Final Project - Advanced Machine Learning

Satellite image segmentation using U-net neural networks

Petrus Salminen, Daniel Richter

https://github.com/petsal96/aml-fp-repo

https://drive.google.com/drive/folders/1lo–dPFTSVnS9NlU5bmgakanvdERZgTD?usp=sharing

A segmentation of a satellite image our model is able to generate.

Abstract

In this report we use the U-Net-architecture semantic segmentation of satellite images. We present the different data preparation steps needed for our project, the different models we implemented, and the results we obtained. We show good performance of U-Net in satellite image segmentation with four different classes.

Contents

1 Introduction

Author: Petrus Salminen

Image segmentation is the process of categorizing an image on the pixel level into multiple regions. These regions result in a simplification of the original image which can then be used for further analysis. Typical examples for image segmentation problems are:

- Recognizing streets, buildings, cars, pedestrians etc. in real-time on videos. Such methods are utilized especially in the fields of robotics and autonomous driving.
- Finding abnormal formations on medical images. The resulting segmentations can help doctors to find outbreaking diseases or other severe anomalies earlier and easier than by visually interpreting the images.
- Creating regions to represent fields, forests, buildings etc. on satellite images. This is especially useful in analyzing scenery changes or automatic landscape generation (e.g. for video games or scientific simulations).

There are three different types of image segmentation:

- Semantic segmentation: Categorize every pixel without differentiation of distinct instances of categories (e.g. background, road, person).
- Instance segmentation: Categorize only relevant pixels with differentiation of distinct instances categories (e.g. person1, person2, car1, car2).
- Panoptic segmentation: Categorize every pixel with differentiation of distinct instances. This is basically a combination of semantic and instance segmentation.

One of the classical deep learning approaches to this problem is the U-Net-architecture first introduced in Ronneberger et al. [1]. For the final project we will also use this approach for the segmentation of satellite images.
In this report we will first present the data structure and sources. In the next section we will go over the exact implementation and evaluation of our U-Net. In the section chapter we will discuss the results of the training with different variations of parameters. In the last section we will describe extensions of the U-Net architecture and discuss more sophisticated methods which are currently used in the industry.

2 Background and Theory

2.1 Data preparation

Author: Petrus Salminen

The data sets described in this subsection can be found on Google Drive.

2.1.1 Initial data set

We will now describe the steps involved in creating the data sets for training, validation and testing. We chose to use a region around Frankfurt am Main with coordinates 50.35695, 8.39033 (top-left corner) and 49.88891, 8.98149 (bottom-right corner) a total area of 113.873 km^2. The satellite image was downloaded from Bing Maps (zoom 18) using SAS.Planet 211230.10225 (see 2.1).

Figure 2.1: Satellite image used for the data sets.

For the labels of the pixels we decided to use four categories:

- FOREST: Forests and other dense formations of trees.

- FIELDS: Areas with low or no vegetation.
- URBAN: Urban areas including roads.
- WATER: Rivers and lakes.

The process of categorization was done using QGIS 3.26.1. The corresponding shapefiles and raster data were downloaded from the following sources:

- `https://www.mundialis.de/en/deutschland-2020-landbedeckung-auf-basis-von-sentinel-2-daten/`
- `http://download.geofabrik.de/europe/germany.html`

The resulting raster file was saved in GeoTIFF-format for further processing (see 2.2). We used gdal 3.4.1 to split the images with a train-validation-test-split of 70%-15%-15%.

