HAW HAMBURG

Bachelor thesis

Semi-Automated Labeling in the Domain of Bloom Segmentation

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Noah Palle

Semi-Automated Labeling in the Domain of Bloom Segmentation

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science Angewandte Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Peer Stelldinger
Supervisor: Prof. Dr. Martin Hübner

Submitted on: 2025 January 16^{th}

Noah Palle

Title of Thesis

Semi-Automated Labeling in the Domain of Bloom Segmentation

Keywords

Machine Learning, Semi-automated-labeling, Efficient labeling, Semantic segmentation, Bloom detection, Fruit trees

Abstract

Supervised machine learning has experienced rapid growth in recent years, becoming increasingly relevant to various aspects of our lives. However, one of the significant challenges in this domain is the need for labeled data. Traditional data labeling methods involve substantial manual effort, which can be tedious and resource-intensive. This thesis aims to solve this problem by adapting a semi-automated labeling framework to partially automate the data labeling process. The focus is on implementing and testing the framework for the area of semantic segmentation.

The results show that the adapted framework does not achieve the same results as the original paper. The adjustments lead to the appearance of the problem of catastrophic forgetting, which could not be solved with clearly positive results within the scope of this work. After a lot of manual work and many training runs, the framework does not achieve better results than the standard U-Net training after the same number of epochs.

Noah Palle

Thema der Arbeit

Semi-Automatisiertes Labeln in im Bereich der Blütensegmentierung

Stichworte

Machine Learning, Semi-automatisiertes-labeling, Effizientes Labeln, Semantische Segmentierung, Blütenerkennung, Obstbäume

Kurzzusammenfassung

Machine Learning hat in den letzten Jahren ein rasantes Wachstum erfahren und wird zunehmend zu einem festen Bestandteil unseres Lebens. Eine der größten Herausforderungen im Bereich des Supervised Machine Learning ist der hohe Bedarf an gelabelten Daten. Herkömmliche Methoden des Datenlabelns erfordern einen erheblichen manuellen Aufwand, der mühsam und ressourcenintensiv ist. Diese Arbeit zielt darauf ab, dieses Problem durch die Adaption eines semi-automatisierten Labeling-Frameworks zu lösen, welches den Datenlabelingprozess teil-automatisieren soll. Der Fokus liegt auf dem Implementieren und Testen des Frameworks für den Bereich der semantischen Segmentierung.
Die Ergebnisse zeigen, dass das angepasste Frameworks nicht die gleichen Ergebnisse erzielt wie das originale Paper. Die Anpassungen führen zum Auftreten des Problems des catastrophic forgetting, welches im Rahmen dieser Arbeit nicht mit klar positiven Ergebnissen gelöst werden konnte. Das Framework erzielt nach viel manueller Arbeit und vielen Trainingsdurchläufen keine besseren Ergebnisse als das Standard U-Net Training nach der gleichen Anzahl an Epochen.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The rapid advancement of machine learning has enabled the automation of various tasks, particularly supervised learning, where labeled datasets are essential for model training. However, the creation of these labels, especially in the field of computer vision, remains a labor intensive and error-prone process [10]. *Semantic segmentation*, a key area in computer vision, is highly dependent on high-quality pixel-level annotations for accurate model performance [13]. In the context of agriculture, one practical application of semantic segmentation is the identification and classification of blooms in fruit trees.

Currently, the classification of *bloom strength* is performed manually by assessing the extent of flowering on trees. This process is crucial to determine the application of pesticides and hormones, as well as the amount of thinning that affects tree health and fruit production. Without these interactions, trees would be overgrown or undercropped leading to smaller fruits, poorer fruit quality, and more [14]. However, manual estimating the *bloom strength* is time-consuming, inconsistent, and subject to human error, making it challenging to accurately measure the impact of applied treatments based on the strength. This inconsistency introduces variability into *bloom strength* classification, hindering the ability to reliably evaluate the effects of agricultural interventions.

In an effort to minimize labeling effort in supervised ML, different approaches have been tested. One of them is crowd-sourcing, where a group of individuals perform a task on the internet. One of the most famous examples being Google Captchas where one has to identify objects in street view. But this does not work when expert knowledge is required. Another one is labeling assistance tools like CVAT, which support the human labeler. While these speed up the process, the human still has to cover all the examples of the dataset. Desmond et al. have presented an approach to fix that, which is called *semi-auto-labeling* [7]. It combines an assisting tool with a machine learning model in

a cycle where the model trains on human corrections and makes predictions based on its training, which are then shown to the labeler saving him time until he decides to completely delegate the task to the machine labeler.

1.2 Goal of the thesis

This thesis attempts to adapt and implement the semi-automated labeling framework from Desmond et al. [7] for the task of semantic segmentation. This is done with the goal of reducing the time and cost associated with labeling, thus improving the efficiency of preparing datasets for supervised learning models. This framework will be tested to assess its effectiveness and potential as a viable solution to minimize the burden of manual data labeling in machine learning workflows for the task of semantic segmentation.

1.3 Related work

Research in the field of minimizing labeling effort has received a lot of interest. Various methods have been proposed to address the challenges of labeling efficiency and accuracy, many of which share similarities with the approach adapted in this work. But at the same time many of them focus on pure image classification or do not intend to automate the process but rather support the annotator to be quicker. Many of the works regarding semantic segmentation are from the medical domain, as there are many kinds of scans like a CT or MRT that have to be analyzed. If this work could be automated or semi-automated it would very likely improve the health of all of us.

NuCLS, a framework developed for nucleus classification and segmentation in breast cancer, exemplifies a scalable crowdsourcing approach that integrates AI-assisted interfaces to improve annotation speed and quality [1]. This method involves leveraging expert knowledge while minimizing the manual effort required for label correction, thus enhancing the overall efficiency of the annotation process. But it is not automating the process and rather improves the annotation speed by hand.

In one of his earlier works, Desmond et al. did something similar. Him and his colleagues presented an ai-assisted interface [6]. With it they were able to provide label recommendations and reduce the labeler's decision space by focusing their attention on only the

most probable labels. With that interface, they were able to see 6% improvements in accuracy. Yet, with this approach, still every image has to be labeled by hand.

Later Desmond et al. introduced the Semi-Automated Labeling framework [7], a framework that combines semi-supervised learning, active learning, and human-in-the-loop feedback to reduce the labeling effort. The proposed framework works with user feedback on model predictions until the model is good enough to label the remaining images on its own. This combines the ai-assisted interface with actual automation.

Another approach is scribble-based feedback during training. The paper 'Deep Interactive Learning-based ovarian cancersegmentation of H&E-stained whole slide images to study morphological patterns of BRCA mutation' by Ho et al. [15]. Him and his colleagues present a deep interactive learning approach in which the user gives feedback regarding the model predictions during the training via scribbles. Using a pre-trained model from another medical domain, they were able to train a model in 3.5 hours that was capable of creating labels on a human level. The pretrained model they used was from another work of Ho et al. in which they managed to achieve similar results in 7 hours without a pre-trained model [16].

Sambaturu et al. present image-specific scribble feedback in a non-iterative manner [26]. In their work, the feedback is only used for creating proper annotations, but not for training a model along the way that might be able to automate the task. They were able to achieve improvements of up to 12.4 times in user annotation time compared to a full human annotation. It is kind of similar to the first two papers [1][6] but it adds another layer. In their work the user is able to provide feedback in the form of scribbles which are used by the model to improve the annotation until the user if satisfied and moves to the next image.

In the agricultural domain, Bhattarai et al. developed a CNN based algorithm for automatic blossom detection in apple trees [2]. Using pre-existing annotated datasets, their work focused on enhancing segmentation accuracy and deploying the model in real-world environments. They did instance segmentation trying to separate the single blossoms. Although this approach does not incorporate semi-automated labeling or human-in-the-loop strategies, it demonstrates the potential of a blossom-detection model.

1.4 Structure of the thesis

This thesis is structured into the following six chapters, excluding the introduction:

Chapter 2 provides the theoretical foundation for semi-auto labeling, semantic segmentation, and the U-Net architecture.
Chapter 3 presents the dataset as well as the adaptations made to the framework.
Chapter 4 explains the metrics used to evaluate the framework.
Chapter 5 covers the experiments and their respective results.
Chapter 6 evaluates the results of the experiments.
Chapter 7 formulates a conclusion and provides an outlook for future work in this area.

2 Fundamentals Machine Learning and Labeling

In the following, topics relevant to the context of the entire thesis will be explained for a better general understanding. These architectures and methods are important as they make up the foundation for the experiments and evaluation.

The words human labeler and oracle, as well as cycle step and iteration, will be used interchangeably in this work.

2.1 Machine learning foundation

The field of image processing involves two main learning paradigms: supervised and unsupervised learning. In **supervised learning**, the model learns from pre-labeled inputs, which serve as targets. Each training example consists of input values (vectors) and one or more designated output values [24]. The objective of this training method is to minimize the overall classification error, also known as loss. It is determined by a specific function based on the context.

Unsupervised learning differs in that it does not involve labeled training sets. The success of this approach is typically evaluated based on the network's ability to minimize or maximize an associated cost function [23]. It is important to note that most machine learning models are currently trained through supervised learning [10].

Labeling
Since SML needs these labeled datasets, the process of label creation is crucial. It is called data labeling, and during the process, raw data is annotated with meaningful labels that guide the learning algorithm. This process involves pairing each input data point with an expected output, which serves as the ground truth from which the model learns [24].

2.2 Learning Paradigms

To overcome the time and cost issues of the labeling process, learning paradigms have been evolved that need less human interaction, less pre-labeled data, and thus speed up the process.

Semi-Supervised Learning (SSL)
Semi-supervised learning [34] is one of them and is in the middle between supervised and unsupervised learning. It uses a large amount of unlabeled data together with a small amount of labeled data and attempts to draw conclusions from these two datasets. Then it tries to apply these conclusions while making label predictions on the unlabeled data. Therefore, it is very well suited for datasets in which only a small portion is labeled [34]. In SSL there is no interaction with a human or other system that monitors the process or interacts with the labeling process. Therefore, it relies entirely on the model being correctly biased based only on the initial labeled images. There is a lack of further options for controlling the model to achieve better results.

Active Learning (AL)
Another of these learning paradigms is Active Learning [27] which works well with a small amount of labeled data, just like SSL, and iteratively trains increasingly accurate models. In doing so, the algorithm attempts to choose the next examples of the data (also called batch) to be labeled based on the largest uncertainties in the labeling process. These examples are labeled by an oracle (usually a human). The goal is to gain as much knowledge as possible by selecting images that the model cannot yet label with a high degree of certainty to improve quickly. The uncertainties of the model are actively focused to accurately label as quickly as possible [27].
This speeds up the process, but there is no active label support from the model, and the oracle does not receive any feedback, according to which it can decide how well the process is working and how accurate the model actually is. Another instance where the progress and performance of the system is checked would be very helpful.

Interactive Learning (IL)
The third paradigm solves the issue of missing interaction and control: Interactive learning [17] includes an expert in the standard machine learning training routine. Expert knowledge can help in cases of limited or nonexistent training data, and can oversee and control the labeling process. He does not have to do everything by hand; for example,

the model suggests labels and only the expert has to decide whether they are correct or not [17].

2.3 Semi-Automated Labeling

All these three paradigms have their benefits but only cover a part of the optimal automation cycle. However, by combining all of them into a framework to overcome their drawbacks and combine the benefits, the process of data labeling can be automated as well as possible. That is exactly what 'Semi-Automated Labeling' [7] does.

