels easily can occur.

A further possible way to make the predictions more robust would be simply to increase the amount of data available during training. Although the used area is quite large, the amount not covered by any clouds is not. Though the network was still able to detect deforestation, a more balanced data set could never be of any disadvantage.

It can also be concluded that sole use of optical images is probably not enough to reliably check large areas for deforestation. The exact points in time would have to be chosen carefully, so that there would be next to no cloud coverage over a whole area. Not only are such moments unlikely to occur, but for an automated system which is supposed to check an area in frequent intervals, this would not be feasible in practice.

9. Outlook

As already addressed, the use of only optical data to detect any change between two or more points in time is probably not useful in real world applications. Therefore a combination with radar data is proposed. The nature of radar allows the penetration of any form of cloud coverage and thus every dependence on cloud free areas disappears. Furthermore could the use of additional data make the predictions in cloud free areas even more robust and reliable. Of course the whole data preparation workflow and the architecture of the network would have to be adapted.

The employment of radar data could also introduce other interesting aspects, as for example the use of polarimetry, which could open up completely new fields of use.

It could also be conceivable and interesting to apply the used network in other areas. In principle we could detect any changes with the implemented network. This could include the monitoring of flooded areas, glaciers or sustainable urban development. The only constraints would be correct labels and detectable differences in the optical channels.

The areas for deep learning applications with satellite data seem countless and surely will only become more important in the future, as there will be an increased demand for automated monitoring of our planet.

10. Acknowledgment

I would like to thank the following people.

- Nico Lang, for the numerous inputs and the continuous help during the project.
- Andrés Rodríguez, for helping with the processing of the raw satellite data and providing many other helpful tips.
- Dr. Jan Dirk Wegner, for his input and expertise during the weekly meetings.
- Prof. Dr. Konrad Schindler, for making this project possible.
- Sho Koizumi, for providing the labeled changes used in this project.

11. References

Deep Residual Learning for Image Recognition [He et al. 2015]

Keras Documentation [Keras, 2018]

https://keras.io/

CS231n: Convolutional Neural Networks for Visual Recognition [Stanford, 2018]

http://cs231n.github.io/

Deforestation [WWF, 2018]

https://www.worldwildlife.org/threats/deforestation

Sentinel-2 Mission [ESA, 2018]

https://sentinel.esa.int/web/sentinel/missions/sentinel-2

Sentinel Application Platform [ESA, 2018]

http://step.esa.int/main/toolboxes/snap/

Sentinel Hub [ESA, 2018]

https://www.sentinel-hub.com/

80% of Malaysian Borneo's rainforests destroyed by logging [climatechangenews, 2013]

http://www.climatechangenews.com/2013/07/18/80-of-malaysian-borneos-rainforests-destroyed-by-logging/, 2013