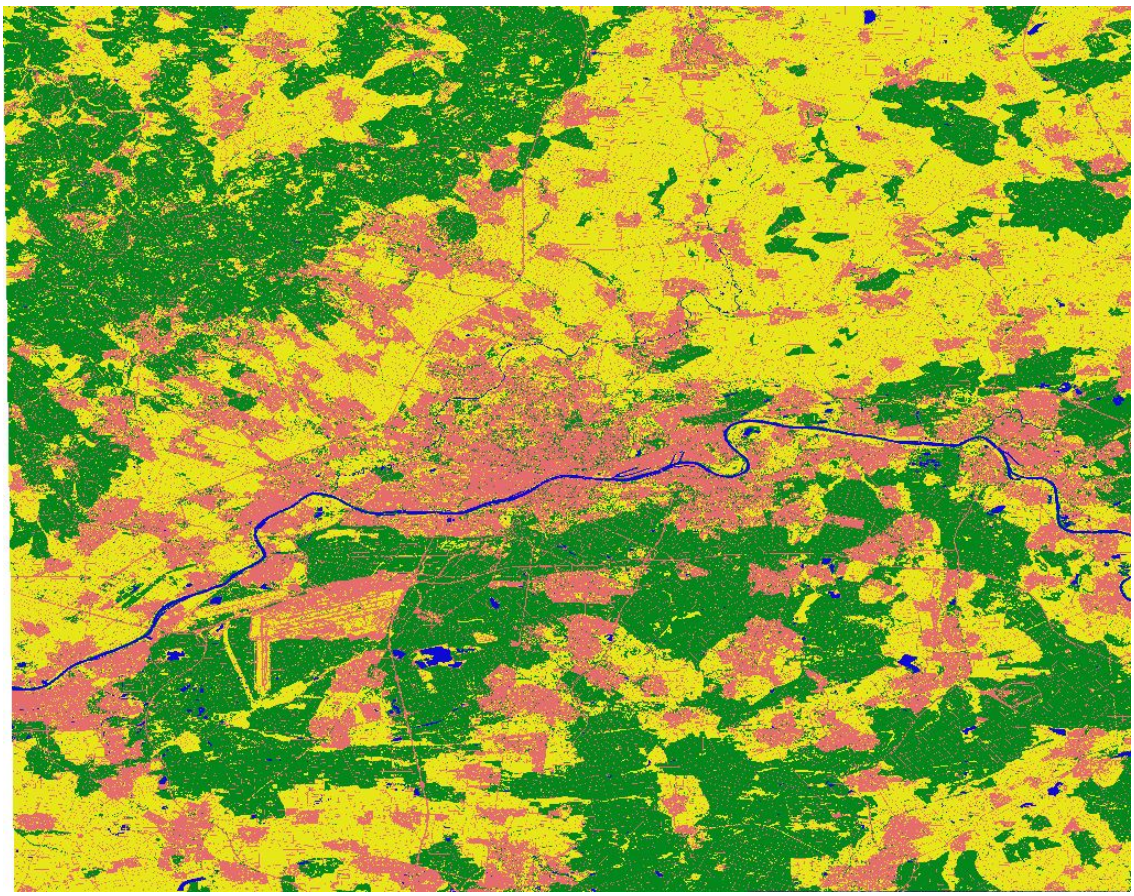

Figure 2.2: Categorized satellite image.

2.1.2 Optimization of the initial data set

There were several problems with the initial data set. Using QGIS, we analyzed the areas of the different classes (see 2.3). The data was highly imbalanced with the largest class "FIELDS"

taking 43.01% of the total area and the smallest class "WATER" only 3.99%. Furthermore, there were many unpaved roads that were indistinguishable from fields and roads covered by trees.

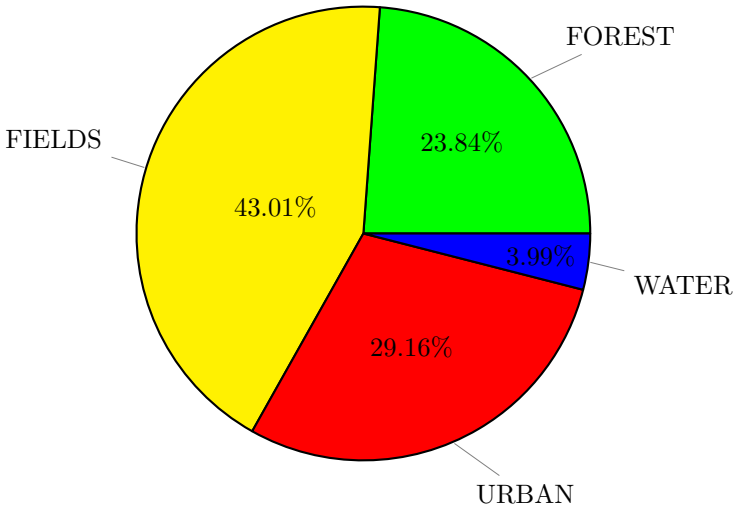

Figure 2.3: Distribution of labels

We knew that the first data set was not perfect but decided to go on due to time limitations. However, the training process went so smoothly that there was still time left to try to create better data. We found a more accurate land cover data set for the labels. The corresponding shapefiles and raster data were downloaded from the following sources:

- `https://hub.arcgis.com/datasets/44fe5cb0b33a4b9c93afe9aec65c196b_4`
- `http://download.geofabrik.de/europe/germany.html`

It turned out that the class imbalances were actually not a problem. Especially the sparse class "WATER" was predicted accurately. However, as we had used a small resolution for the raster-files, the roads were recognized in squares (see 4.2). There was also no differentiation between wide and narrow roads in the first training set. We addressed this issue (and the issue with hardly visible roads) by downloading only shapefiles with specific tags from OpenStreetMap. We splitted the shapefiles in the following sets:

- highway=motorway, highway=motorway_link, highway=trunk, highway=trunk_link and highway=primary (widest, e.g. "Autobahn", "Schnellstraße", "Bundesstraße")
- highway=primary_link, highway=secondary (e.g. "Landesstraße")
- highway=secondary_link, highway=tertiary, highway=tertiary_link, highway=unclassified, highway=residential and highway=living_street (e.g. "Kreisstraße")
- highway=tack with tracktype=grade1 and tracktype=grade2 (narrowest, e.g. "Wirtschaftsweg")

We used a different width for each set so that the roads were categorized accurately. We also used a very high resolution for the output file. Finally, we also downloaded the "water"- and "waterways"-shapefiles from OpenStreetMap and used them instead of the categorization

provided by the land cover map as they had more accurate borders. As can be seen in 2.4, the new annotations are clearly more accurate and smoother.