It is a labeling approach that combines the advantages of IL, AL and SSL in an iterating manner while working well with a small amount of labeled data. It is a refinement of the assisted labeling approach with an AI assistant interface that has already shown significant increases in accuracy and speed, depending on the setup, in a labeling task [6]. In that approach, every single image is presented to the labeler together with generated labels and the oracle applies corrections if necessary. But the approach has its limitations, hence the labeler still needs to consider and manually label every example in the dataset.

Semi-Automated Labeling [7] improved this by constructing a framework that iterates over the data and offers the option of completely handing over the labeling task to the algorithm. It therefore is a system consisting of a human labeler having control over the labeling flow and a machine labeler (model) supporting the oracle and being able to finish the task on its own when it is delegated. To properly assist and make predictions, the model still needs an initial amount of labeled data. This data, if not already present, is selected by bootstrapping and labeled by hand.
Bootstrapping is the process of selecting a meaningful subset that represents the dataset. In their work Desmond et al. [7] showed that K-means clustering works well for this task over a random selection. After the model trained on that ground truth data the following steps occur during the cycle [7]:

1. **Label Spreading:**
   At the start of every iteration the machine labeler predicts label distributions for the remaining unlabeled data. That is, it predicts the certainty for every label.

This happens by using a semi-supervised based algorithm that works well on a low percentage of labeled data in a dataset.

2. **Min-margin Active Learning:**
   Now an active learning heuristic is applied to select a batch (a predetermined number of unlabeled examples) with the most uncertain predictions. Focusing on the weaknesses of the algorithm (the highest uncertainty) helps to have a descending gradient of labeling difficulty.

3. **Assisted Labeling:**
   Afterwards the oracle is presented with the batch of the most uncertain examples with the corresponding predicted labels. One after another. Now, it can accept the predicted labels or make corrections. Afterwards, the pre-trained model is being trained on the additional data.

4. **Checkpoint:**
   The human labeler is presented with different metrics that display the current performance of the machine labeler (the algorithm) and decide whether the rest of the unlabeled data should be automatically labeled or if further iterations are necessary for refinement.

5. **Auto Labeling:**
   The machine labeler (algorithm) will label the remaining unlabeled data on its own.

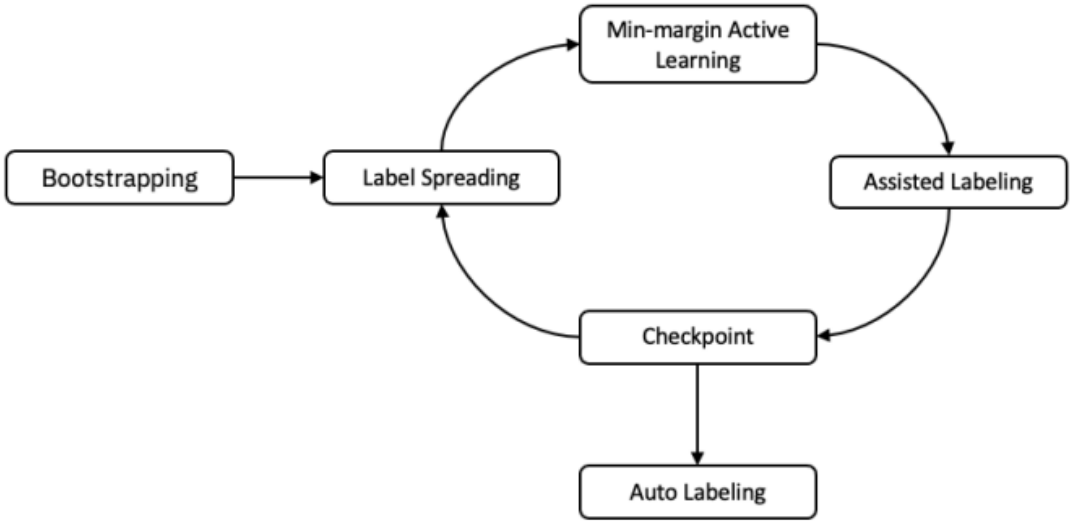

Figure 2.1: The outlined structure of the semi-supervised labeling framework.

2.4 Semantic segmentation

After knowing the framework, it is important to understand the machine learning area for which the label generation should be automated.
Semantic segmentation is a process in computer vision that involves classifying each pixel in an image into a predefined category, also called *class* [13]. This makes it very attractive for real-world applications, such as self-driving vehicles [21], pedestrians [3], and bloom detection, since it can not only identify different objects in an image but also their respective positions. Whilst doing that it won't differentiate between different occurrences of the same class. These pixel labels are usually based on image features like color and shape.

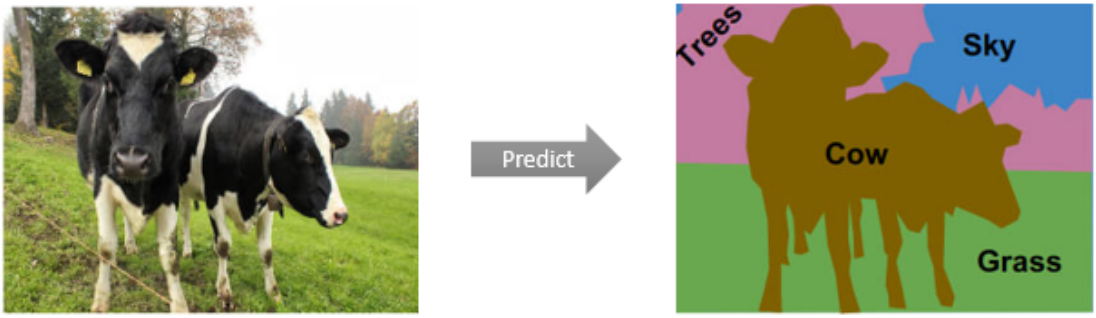

Figure 2.2: An example of semantic segmentation with four classes: Trees, Sky, Gras and Cow.

https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

2.5 Convolutional neural network (CNN)

To recognize such shapes and patterns in images a special type of neural network is needed, a so-called convolutional neural network or CNN. This is a type of feedforward network that is able to detect patterns through convolutional layers. These are hidden layers that contain a set number of filters, which are matrices that are usually smaller than the input image. The filter size can change from one layer to another. These filters run over the image trying to find specific features, resulting in a mapping of that feature, namely a feature map. In the initial layers of a CNN, feature maps capture simple patterns such as edges or straight lines. As the network deepens, later layers begin to identify more abstract features, including objects such as people, animals, or other complex structures. Essentially, each filter within the network learns to detect a specific pattern present in the input data.

To decrease the complexity of this task, pooling layers are introduced. They serve to down-sample feature maps, reducing their spatial dimensions. This process decreases the number of parameters that the network must learn, thus lowering the computational complexity.

Although fully connected layers are commonly added at the end of a CNN for tasks such as classification, semantic segmentation requires the output to have the same spatial dimensions as the input. This is typically achieved using a U-Net or similar architecture, which enables the network to maintain spatial information throughout the process.

2.6 U-Net

The U-Net is a convolutional neural network architecture designed for image segmentation in scenarios where only a low amount of labeled data is given. It was specifically designed to make efficient use of smaller datasets while ensuring both speed and accuracy [25]. Because of this, the U-Net architecture perfectly aligns with the goal of this thesis.

Developed by Ronneberger et al. [25] in 2015, U-Net has gotten the name from its symmetric U-shaped structure, which consists of two main parts: a contracting path and a symmetric expansive path. The contracting path is a typical deep convolutional network that reduces the spatial dimensions of the input while capturing features, while the expansive path upsamples the reduced data back to the original resolution, making pixel-level predictions. To cut it short, the contracting path captures contextual information, and the expanding path enables precise localization [25].

Both sides are made out of n blocks and are connected by the bottleneck. Each of the blocks b_i^c with $1 \leq i \leq$ on the **contracting** path consists of multiple convolutional layers followed by a pooling layer resulting in a downsampling of the feature maps. The output o_n^c of the last block b_n^c is given to the bottleneck, which runs the output through more convolutional layers followed by an upsampling operation, resulting in the output x which is given to the highest block on the expanding path.

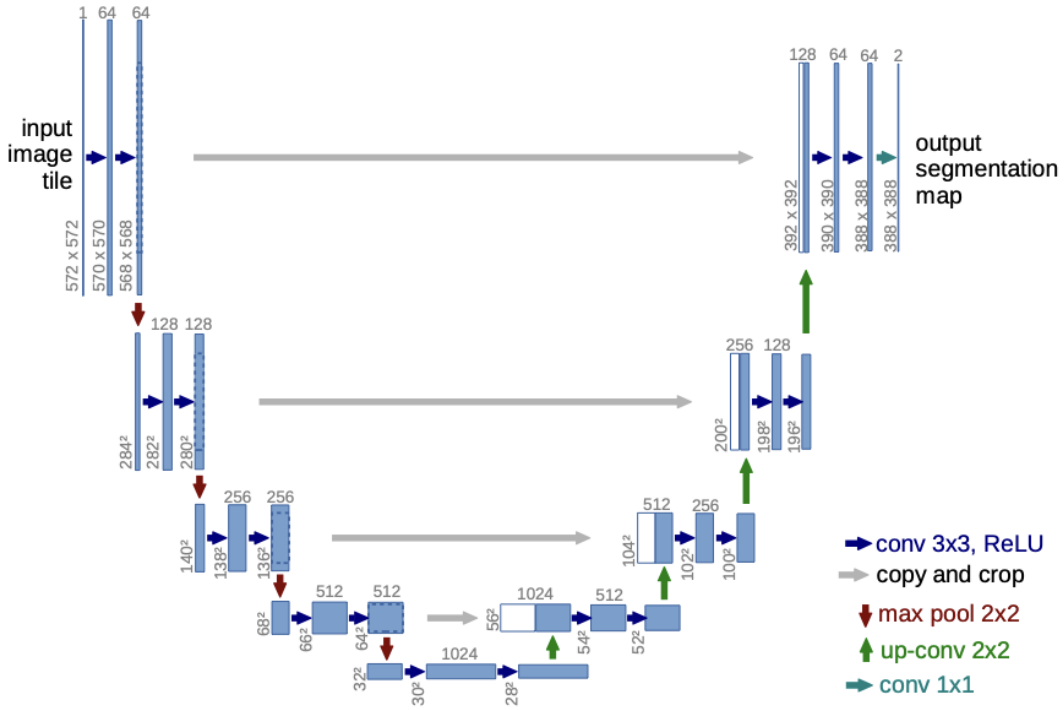

Figure 2.3: U-Net architecture from Ronnberger et al. showing the contracting path at the left and the expansive path at the right [25].

The **expanding path** is symmetric to the contracting path consisting of $m = n$ blocks. These blocks b_i^e similarly contain multiple convolutional layers but they are followed by an up-convolutional layer resulting in an upsampled output o_i^e. Additionally to the output of the previous block o_{i+1}^e, each block gets the output o_i^c of the corresponding block from the contracting path through *skip connections*, shown as grey arrows in Figure 2.3. These connections allow for both high- and low-level features to be preserved and learned, reducing any information loss that occurs in the following blocks during the contracting path. The outputs o_i^c and o_i^e are concatenated before the convolutional operations occur. The final output o_1^e of the expanding path is a segmentation map of the size of the original image.

Having established the foundational principles of machine learning relevant to this thesis, the next chapter addresses the dataset that serves as the backbone of the model.

3 Setup

This chapter will cover the changes made to the original Semi-Auto-Labeling cycle to properly fit it to the task of this thesis. But before doing so, it will describe the dataset and the issues which had to be solved before properly running the cycle.

3.1 Dataset

The dataset to be labeled consists of 339 images taken by a ZED 2i camera, which is part of a bigger sensor box connected to the back of a tractor. They are 1080x1920 pixels in size and were taken while driving through fruit tree rows at the Esteburg Research Institute [9]. They were taken on separate days under different weather conditions and times of the day. The images display apple trees during their blooming-phase.

3.1.1 Imbalanced Dataset

One of the issues machine learning projects often have to tackle is class imbalance. This data set has far more background pixels than bloom pixels. Therefore, there are significantly more labels and instances of the *Not_Bloom* class. This is an issue, which can lead to the model being biased towards the majority class.

Common solutions would be balanced batch creation or over- and undersampling. To balance the batches, the program would have to precisely pick the same amount of samples per class and train on these, but this would lead to way less batches and epochs being trained on, which would lead to a worse model. Over- and undersampling is an approach where samples of the minority class are randomly duplicated and samples of the major class are being randomly removed from the dataset. This isn't an option either

Figure 3.1: A graphical representation of the tractor with the sensor-box.

since both classes appear in every sample and not apart from each other meaning that samples cannot be removed or added easily.

Fortunately, there are other methods that work for the use case of this thesis, one of them being **class weighting** [11].

Class weighting is a technique in which different weights are assigned to classes within the loss function. This ensures that the model pays more attention to the minority class during training, as it receives a higher weight. The weights are assigned inversely proportional to the frequency of class appearances in the dataset.

By modifying the loss function, this approach penalizes errors in predicting the minority class more heavily than errors in predicting the majority class.

The weight for each class can be calculated using the following formula:

$$w_i = \frac{N}{n_i} \tag{3.1}$$

where:

- w_i is the weight for class i,

- N is the total number of samples,
- n_i is the number of samples in class i.

Once the weights are calculated, they can be incorporated into the cross-entropy loss function. The weighted formula for binary cross-entropy is as follows:

$$L_{\text{weighted}}(y, p) = -\frac{1}{N} \sum_{i=1}^{N} [w_1 \cdot y_i \log(p_i) + w_0 \cdot (1 - y_i) \log(1 - p_i)] \tag{3.2}$$

- $L_{\text{weighted}}(y, p)$: The weighted binary cross-entropy loss.
- N: The total number of samples.
- y_i: The actual label of the i-th sample, which is either 0 or 1.
- p_i: The predicted probability that the i-th sample belongs to class 1.
- w_1: The weight given to the class with label 1.
- w_0: The weight given to the class with label 0.

The loss function is an if not the most important part of the training as it defines the changes made to the models weights during each iteration. This makes applying the proper weights to the loss crucial.

3.2 Cycle adaptions

After properly covering the dataset, the cycle had to be adapted to the differentiating scope of this work compared to the work of Desmond et al.[7].

The first step to be performed was to select a small portion of the images(10%) to be manually labeled for the initial training. In the original work [7], this selection process was executed using a k-means clustering. This approach was shown to work better in finding a meaningful subset than in choosing random ones. In this thesis, bootstrapping was done by hand. The samples were not chosen randomly but purposefully. In the domain of this work the recognition of images covering the different scenarios works semantically quite well. Therefore, a meaningful subset was created manually.

Figure 3.2: Example of a ground truth image with its hand labeled version on the right. The *Bloom* class labels are pink and the *Not_Bloom* class labels are yellow.

The labeling effort did not cover all pixels as seen in Figure 3.2. This was done to save time. Labeling an image in this way already took an average of 40 minutes. The human labeler covered all the different colors and appearances of the two classes to the best of his knowledge. The unlabeled pixels are being ignored in the loss function by creating a label map with ones for the labeled pixels and zeros for the unlabeled pixels. That map is multiplied with the loss matrix of the image to calculate the final loss.

The initial label prediction has been changed to use a standard U-Net output instead of the label spreading algorithm. In image segmentation, each pixel in an image needs to be labeled, rather than one label per image. Label spreading, however, is designed to work on a graph where each node represents an entire data point. Extending it to segmentation would mean building a graph at the pixel level, which is computationally

expensive and challenging for large images. It is also working with a global context over images and is based on similarity between those. It therefore might struggle to capture these nuanced variations at a pixel level, particularly when edges or boundaries are sharp or irregular.

Or in other words, it wasn't designed for the task of this thesis. The U-Net instead works well without including the information of the unlabeled images. It creates label prediction maps for all unlabeled images at each cycle step and has proven its performance in the past [25].

The min-margin heuristic is applied to all the images, as in the original paper. It is calculated per pixel and then averaged over the image. The batch of the most uncertain images is selected based on this score.

Probably the biggest adaptation has been made in the assisted labeling phase. Although the oracle had to assign the correct label to each image in the original work [7] this wasn't going to work here as one datapoint is one pixel. Identifying and correcting each mislabeled pixel from the machine labeler would take nearly as long as labeling the image from scratch. Assuming the human would even find all the pixel-errors. A new type of human feedback was needed.

Figure 3.3: Example image from the assisted Labeling. **Left**: Image with machine predicted labels. Pink for *Bloom* and yellow for *Not_Bloom*. **Middle**: Image with human correction labels. Blue for *Corrected_Not_Bloom* and green for *Corrected_Bloom*. **Right**: Combination of both, which the model receives for further training.

Introducing **major issue correction feedback**: Instead of correcting every pixel the feedback the goal is to correct only the biggest 'flaws' of the machine labeler. The ones that stood out and are big enough to be correctable in a time frame of 15 to 20 minutes. Now there is a set of two label maps per image. These two label maps are merged in the way that human labels overwrite machine labels. This combination map is returned to the machine labeler as the label map for training in this cycle step. All pixels that have been marked as *Corrected_Not_Bloom* are being converted to *Not_Bloom* and all the *Corrected_Bloom* labels are converted to *Bloom* labels. The rest of the labels are kept from the model prediction.

In Figure 3.3 you can see that the machine labeler was struggling with the metal chords connecting the stems, the transition from trees to the sky, the trunk and some of the branches, as well as the brightest part of some blooms. These miss-predictions, which were clearly visible, occurred multiple times and covered multiple pixels, were then corrected by the human labeler.

After training on the corrected batch of the most uncertain data, just like in Semi-Automated Labeling [7], metrics for the checkpoint are presented to the oracle. These metrics were also adapted, but will be described in the evaluation chapter.

The only thing left in the iteration is the decision if there should be another one. In the event that the performance of the machine labeler satisfies the oracle, the work is delegated. The machine labeler will label all the data including the ground truth. Neither will it skip the hand-corrected images from earlier and count them as labeled as only major errors were corrected. There might still be minor issues which should not persist in a training dataset. The labels should be as accurate and high quality as possible.

All of this results in the following framework:

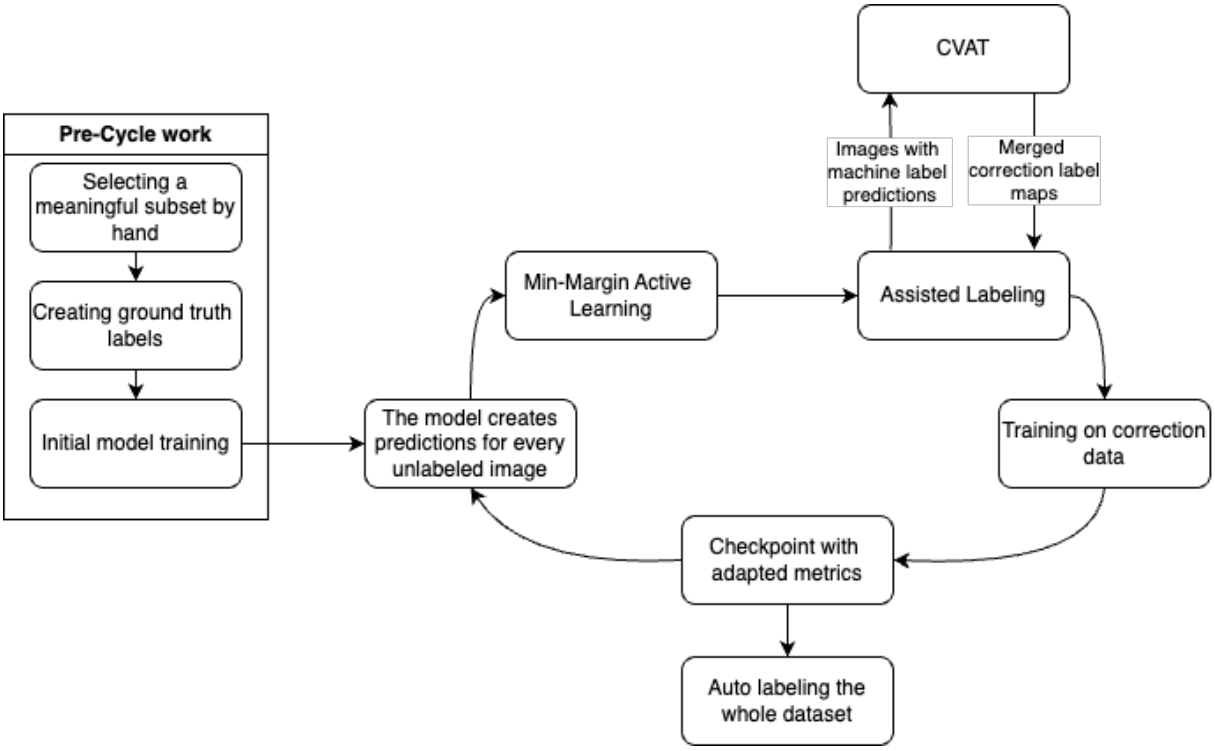

Figure 3.4: The adapted labeling cycle.

The final output of the framework is a new dataset containing all images with labels for every pixel in every image. The labels belong to the classes: *Bloom* or *Not_Bloom*.

4 Evaluation Metrics

In the previous section, the setup and dataset characteristics that form the foundation of this work have been detailed. Based on this foundation, it is essential to outline the metrics that will be used to evaluate the performance of the proposed approach. These metrics are critical not only for assessing the effectiveness of the model, but also for tracking its progress and supporting the oracle pinpointing the moment of delegation.

Starting with **accuracy**, which is a widely used metric in classification tasks. However, in semantic segmentation, its relevance is limited due to significant class imbalances. Such imbalances can lead to high accuracy scores even though the model predicts the whole image as background or, in the case of this thesis as *Not_ Bloom*. To address this, alternative evaluation metrics have been employed.

4.1 Confusion matrix

To understand some of these other metrics, it is necessary to first introduce the confusion matrix.

This paragraph covers the scenario of this thesis in which blooms should be classified as *Bloom* and background as *NotBloom*. As seen in Figure 4.1, the matrix shows the four different cases that can appear while comparing the predictions of the model with the ground truth:

1. True positives (TP) are blooms that are correctly classified.
2. True negatives(TN) are correctly classified background.
3. False positives(FP) are backgroundpixels labeled as *Bloom*.
4. False negatives(FN) represent missed bloom pixels that are classified as *Not_ - Bloom*.