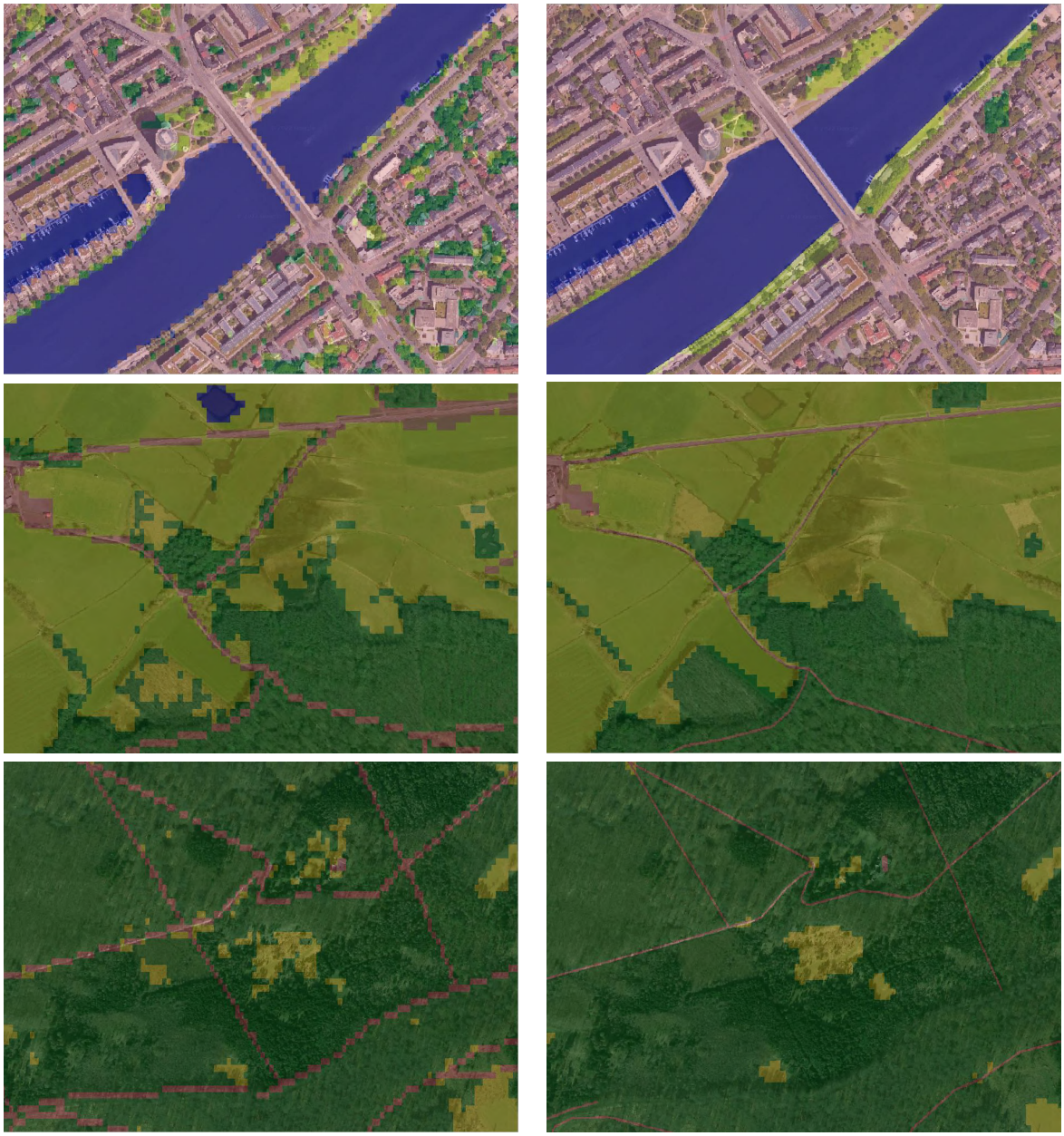

Figure 2.4: Comparison of old and new annotations (left: old, right: new)

We created following data sets from the files described above:

- Bing satellite images with old land-cover annotations and new road- and water-annotations
- Bing satellite images with new land-cover annotations and new road- and water-annotations

- Google satellite images with new land-cover annotations and new road- and water-annotations

2.2 Image Segmentation

Author: Daniel Richter

There are many different techniques for segmenting images, mainly divided into classical and AI-based methods. The following paragraphs give a short introduction into some commonly used segmentation methods.

2.2.1 Classical Methods

Edge based segmentation

By identifying edges of different objects in a given image a simple segmentation method is given. Edge detection strip images of redundant information and therefore reduces their size and facilitate further analysis. Based on contrast, color and saturation edges are identified by specific algorithms.

Threshold based segmentation

Dividing pixels based on their intensity relative to a given value or threshold is arguably the simplest form of segmentation. This concept is suitable for segmenting objects with high intensities or filtering out backgrounds.
Usually a constant threshold value T is used, but in some cases a dynamic threshold can be used as well. Thresholding a grayscale image into two segments according to their relationship to T results in a binary image.

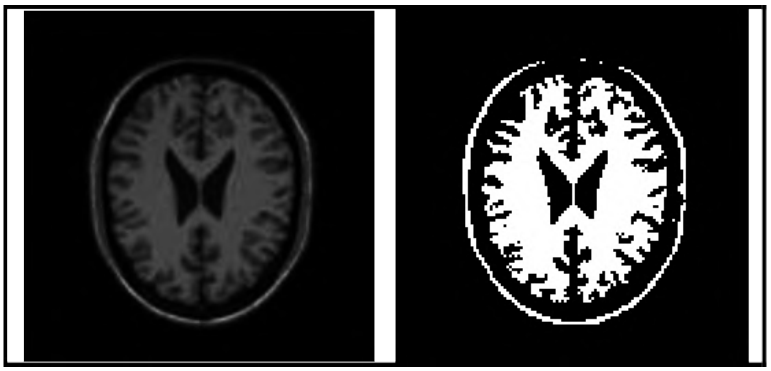

Figure 2.5: Thresholding in medical image segmentation [2] to generate binary segmentation maps

Region based segmentation

A more complex segmentation method used for instance segmentation are region based models [3]. By dividing images into regions with similar characteristics, distinct groups of pixels can be formed segmenting the image. Usually a variety of simpler methods is used as well e.g. thresholding to find similar regions.