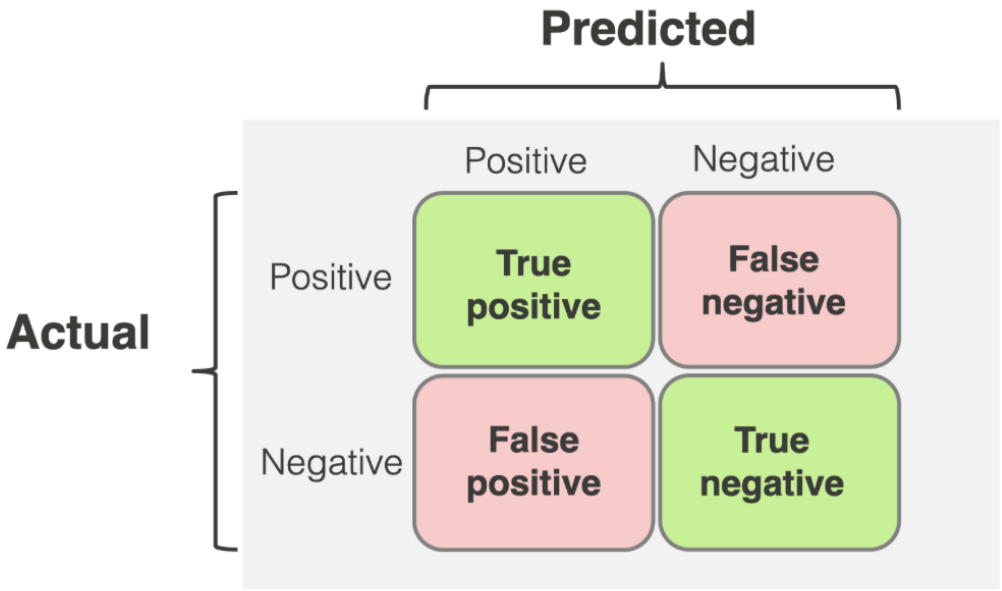

Figure 4.1: Confusion matrix standard schema.

https://www.evidentlyai.com/classification-metrics/confusion-matrix

4.2 Dice coefficient

Based on these four cases performance measurements such as the dice coefficient can be calculated. It is one of the typical measurements used in the context of semantic segmentation, also known as the f1 score, which combines the different cases of the confusion matrix into one number. It is calculated by 2*area of intersection divided by the total number of pixels in both the prediction and the ground truth image.

$$dice = \frac{2TP}{2TP + FP + FN} \tag{4.1}$$

This means that the TN are being ignored having the benefit of actually focusing on the *Bloom* class and not having to worry about the majority of the pixels being correctly classified and therefore influencing the score drastically. A high dice score of 1 reflects a model that is able to perfectly recognize and classify the foreground pixels.

4.3 Checkpoint feedback metrics

Additionally to the metrics used to evaluate the final model, some metrics providing feedback in the cycle are necessary to support the oracle on the delegation decision of the labeling task. Exactly like the ones introduced by Desmond et al. [7].

Him and his colleagues are introducing agreement, agreement similarity, and labeling difficulty. Agreement simply counts the number of times the machine labeler and the oracle agree in the current cycle step, agreement similarity takes into account all of the models' label predictions and their probability to calculate a more sophisticated agreement number. Labeling difficulty is the mean scaled entropy of the examples in the current iteration, with scaled meaning that the result is a number between 0 and 1 [7].

All of the following metrics are averaged over the number of images in the current iteration.

4.3.1 Average uncertainty

Due to the already existing min-margin calculation per image as described in *Semi-Automated Labeling* it was used to measure the labeling difficulty.

$$uncertainty = \frac{\sum_{i=1}^{m} 1 - p_i^{min-margin}}{m} \tag{4.2}$$

- m: The amount of pixels in an image
- $p_i^{min-margin}$: The calculated min-margin between the class probabilities of the i-th pixel

This metric is expected to trend downward as the machine labeler learns and improves on distinguishing the blooms from the background.

The paper of Desmond et al. [7] is about standard image classification. To recreate the agreement metrics of his work, the oracle in this work would have to make pixel-perfect corrections on every pixel of the predicted label image. This would totally annihilate the time saving goal. Thus, two different new metrics were introduced. The *changed pixel count* and *accuracy per class*.

4.3.2 Amount of correction

This metric returns the number of pixels corrected by the oracle in the assisted labeling step.

$$correction\_amount = \sum_{j=1}^{m} c(p_j) \tag{4.3}$$

- m: The amount of pixels in an image
- $c(x)$: A function identifying the pixels that were corrected
- p_j: The j-th pixel in the image

This metric helps the oracle to understand the model due to its simple intuitive approach along with the other more complex metrics. Due to the decreasing labeling difficulty, this metric is expected to decrease as well.

4.3.3 Accuracy per class

This metric conquers the imbalance problem that the standard accuracy has by calculating the accuracy per class.

$$Accuracy_c = \frac{TP_i^c}{TP_i^c + FP_i^c + FN_i^c} \tag{4.4}$$

- $Accuracy_c$: The accuracy for class c.
- TP_c: The number of true positives for class c
- FP_c: The number of false positives for class c
- FN_c: The number of false negatives for class c

Using this metric, the human labeler can clearly see the effects of the last corrections. It gives a more in depth look of why the dice coefficient changed the way it did in the current iteration.

5 Experiments

With the evaluation metrics defined, this chapter focuses on the experimental execution and the preliminary results. These experiments are conducted to validate the proposed framework and analyze its performance in the next chapter.

The experiments were all executed using the U-Net architecture for the model. They were run on NVIDIA T4 GPU's with 15GB internal memory and 52GB system memory. The execution happened on Google hardware through Google Colab [12]. The programming language used was python3.10 with the TensorFlow [29] framework and the Keras [20] library.

5.1 Initial training

The first experiment focused the initial training on ground truth data. The goal was to figure out the epoch until which it is the most effective to train the model before actually starting the cycle.

Setup
In this first experiment, the ground truth size was 10% of the whole dataset and the validation set size was 2% of the dataset. The weighted cross-entropy function, described in chapter 2, was used as loss function.

Preliminary Results
The validation loss curve continuously remained well below the training loss curve. This is rather unusual because it means that the model performs better on data it has never seen before than on the training data. Therefore, training was stopped. Having a high validation loss and a low training loss indicates overfitting, having a high validation loss and a high training loss indicates underfitting, and in the case of both being low, the model is trained well. But the case shown in the graph was not expected. Reasons for the

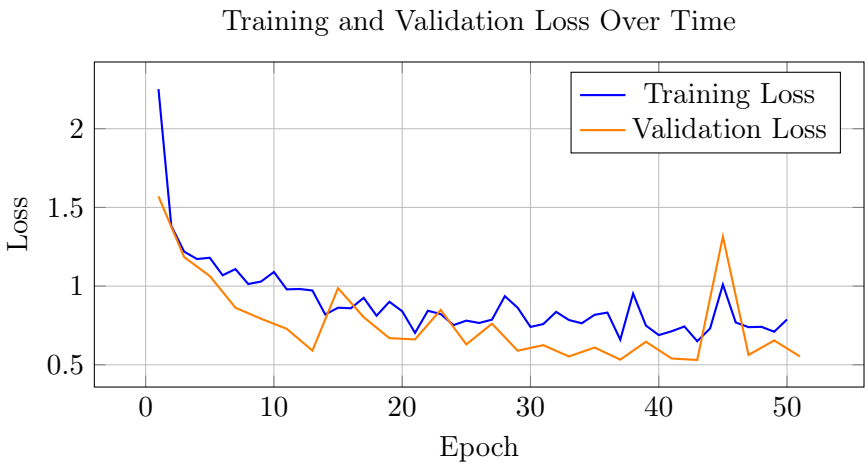

Figure 5.1: Training and validation loss evolution over 50 epochs.

behavior could be training data leaking in the validation set, the timing offset between the training loss calculation and the calculation of the validation loss, or the validation set was too small. Due to that another experiment was conducted using 12% of the dataset as training data and 4% as validation data. The loss function stayed the same.

As the results with this new distribution show, the validation set was indeed too small.

Preliminary Results with new setup

The loss chart shows the case where both the validation and the training decrease over time and stay pretty close together. No signs of over- or underfitting. The loss starts at 1.4 and goes down to 0.4. It starts to lose its downward momentum at epoch 80.

Now to figure out the point at which the cycle should start, it makes sense to look at the other metrics as well.

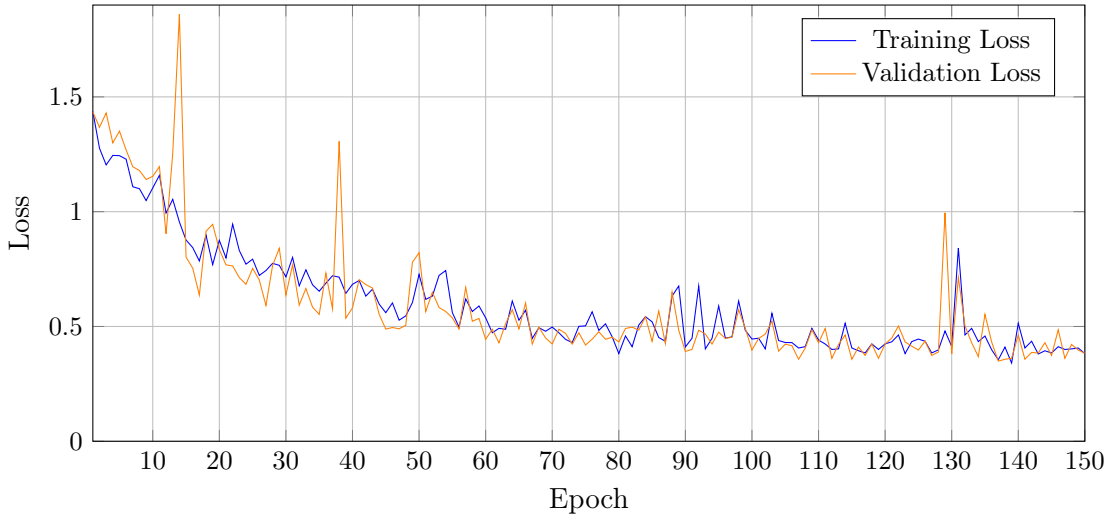

Figure 5.2: Training and validation loss evolution over 150 epochs.

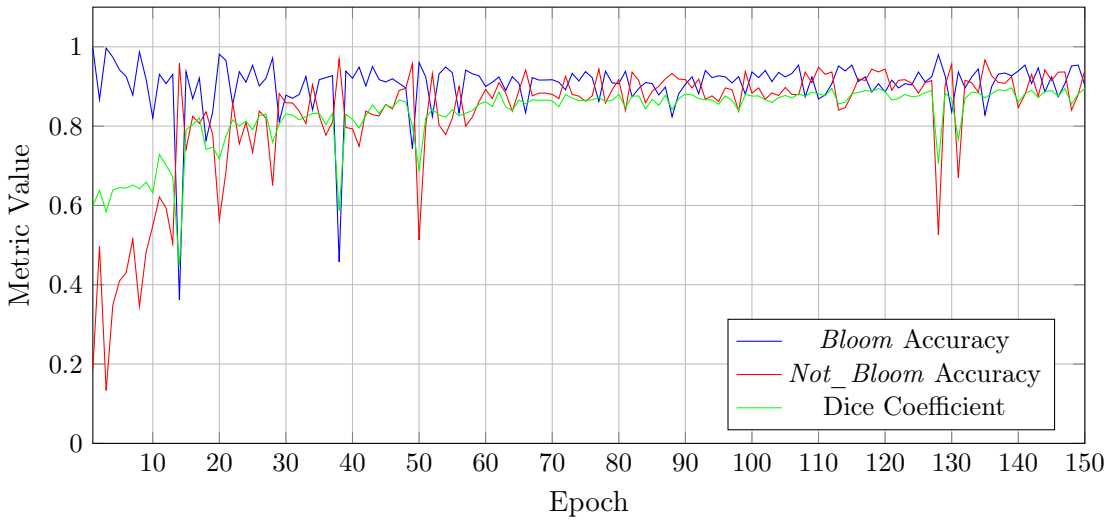

Figure 5.3: Accuracy metrics evolution over 150 training epochs.

Looking at the accuracy metrics in Figure 5.3, a clear upward trend is visible. Focusing on the green line indicating the value of the dice coefficient it shows that its losing a lot of its momentum around epoch 80 as well.
Adding the uncertainty metrics into that train of thought, it is clear that epoch 80 is a good time to start the cycle. The model seemingly scratches the lowest uncertainity it can get to from shortly before epoch 80 until the end of the graph. From epoch 80 until

150 there is no clear sign of the model confidently crossing the 0.1 line downwards. The cycle training therefore starts with a dice score of 0.8799 and a loss score of 0.4325.

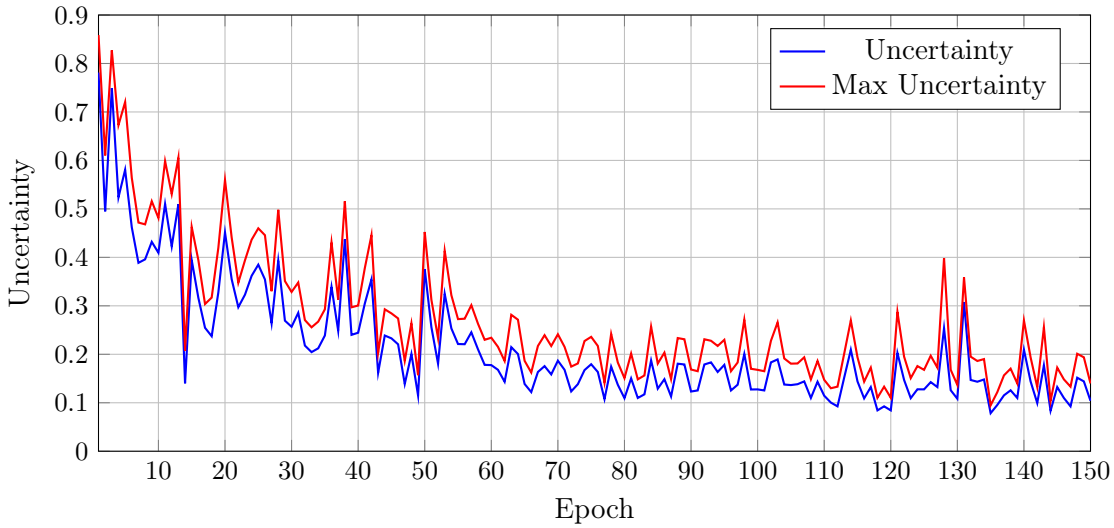

Figure 5.4: Uncertainty metrics evolution over 150 training epochs.

5.2 Cycle Steps

With the pre-trained model set, it was time to test the cycle. The U-Net model, which was pre-trained over 80 epochs was used as foundation for improvement in the cycle. At this point, there were three distinct datasets. The ground truth dataset(12%), the evaluation dataset(4%) and the unlabeled training dataset(84%).

Setup

All experiments were performed twice and the results were averaged. In each iteration, the model predicted the labels on the whole training dataset and selected the 10 most uncertain samples for correction. The oracle corrected them in the major issue correction feedback style using CVAT for assistance. Between 11,4 to 13,7% of the pixels labels were corrected by hand. The correction average was 12,5%.

After correction, the model trained on the corrections using the weighted cross-entropy loss. The learning rate(lr) was set to the values 10^{-3}, 10^{-4} and 10^{-5}.

In addition, a ratio was used. The ratio indicates the percentage of data used for training that is attribute to corrections. But note that the 10 selected images will always be part of the training, instead of reducing those, the amount of ground truth data which is added to the total training amount, increases as the ratio decreases. With a ratio of 100%(1.0) only the 10 correction images are part of the training. If changed to 50%(0.5) the 10 correction images are used alongside 10 ground truth images. This concept has been introduced after noticing that exclusive training on the correction data didn't seem to work.
The ratios 0.1 and 0.2 are not included as the ground truth does not include enough images to reach the right distribution. However, these ratios are expected to perform worse than the examined ones and follow the Gauß-like curve where the mid-range ratios perform the best, as seen in Figure 5.6.
The 10 most uncertain images with their corrections remained the same for all experiments in this section.

Preliminary Results

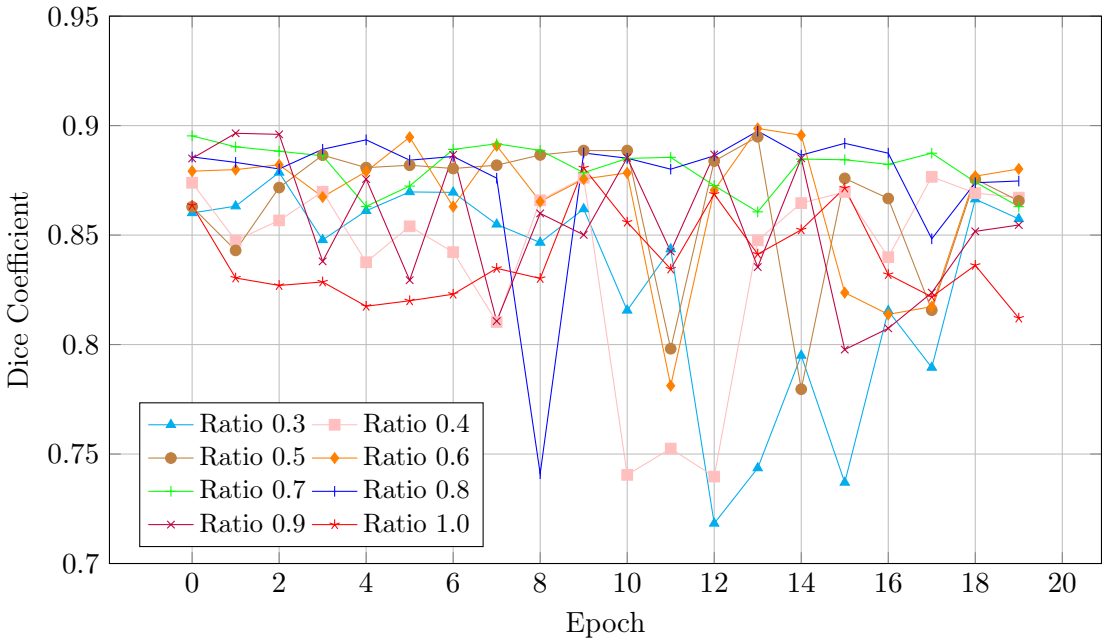

Figure 5.5: Comparison of Dice coefficients across different ratio values with learning rate 10^{-3} over 20 epochs.

		Ratio							
		0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
lr	10^{-3}	0.8574	0.8671	0.8656	**0.8802**	0.8630	0.8747	0.8546	0.8121
	10^{-4}	0.8702	0.8725	0.8772	0.8734	**0.8785**	0.8784	0.8752	0.8661
	10^{-5}	0.8767	0.8758	**0.8782**	0.8775	0.8784	0.8735	0.8760	0.8703

Table 5.1: Dice score results after 20 epochs. The best result per lr is **bold**.

As seen in Figure 5.5 and Table 5.1 all the runs achieved a lower dice score than 0.8799, which was the score after the initial training. Except for the lr=10^{-3} ratio=0.6 configuration which reached a dice score of 0.8802 after 20 epochs. This is a bit less than the initial training reached in epoch 90. In Figure 5.6, it becomes clear that the runs have even higher temporary dice scores. But these didn't turn into an upward trend. It is visible as well that the mid range ratios performed the best and the outer more extreme ratios achieved worse scores. Figure 5.5 and Figure 5.6 display the runs with a learning rate of 10^{-3} as they performed the best.

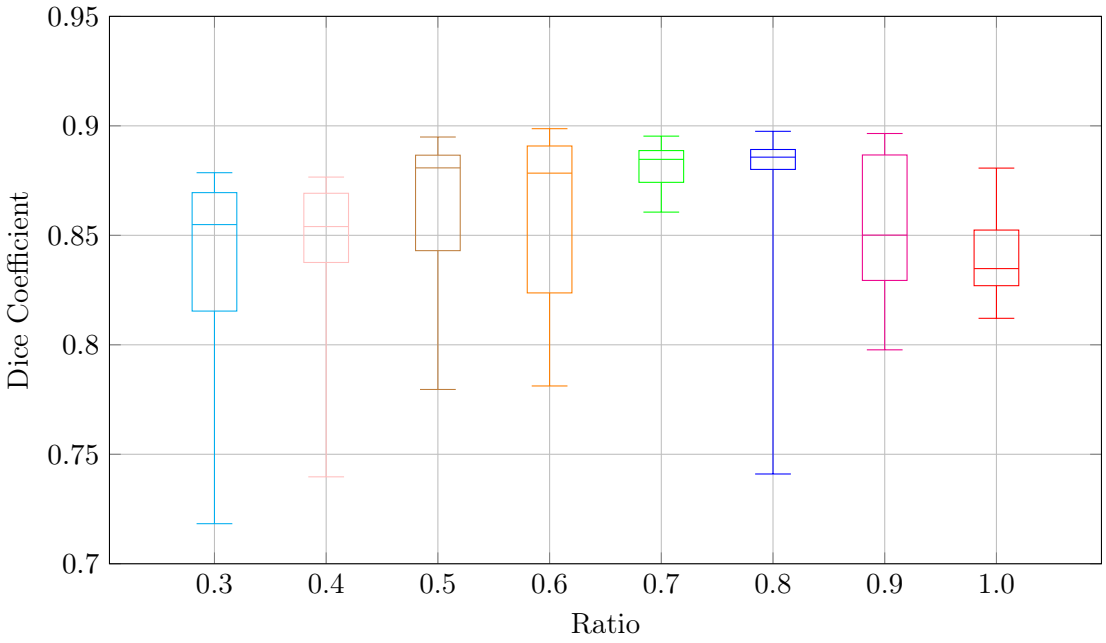

Figure 5.6: Box plot showing the distribution of dice coefficients for each ratio. The boxes show the interquartile range (IQR) with the median line, while the whiskers extend to the minimum and maximum values.

	Ratio							
	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Upper Whisker	0.8786	0.8766	0.8949	**0.8987**	0.8953	0.8975	0.8965	0.8807
Upper Quartile	0.8695	0.8692	0.8866	**0.8908**	0.8887	0.8892	0.8867	0.8524
Median	0.8549	0.8540	0.8808	0.8784	0.8847	**0.8857**	0.8501	0.8348
Lower Quartile	0.8154	0.8376	0.8430	0.8237	0.8742	**0.8801**	0.8294	0.8270
Lower Whisker	0.7183	0.7397	0.7796	0.7812	**0.8606**	0.7410	0.7977	0.8121

Table 5.2: Boxplot values of Figure 5.6. Highest value per row across the ratios is **bold**.