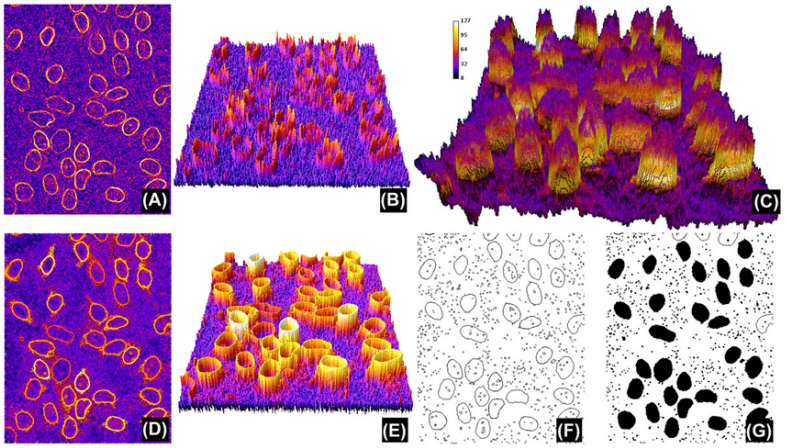

Figure 2.6: A depiction of different steps of a region and threshold based segmenting method of [3]. After amplifying the measurements, thresholds are applied. The enclosed areas are then segmented using a gradient based algorithm.

2.2.2 Machine Learning Methods

There are several different approaches to image segmentation using machine learning techniques. Here we present two variations which lead to the development of the U-net, the model we used for our final project.

Fully Convolutional Network

One of the first deep learning segmentation algorithms developed is the so called *Fully Convolutional Network*. It works by training a neural network that takes a patch of the image around a pixel and predicting each pixel class individually [4].

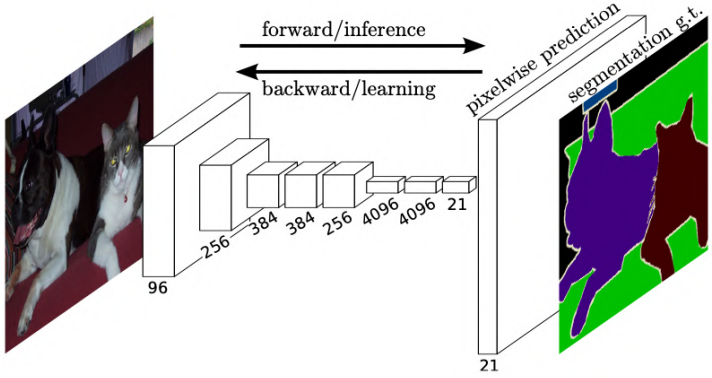

Figure 2.7: A depiction of a Fully Convolutional Network as shown by [4].

This approach has two major downsides; firstly it is very slow, since every pixel has to be evaluated individually (see Figure 2.7). Secondly a trade-off between large and small patch sizes has to be chosen. A too large patch may decrease accuracy and increase training and evaluation

times, while a too small patch lowers the acquired context reducing accuracy as well.

Convolutional and Deconvolutional Networks

This approach proposed by [5] combines a convolutional network with a deconvolutional network to generate a pixel-wise segmentation map. First a convolutional network processes an image to generate a dense vector of features. In the second part the deconvolutional network uses unpooling to extend the vector information into the segmentation map (see Figure 2.8). Compared the

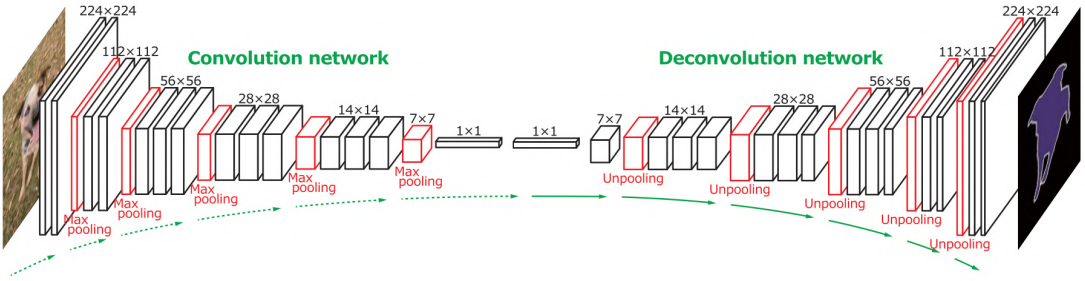

Figure 2.8: A deconvolutional network proposed by Hyeonwoo Noh et al.[5] for semantic segmentation. Based on the VGG 16-layer net [6] a deconvolutional network is added to generate an accurate segmentation map for a given input.

fully convolutional network, this network does not require pixel by pixel evaluation, reducing the needed training set size and training time. Due to the dense compression the network is limited in performance. This results in miss-labeling and ignoring smaller objects by regarding them as background [5].

U-net

The final model we chose incorporates features of both networks, and is called *U-net*. It is built similar to the deconvolutional network, using UP-convolutional layers instead of unpooling and skip connections to improve learning. The compression size in the middle of the network is increased, resulting in better accuracy. A detailed introduction can be found in the following section (3.1).

3 Methods

Author: Daniel Richter

3.1 U-Networks and backbone variations

Deep convolutional neural networks have demonstrated good performance in a variety of visual recognition tasks, like object detection [7] and image classification [8]. These tasks require the input image to have an assigned class label. For projects like semantic satellite image segmentation, however, it is necessary to include precise localization, meaning each pixel is assigned an individual class label.
We build upon Ronneberger et al. (2015) [1] using a fully convolutional neural network named *U-net* (see 3.1) to segment satellite images. By incorporating different variations for the encoder blocks of the network, also called backbone, we try to improve the performance even further. Three different backbones based on the general U-net architecture were implemented. The first backbone is a classical U-net constructed similar to Figure 3.1 but smaller in total dimensions due to time and computational constraints. The two larger, second and third backbone variations are a network with additional residual connections called ResNet and an architecture with inception layers which we call Inception V1.

3.1.1 U-net

The *U-net* expands on the convolutional and deconvolutional approach from [5]. By replacing uppooling layers with Up-convolutional layers and including additional skip connections a more stable and better performing network was found. Each "level´´ can be seen as a distinct block,

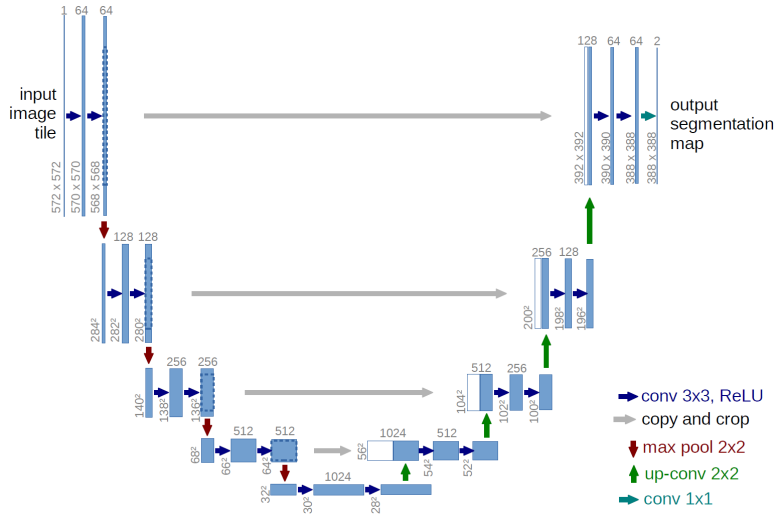

Figure 3.1: The general architecture of a U-network as proposed by Ronneberger et al. [1]. The network has an encoding and decoding structure, connected by various skip connections between those encoding and decoding blocks.

which is enclosed by a downsampling or upsampling layer. In each of those blocks several cascading convolutional layers interpret and process the input. By adding a downsampling layer between each of the blocks, the receptive field of each cascading block is increased, to

further accelerate the information processing. To facilitate the UP-convolutional process it can be assumed, that some parts of the segmentation map are resembled in the input image. By adding skip connections between the encoding and decoding blocks, the reconstruction for the segmentation map can be made easier [5].
For our model we chose an input size of $(128 \times 128 \times 3)$ which is processed into a much smaller tensor with dimensions of $(8 \times 8 \times 1024)$ and then upconvoluted back to the original size of $(128 \times 128 \times 4)$, where 4 represents the number of classes we considered. The exact architecture used can be found in the appendix in Table A.1. In total this model had approximately 31 million trainable parameters.

3.1.2 ResNet18

A residual neural network incorporates shortcut connections mapping the identity function across every few stacked layers.

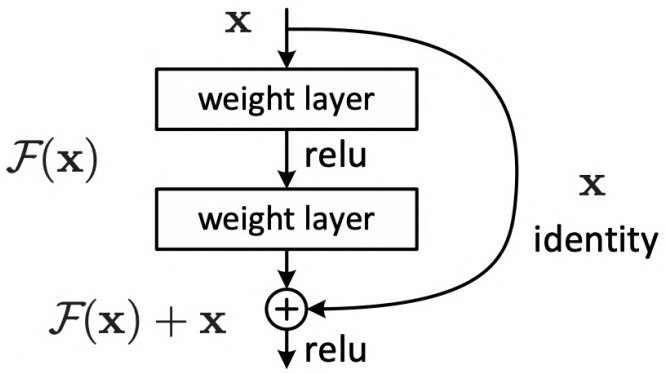

Figure 3.2: The structure of a residual block. Adding shortcut connections results in decreasing training time and increasing performance for very large networks [9].

This approach is called residual learning, mathematically a residual building block is defined as [9]:

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}, \{W_i\}) + \boldsymbol{x} \tag{3.1}$$

Here $\boldsymbol{x}$ and $\boldsymbol{y}$ are input and output tensors of the layers considered. The function $\mathcal{F}(\boldsymbol{x}, \{W_i\})$ is representative of the residual mapping along the shortcutted layers to be learned.
In our ResNet model, based on the structure of ResNet18, we increased the number of convolutional layers in each encoding block, totaling in $4 - 5$ convolutional layers per block, and added shortcut residual connections skipping every few layers. This results in a much deeper model with 42 hidden layers and a little bit less than 57 million trainable parameters. The exact configuration can be found on the github repository (see https://github.com/petsal96/aml-fp-repo).

3.1.3 Inception V1

The third backbone variation we inspected is the so called inception module. Proposed 2014 by Christian Szegedy et al. [10], the idea is to enable deeper convolutional layers by using parallel convolutions. The authors have shown an improved utilization of the computational resources

inside the networks and set a new state of the art record in the ImageNet Large-Scale Visual Recognition Challenge 2014. The inception structure we used is depicted in Figure **??**:

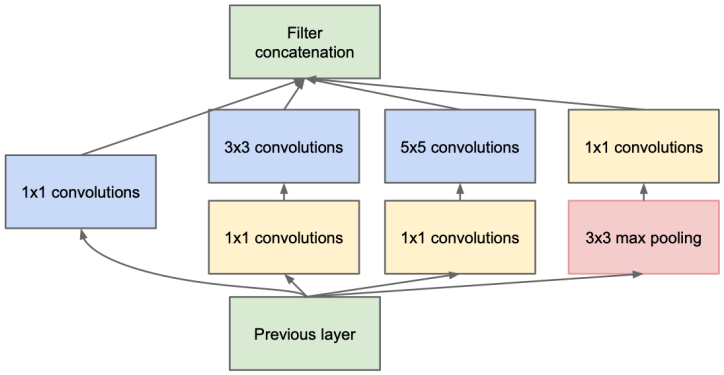

Figure 3.3: The structure of a inception module with dimension reduction as proposed by Christian Szegedy et al. [10].