All of these results have something in common. Except for run lr=10^{-3} ratio=0.6. They suffer more or less from *catastrophic forgetting*. This is a term introduced by Michael McCloskey and Neal J Cohen [22] in 1989. It describes the scenario in which a connected network is trained on a set of items and then trained on another set of unknown items in the same domain. In this scenario, neural networks tend to completely and abruptly forget previously learned knowledge [19]. As McCloskey et al. state: "New learning may interfere catastrophically with old learning when networks are trained sequentially" [22].

That meant that a solution was needed that could feed the model with the new information from the correction data whilst continuing the knowledge of the network.

At this point, it became clear that it is rather hard to tackle this problem in general but specifically in the domain of semantic segmentation. There is no work out there that is trying to improve a pretrained semantic segmentation model based on human feedback combined with model assumptions to improve its overall performance with limited labels and effort that presents a solution that can be implemented without major changes. But in this work, there was a need to improve the way the model receives its feedback.

During literature research two surveys from 2021 and 2022 covering the areas of active learning and human-in-the-loop in the medical domain [4] and human-in-the-loop on its own [31] were quite interesting. Both sum up the current state of the art. In neither of these two, an approach is described that properly combined human-in-the-loop feedback and semantic segmentation to improve and speed-up model training and, therefore, labeling. The second survey even describes the exact approach of this thesis in it's future work and unanswered questions chapter.

Still, there are some approaches, but they make the assumptions of:

1. Not having the previous image data and labels anymore [5].
2. Only a small part of an image is used/ only a small number of annotations / selections have to be made by the oracle [33].
3. Wanting to introduce a new context. For example, a new class to recognize or a new task to do [18][8].

Therefore, these are quite complex, and none of them fit the context of this work. Since this thesis does not have all of these restrictions and creating the new best solution for solving the catastrophic forgetting issues in the context of semantic segmentation isn't the goal of this thesis, the issue was approached with the only solution that was adaptable to this thesis in the time frame given.
It was proposed in the paper: Interactive Medical Image Segmentation Using Deep Learning With Image-Specific Fine Tuning from Wang et al. [30]. Him and his colleagues presented an interactive segmentation framework.

The goal is to train a model for binary segmentation. It should mark the areas of a humanly preselected bounding box that covers part of an image. After an initial training, the model makes predictions on unseen image bounding boxes. The areas in these boxes which are falsely labeled are then marked by a human with scribbles. The scribbles and the initial model prediction are given back to the model for further training.

Instead of a bounding box, the input of the U-Net in this thesis is the whole image. The interesting part of the work of Wang et al. [30] is the image-specific fine-tuning. During this part of the training, human feedback is involved. Him and his colleagues have presented an approach to include human feedback in the form of scribbles, which is then being used to further train the model.

This is exactly the area that caused the current problems and catastrophic forgetting. The combination of the model predictions and the oracle corrections. Yet still this is a paper that proposes image specific fine-tuning and therefore is not a perfect fit.

They used a cross-entropy loss just as this thesis has so far but with two adaptations. The oracle-corrected pixels got a higher weight $\boldsymbol{w}$ than the model pixel predictions. Additionally, the predicted pixels the model was most uncertain about were ignored during the loss calculation. This was specifically done by introducing two threshold

values *t_0* and *t_1* marking the uncertainty range in which pixels are ignored. This comes down to the set of pixels being ignored in the loss calculation:

$$U_p = \{i \mid t_0 < p_i < t_1\} \tag{5.1}$$

- U_p: The set of pixels being ignored as the model is too uncertain.
- t_0: The lower threshold of the pixel ignorance area.
- t_1: The higher threshold of the pixel ignorance area.
- p_i: The probability of the foreground class for the current pixel i.

Leading to the weight function:

$$w(i) = \begin{cases} \omega & \text{if } i \in S, \\ 0 & \text{if } i \in U_p \\ 1 & \text{otherwise.} \end{cases} \tag{5.2}$$

- w: The weight being applied to pixel i of the image
- S: The set of oracle-corrected pixels.

This function takes the value of the weighted cross-entropy loss described in chapter 3 and multiplies them by their weight. That happens for every single pixel.

5.3 Grid Search

After making the appropriate changes to the loss function, it was time to find the best hyperparameter values for w, t_0 and t_1. Therefore, a grid search was performed just as in the work of Wang et al. [30].

Setup

The grid search was performed on a subset of 4% of the images covering all the scenarios present in the dataset. They were selected from the unlabeled image set just as the validation set and the training set at the beginning. The model made initial predictions, followed by the oracle's corrections. This was the basis for all 1400 grid search runs. Each run lasted 20 epochs with a learning rate of 10^{-3}. The t_0 was tested with values

ranging from 0.1 to 0.5 and t_1 was tested with values from 0.6 to 0.9 in steps of 0.1. Values of 1.1 to 7.0 in steps of 0.1 were tested for w .

Preliminary Results

The highest dice score reached during the grid search was 0.8878. This is a small increase from the 0.8799 reached by the initial training in epoch 80. A dice score that high was reached in the initial training after 112 epochs. The lowest grid search score was 0.7872 creating a result range of 0.1006. It is also visible that the greater the gap t_0 to t_1, combined with a higher weight, the worse the performance. The grid search was performed twice and averaged. The runs lasted 19.5 minutes on average. Table 5.3 displays the five best and Table 5.4 shows the five worst hyperparameter settings.

Hyperparameters	Dice values
t_0=0.3 t_1=0.6 w=1.9	0.8878
t_0=0.5 t_1=0.6 w=2.8	0.8877
t_0=0.5 t_1=0.6 w=2.4	0.8874
t_0=0.4 t_1=0.6 w=1.9	0.8869
t_0=0.4 t_1=0.6 w=1.8	0.8867

Table 5.3: Best 5 gridsearch results.

Hyperparameters	Dice values
t_0=0.5 t_1=0.9 w=4.0	0.8088
t_0=0.1 t_1=0.9 w=5.2	0.8076
t_0=0.3 t_1=0.9 w=5.6	0.8006
t_0=0.1 t_1=0.9 w=3.0	0.7998
t_0=0.2 t_1=0.7 w=6.8	0.7872

Table 5.4: Worst 5 gridsearch results.

The runs that created the five best and five worst results can be seen in Figure 5.7. When looking at the graph, it becomes clear that the runs with higher weight w start with a worse dice score and have high volatility due to the additionally big t_0 to t_1 range. But even the best runs do not show extraordinary performance. It gives the impression that this approach does not work much better than the first one. This impression is validated by the results in section Multiple cycle steps with optimal settings.

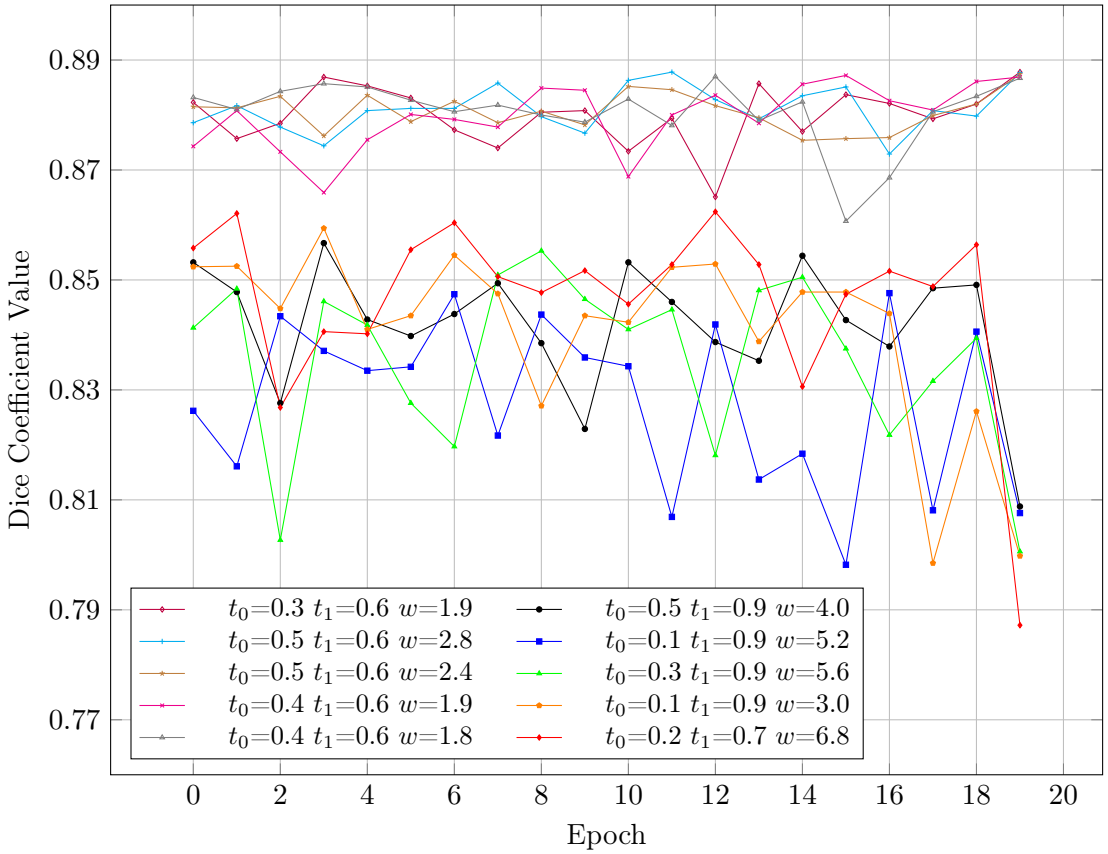

Figure 5.7: Comparison of dice Coefficients across different parameter configurations over 20 epochs during the grid search.

5.4 Cycle Steps II

Building on these hyperparameters and the adapted loss function, the next approach to cycle training was adopted.

Setup

This time, the cycle was tested with the adapted cross-entropy-loss function. The hyperparameters were set to $t_0 = 0.3$, $t_1 = 0.6$ and $w = 1.9$, which worked the best in the grid search. For the learning rate, the same values have been tried. Cycle training was performed on the same 10 most uncertain images with the same corrections as in section Cycle Steps. A comparison of the dice score of the different ratios and learning rates can be seen in Table 5.5. However, Figure 5.7displays the evolution of the different runs

with lr 10^{-4}. This learning rate was chosen for the graph because it achieved the best results.

Preliminary Results

The new approach worked better than the first Grid Search attempt. The results show smaller decreases at the outer ratios and higher more consistent results for the mid-range ratios. As in the experiments before, the ratios from 0.5 to 0.7 performed the best. This is best visible in Figure 5.9. However, the increase is fairly small. The highest score reached is 0.8853 with a learning rate of 10^{-4} and a ratio of 0.5. The lowest score was reached with a learning rate of 10^{-3} and a ratio of 1.0, meaning that training was only performed on the correction images.

		Ratio							
		0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
lr	10^{-3}	0.8618	0.8759	0.8778	0.8794	**0.8812**	0.8741	0.8401	0.8368
	10^{-4}	0.8828	0.8801	**0.8863**	0.8851	0.8854	0.8840	0.8678	0.8705
	10^{-5}	0.8802	0.8783	**0.8801**	0.8789	0.8788	0.8765	0.8744	0.8666

Table 5.5: Dice score results after 20 epochs. The best result per lr is **bold**.