We used multiple inception blocks inside the the 3rd to 5th encoder blocks, resulting in a very deep inception network with over 60 parallel convolutional layers and over 48 million parameters. The exact structure can again be found in the github repository .

3.2 Loss functions

There is a variety of different loss functions optimized for image segmentation [11, 12]. We conduct training with two different functions, the Categorical Focal and Jaccard losses. This section gives a short introduction into these two functions.

3.2.1 Categorical Focal Loss

The Categorical Focal Loss is designed to reduce the impact of highly imbalanced training data. Rare classes tend to increase the classification error, which encourages the network to pay more attention to those examples. By preventing large numbers of false negatives to saturate the network during training a better performance can be achieved. Mathematically it is defined in dependency of the ground truth y and prediction $\hat{y}$ as:

$$\mathcal{L}_{CFL}(y_i, \hat{y}_i) = -\alpha \cdot y_i \cdot (1 - \hat{y}_i)^{\gamma} \cdot \log(\hat{y}_i) \tag{3.2}$$

where α and γ represent hyperparameters of the loss function. The total loss is then calculated by taking the mean loss of each training batch and pixel.

3.2.2 Jaccard Loss

The Jaccard index, introduced by (Jaccard, 1901) measures the similarity between two finite sample sets. In set theory it is defined as the Intersection over the Union (IoU) of two sets A and B:

$$\frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.3}$$

For minimization tasks the Jaccard distance should be used, which is defined as:

$$JD = 1 - \frac{|A \cap B|}{|A \cup B|} = 1 - \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.4}$$

In machine learning we want to obtain the similarity of predictions to ground truth values, the Jaccard loss is therefore defined as:

$$\mathcal{L}_{JL}(y_i, \hat{y}_i) = 1 - \frac{y_i \cdot \hat{y}_i + \epsilon}{y_i + \hat{y}_i - y_i \cdot \hat{y}_i + \epsilon} \tag{3.5}$$

Here ϵ denotes a smoothing factor so that dividing by zero becomes impossible.

3.3 Evaluation metrics

To evaluate our trained models we need metrics that quantify the goodness of predictions on unseen test data. For image segmentation with different classes two metrics are commonly used, the Categorical Accuracy and the Jaccard Coefficient.

3.3.1 Categorical Accuracy

Categorical Accuracy calculates how often predictions match one-hot labels. The best score is 1 for predicting all classes for all samples correctly, while the worst score is 0 if all predictions are false.

3.3.2 Jaccard Coefficient

The Jaccard Coefficient, often referred to as Jaccard Index predeceases the Jaccard Loss and is mathematically defined as:

$$JC(y_i, \hat{y}_i) = \frac{y_i \cdot \hat{y}_i + \epsilon}{y_i + \hat{y}_i - y_i \cdot \hat{y}_i + \epsilon} \tag{3.6}$$

where ϵ refers to the same smoothing parameter as in 3.2.2.

4 Experiments and Results

Author: Petrus Salminen

Throughout this section we will use the term "best model". This refers to the best performing model within a specific training in respect of the corresponding accuracy measure.

4.1 Comparison of different backbones

The first task was to identify the best performing backbone in the U-Net architecture. To speed things up, we trained the models on a random subset of 6400 training images and using an early stopping rule, if the validation loss didn't improve within five epochs. For each of the three backbones, we trained models with the following setups:

- batch sizes 32 and 64
- learning rates 0.001, 0.0001 and 0.00001
- dropout rates 0.0, 0.1 and 0.3
- categorical focal loss and Jaccard-loss

In each training, we saved the best performing models in terms of categorical accuracy and Jaccard-coefficient. There were 108 trainings in total.
We made following observations when comparing the different setups:

- there were only small differences in models trained with different batch sizes
- a learning rate of 0.001 was in general better in terms of categorical accuracy
- a learning rate of 0.0001 was in general better in terms of the Jaccard-coefficient
- using a dropout rate led in general to worse results
- models trained with categorical focal loss performed very poorly in terms of the Jaccard-coefficient
- models trained with Jaccard-loss performed almost as good as with the categorical focal loss in terms of categorical accuracy

The best setups for the classical U-Net-backbone, the ResNet18-backbone and the Inception V1-backbone can be taken from 4.1.

backbone	batch size	learning rate	dropout rate	loss	cat. accuracy	Jaccard-coeff.
U-Net	64	0.001	0.0	cat. focal loss	0.8628	0.4355
	64	0.0001	0.0	Jaccard-loss	0.8576	0.6109
ResNet18	64	0.001	0.0	cat. focal loss	0.8580	0.4202
	64	0.0001	0.0	Jaccard-loss	0.8517	0.6103
Inception V1	64	0.001	0.0	cat. focal loss	0.8588	0.4291
	64	0.0001	0.0	Jaccard-loss	0.8518	0.6035

Table 4.1: Best validation accuracies of models trained with different backbones.

The ResNet18- and Inception V1-backbones performed equally well but the classical U-Net-backbone overperformed both. Thus, we concluded that the classical U-Net-backbone is most suitable out of these three options for the segmentation of our satellite images. In the following we focused on improving the classical U-Net.

4.2 Interim results

The next step was to test if training on larger images would give better results due to more context. However, we didn't change the dimensions of the input layer in the network but resized the larger images to 128 px. We compared image sizes 128 px and 256 px (see 4.2). In the training we used early stopping with ten epochs.

image size	cat. focal	Jaccard
128 px	0.8798	0.8756
256 px	0.6996	0.7253

Table 4.2: Best validation accuracies of models trained with different image sizes (accuracies calculated corresponding to the loss).

There seemed to be no significant differences in terms of categorical accuracy but the model trained on larger images was more accurate in terms of the Jaccard-coefficient. We decided to use the smaller images for categorical focal loss and the larger images for the Jaccard-loss in the last steps of the otimization.

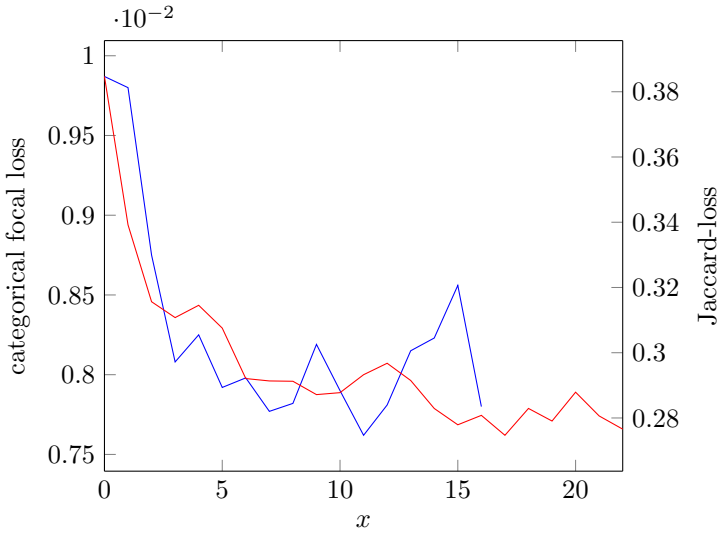

Figure 4.1: Graph of learning rates (y-axis) over epochs (x-axis) for the models trained with categorical focal loss (blue) and Jaccard-loss (red)

Up until now we had used a static learning rate throughout the trainings and decided to use the following learning rate scheduler to get the best model:

$$lr_{e+1} = \begin{cases} lr_e, \text{ if } e < e^* \text{ or } e \bmod \delta \not\equiv 0 \\ \gamma \cdot lr_e, \text{ else} \end{cases}$$

where e indicates the epoch, lr_e the learning rate at epoch e, e^* the start of decay, δ the number of epochs between decays and γ the decay rate. To save some time, we fixed the decay starts to the epochs at which the learning rates started to stagnate (12 for categorical focal loss and 17 for Jaccard-loss, see 4.1) and decided to optimize only the decay rates over the set $\{0.99, 0.95, 0.9, 0.75, 0.5\}$ and the steps between learning rate decay over the set $\{1, 3, 6\}$.

image size	δ	γ	loss	acc. measure	accuracy
128 px	1	0.75	cat. focal	cat. accuracy	0.8808
256 px	1	0.5	Jaccard	Jaccard-coeff.	0.6506

Table 4.3: Best validation accuracies of models trained with different hyperparameters.

Finally, we calculated the overall accuracies of the trained models on the test set (see 4.3). At this point we were very satisfied with the results. Guérin et al. [13] had reported a Jaccard-coefficient of 0.5422 which was significantly smaller than our best accuracy. Furthermore, we used a randomly selected area in Hessen (which was not included in the data sets used for training and evaluation) to visualize the results (see 4.2).

Figure 4.2: Predictions from the best models (left: model trained with categorical focal loss, right: model trained with Jaccard-loss).

When compared visually against each other, there seemed to be only small differences which indicates that both, the categorical focal loss and the Jaccard-loss, are suitable for training an

U-Net for this task. Although the results looked very good already, we had still time left and continued with the optimized data sets described above.

4.3 Models trained with new land cover data

Figure 4.3: Predictions from the best models (left: models trained with categorical focal loss, right: models trained with Jaccard-loss, top: models trained with Google satellite images, bottom: models trained with Bing satellite images).

In this subsection we will shortly describe the results from trainings with the data sets containing the new land cover data described in Optimization of the initial data set. As the categorization seemed to be very accurate, we started with these trainings and used the tuned hyperparameters from Interim results.
When analyzed visually, the results were not great (see 4.3). All models had problems in detecting forests and/or waters. Although all models predicted urban areas better than the previous models, we didn't see this as a significant improvement and skipped further evaluations to save some time.

4.4 Final models

By now we assumed that the new land cover data wasn't as accurate as we had thought. We trained the final models with the old land cover data with corrected roads and waters and Bing satellite images as described in Optimization of the initial data set. As can be seen in 4.4, the resulting test set accuracies were significantly better than before. Also compared to the results obtained by Guérin et al. [13] these are a huge improvement although one should remember that the class definitions were also slightly different.

image size	δ	γ	loss	acc. measure	accuracy
128 px	1	0.75	cat. focal	cat. accuracy	0.9121
256 px	1	0.5	Jaccard	Jaccard-coeff.	0.7609

Table 4.4: Best validation accuracies of models trained with different hyperparameters.

The resulting prediction were even more impressing than the numbers (see 4.4). Although the road detection could still be improved, the edges between different categories are very smooth and even small details like single buildings in midst of fields and forests are predicted mostly correctly. When looking at these images, it seemed that the model trained with categorical focal loss was slightly better in recognizing narrower roads. As these are hardly visible on the small images, we uploaded them also to the GitHub-repository. The trained model weights can be downloaded from Google Drive.