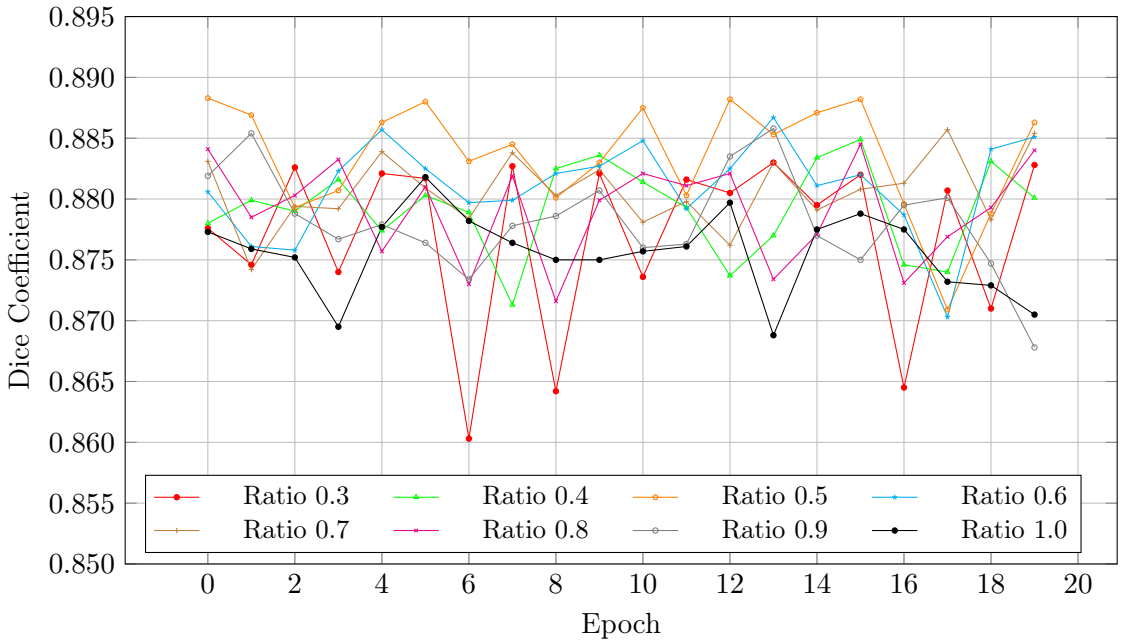

Figure 5.8: Comparison of dice coefficients across different ratio values with learning rate 10^{-4} over 20 epochs.

In the results of this experiment the whisker for the ratio 0.5 stands out even more clearly in being the best ratio than in section Cycle Steps. The statistical values for the boxplot can be found in Table 5.6.

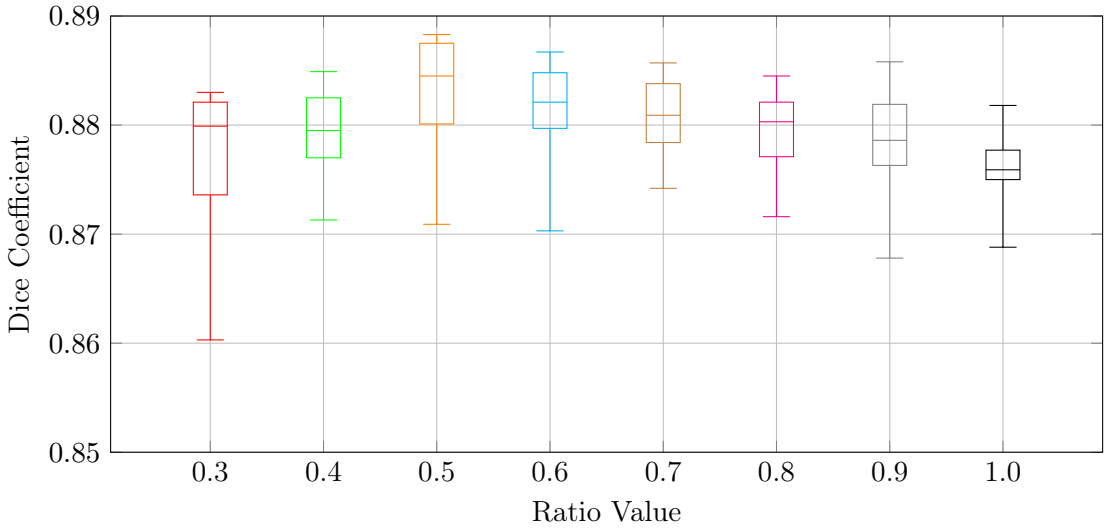

Figure 5.9: Box plot showing the distribution of Dice coefficients for each ratio. The boxes show the interquartile range (IQR) with the median line, while the whiskers extend to the minimum and maximum values.

	Ratio							
	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Upper Whisker	0.8830	0.8849	**0.8883**	0.8867	0.8857	0.8845	0.8858	0.8818
Upper Quartile	0.8821	0.8825	**0.8875**	0.8848	0.8838	0.8821	0.8819	0.8777
Median	0.8799	0.8795	**0.8845**	0.8821	0.8809	0.8803	0.8786	0.8759
Lower Quartile	0.8736	0.8770	**0.8801**	0.8797	0.8784	0.8771	0.8763	0.8750
Lower Whisker	0.8603	0.8713	0.8709	0.8703	**0.8742**	0.8716	0.8678	0.8688

Table 5.6: Boxplot values of Figure 5.9. Highest value per row across the ratios is **bold**.

5.5 Multiple cycle steps with optimal settings

Based on the experiments conducted, a final test with multiple consecutive cycle steps was performed with the optimal settings.

Setup

The best configuration found in the experiments before this is $t_0 = 0.3$, $t_1 = 0.6$ and $w = 1.9$ with a learning rate of 10^{-4} and a hand correction to ground-truth image ratio of 0.5. The cycle was tested for four consecutive iterations.
The following graphs all present consistent lines for the metrics, broken up by orange dotted lines to separate the steps. The different ends and starting points of the steps are connected for a better visualization. These connection lines between two epochs are always penetrated by an orange line.

Preliminary Results

As in the beginning of this chapter, it made sense to look at the dice coefficient combined with the loss and uncertainty metrics to be able to identify trends and lay the foundation for a proper evaluation. This has been done again after the four cycle steps. Starting with the loss, it shows that the validation loss stayed above the training loss and both of them have a downward trend that starts stagnating around the 0.4 mark for the training loss. The final loss value is 0.3986.

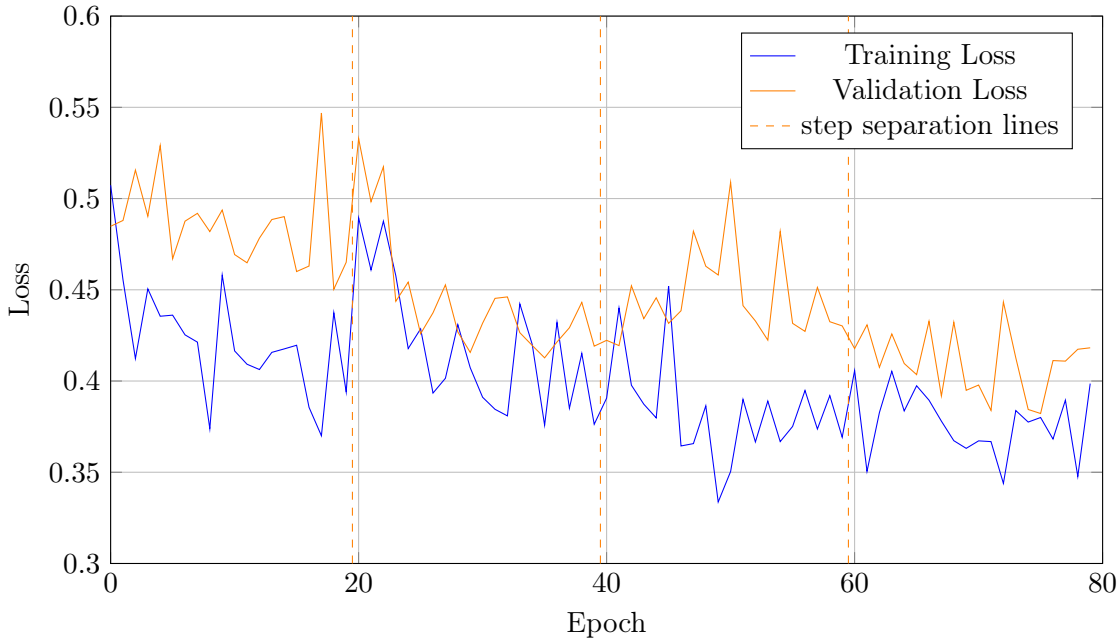

Figure 5.10: Loss and validation loss values across the four training steps.

With that in mind, the minimal upward trend of the dice score fits the loss. After every internal update during the training, the *Bloom* and *Not_Bloom* accuracy reacted in

opposite ways. The dice coefficient reached its peak at epoch 73 with a value of 0.8925. It's final dice score at epoch 80 was 0.8848.

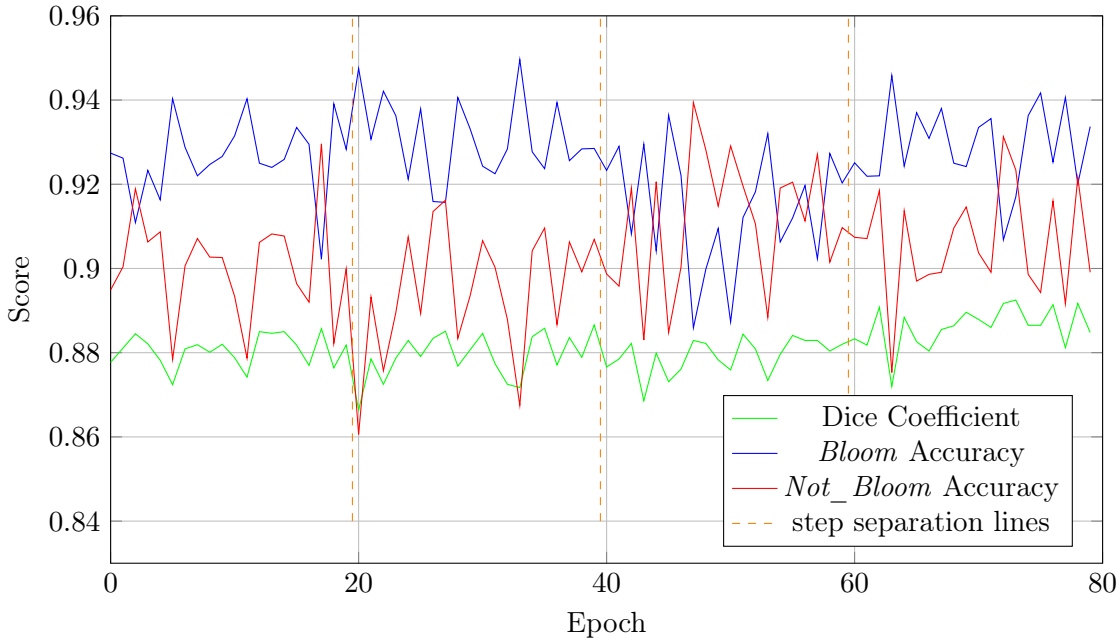

Figure 5.11: Accuracy metrics across the four training steps.

Therefore, it is no surprise that the uncertainty metrics exhibit a downward trend with the average uncertainty stagnating around the 0.1 mark, which was the same for the original training.