Figure 4.4: Predictions from the best models (left: models trained with categorical focal loss, right: models trained with Jaccard-loss).

5 Summary and Outlook

Author: Daniel Richter

Summary

This report summarizes our journey to build a functioning satellite image segmentation model. We have collected and prepared the necessary data sets and trained numerous different models with a variety of customized loss functions and varied hyperparameters. We have taken different steps to improve the models and tried many different variations and optimization methods. During training we used "Google Colab's Pro+" cloud computing service to train our models in total several hundred hours over multiple weeks. We have learned crucially principal parts about image segmentation in general, as well as more elaborate neural network methods, building on the foundational knowledge provided by the Advanced Machine Learning lecture contents.
In our opinion the project was a great opportunity to advance our own skills and get interesting insights into a real world problem being solved by advanced algorithms. Hence, we designate the project as a successful demonstration of knowledge transfer and application.

Outlook

While our results are most certainly not bad, there is always room for improvement. Some thoughts about possible improvements are summarized below:

- Further hyperparameter optimization could help to increase the performance. This includes hyperparameters of the loss functions which could be varied and optimized, as well as the improving the network architecture itself. This, however, would take significantly more time and since we already varied many hyperparameters and tried different network architectures, we would not expect significant differences.

- Increasing the data set size might help for training. The satellite area could be expanded, or the existing area can be augmented by either rotating the images or using different overlapping cropping methods. Since we already needed very powerful GPU's for training and the used data sets were already quite large, these improvements can only be done with more time and better hardware.

- A more recent promising vision transformer we looked into, is the so called *Swin Transformer* proposed by [14] in 2021. It is a hierarchical transformer using **S**hifted **win**dows during calculations. This results in greater efficiency by limiting self-attention and allowing for cross-window connections. The authors further demonstrate great performance, surpassing previous state of the art image segmentation models by large margins. This model is much more complicated and uses a significant amount of necessary knowledge outside the lecture materials, so that we did not chose to further pursue this segmentation model.

A Appendix

Block	Kernel Size	Filters	Stride	Max Pool	Activation Function	Batch Norm	Output	Number of Parameters
conv	3	64	1	-	ReLU	Yes	$128 \times 128 \times 64$	1792+256
conv+max+skip-1-from	3	64	1	2×2	ReLU	Yes	$64 \times 64 \times 64$	36928+256
conv	3	128	1	-	ReLU	Yes	$64 \times 64 \times 128$	73856+512
conv+max+skip-2-from	3	128	1	2×2	ReLU	Yes	$32 \times 32 \times 128$	147584+512
conv	3	256	1	-	ReLU	Yes	$32 \times 32 \times 256$	295168+1024
conv+max+skip-3-from	3	256	1	2×2	ReLU	Yes	$16 \times 16 \times 256$	590080+1024
conv	3	512	1	-	ReLU	Yes	$16 \times 16 \times 512$	1180160 +2048
conv+max+skip-4-from	3	512	1	2×2	ReLU	Yes	$8 \times 8 \times 512$	2359808+2048
conv	3	1024	1	-	ReLU	Yes	$8 \times 8 \times 1024$	4719616+4096
conv	3	1024	1	-	ReLU	Yes	$8 \times 8 \times 1024$	9438208+4096
UP-conv+skip-4-to	2	512	2	-	ReLU	Yes	$16 \times 16 \times (512 + 512)$	2097664+2048
conv	3	512	1	-	ReLU	Yes	$16 \times 16 \times 512$	4719104+2048
conv	3	512	1	-	ReLU	Yes	$16 \times 16 \times 512$	2359808+2048
UP-conv+skip-3-to	2	256	2	-	ReLU	Yes	$32 \times 32 \times (256 + 256)$	524544+1024
conv	3	256	1	-	ReLU	Yes	$32 \times 32 \times 256$	1179904+1024
conv	3	256	1	-	ReLU	Yes	$32 \times 32 \times 256$	590080+1024
UP-conv+skip-2-to	2	128	1	-	ReLU	Yes	$64 \times 64 \times (128 + 128)$	131200+512
conv	3	128	1	-	ReLU	Yes	$64 \times 64 \times 128$	295040+512
conv	3	128	1	-	ReLU	Yes	$64 \times 64 \times 128$	147584+512
UP-conv+skip-1-to	2	64	1	-	ReLU	Yes	$128 \times 128 \times (64 + 64)$	32832+256
conv	3	64	1	-	ReLU	Yes	$128 \times 128 \times 64$	73792+256
conv	3	64	1	-	ReLU	Yes	$128 \times 128 \times 64$	36928+256
conv	1	4	1	-	softmax	Yes	$128 \times 128 \times 4$	260
Total number of Parameters								31,045,636

Table A.1: The final network architecture used for the classical U-Net-backbone.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015.

[2] E. Zanaty, "Medical image segmentation techniques: An overview," *International Journal of informatics and medical data processing (JIMDP)*, vol. 1, p. 1, 01 2016.

[3] G. Landini, D. Randell, S. Fouad, and A. Galton, "Automatic thresholding from the gradients of region boundaries," *Journal of microscopy*, vol. 265, 09 2016.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2014.

[5] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," 2015.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[11] S. Jadon, "A survey of loss functions for semantic segmentation," in *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, IEEE, oct 2020.

[12] D. Duque, S. Velasco-Forero, J.-E. Deschaud, F. Goulette, A. Serna, E. Decencière, and B. Marcotegui, "On power jaccard losses for semantic segmentation," pp. 561–568, 01 2021.

[13] E. Guérin, K. Oechslin, C. Wolf, and B. Martinez, "Satellite image semantic segmentation," 2021.

[14] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021.