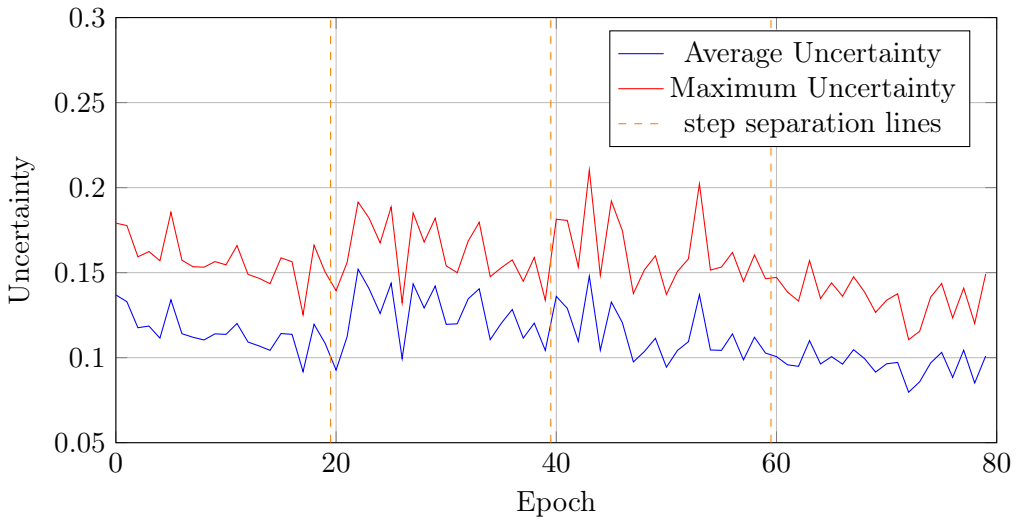

Figure 5.12: Average and maximum uncertainty values across the four training steps.

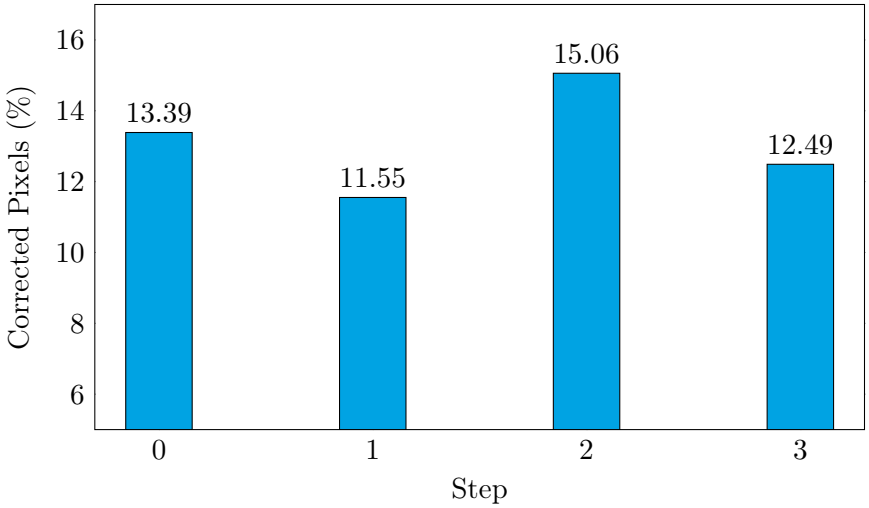

Figure 5.13: Percentage of the training pixels that have been corrected by the oracle. Calculated relative the the pixel amount of the ten correction images.

The last metric is the amount of correction as previously named in chapter Evaluation Metrics. Figure 5.13 displays it as percent. During the four iterations there is no downward trend in sight and the average correction percentage was 13.14%.

Now that all of these results have been gathered, it is time for the evaluation.

6 Evaluation

6.1 Semi-Automated Labeling

This chapter reflects on the experiments conducted. It compares the original U-Net training without any adaptions with the final cycle iteration training done after many experiments. In doing so, it factors in the aspect of time that has been relevant since the beginning of this thesis as it plays a crucial role in evaluating the performance. Additionally, the dice coefficient is the main metric considered during the evaluation.

6.1.1 Quantitative Analysis

The experiments conducted demonstrate that the semi-automated labeling framework, as proposed by Desmond et al., faces significant challenges when adapted to semantic segmentation tasks. The first is the sheer number of labels. Instead of having one label per image, it becomes 2,073,600 labels **per** image, in the case of HD as in this thesis. This introduces a more complex model with more output parameters and more computational complexity. In addition, it also required a totally new way of providing human feedback to the model. Instead of choosing the perfect label out of the list, the human has to correct thousands of pixels without loosing an enormous amount of time. To overcome this issue, the oracle used a labeling tool and only corrected the biggest labeling mistakes of the model in a time of 15 to 20 minutes. This resulted in an average amount of pixel correction of 13.14% per image in the final cycle training.

These challenges were followed by catastrophic forgetting, as described in chapter Experiments.

After overcoming these challenges, these are the dice coefficient values in comparison:

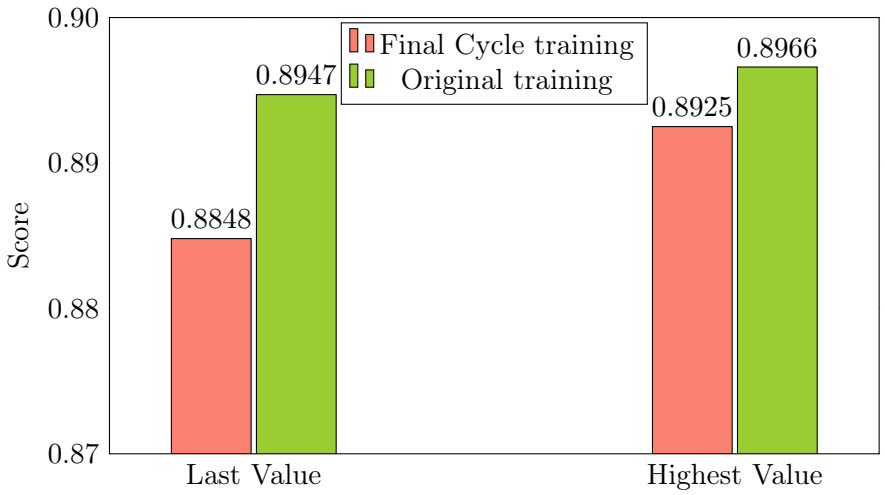

Figure 6.1: Comparison of dice coefficient values between original training and final cycle steps.

When the term original training is used in this chapter, it refers to the 150 training epochs on the ground truth data. And the final cycle training refers to the four cycle steps in section Multiple cycle steps with optimal settings.

The initial model training ran for 150 epochs. The final cycle steps model even trained a total of 160 epochs starting from epoch 80 of the original training. But even after training 10 epochs longer it was neither able to score a higher final score nor was it able to achieve the highest dice score overall during the training. Although the original training with partially labeled data achieved a dice score of 0.8799 and a loss of 0.4325 by epoch 80, subsequent cycle training iterations did not yield substantial improvements.

Looking at the loss metrics, this trend can also be seen. The results of the framework training come close to the original U-Net training, but in three out of four cases, the initial training clearly performed better. Only at the lowest loss do both trainings have almost identical results. The final cycle training clearly does not help the model to reduce the loss, but rather starts to stagnate at higher loss values than the initial training, which shows a worse model.

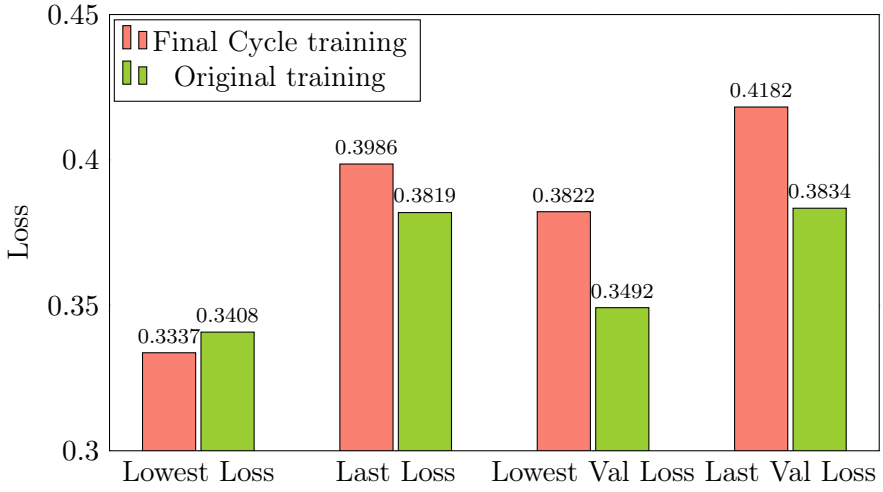

Figure 6.2: Comparison of loss values between original training and final cycle steps.

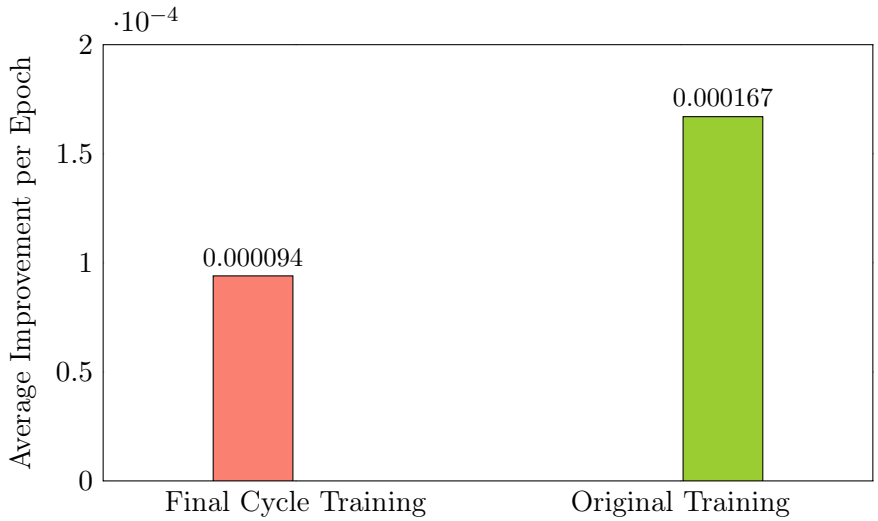

Figure 6.3: Comparison of the average dice coefficient improvement rate per epoch.

The average improvement in Figure 6.3 was calculated over the last 70 epochs of initial training and over the 80 epochs of final cycle training. The method used is linear regression. Looking at the trends of the two different training instances, it becomes clear that the framework has no chance of performing better than the standard U-Net training, as it has a lower dice coefficient improvement rate of 0.000094 compared to 0.000167.

All of this was foreshadowed by the grid search conducted to optimize hyperparameters. As it showcased only marginal gains, the best dice score reached 0.8878. Furthermore, the adapted loss function in Cycle Steps II successfully mitigated catastrophic forgetting with the right hyper-parameters, but primarily stabilized the training without delivering significant dice score increases.

These results were visible not only in the metrics but also when looking at the labels created by the model during the four iterations. As visible in Figure 6.4, the model recognized the shadowy dirt floor as *Bloom* in step one. This was corrected in the example image, but in 21 more images throughout the training as well. Despite all of these corrections, the issue was still persistent in step four. Corrections appear to only have minimal effects during training.

Figure 6.4: Images displaying *Bloom* and *Not_Bloom* correction labels during the final cycle training. The model still recognizes the dirt with shadow on the floor as *Bloom* after that issue was corrected multiple times during the first three steps.

These results indicate that the framework's iterative process fails to substantially improve model performance compared to the baseline.

The time and effort required for the cycle training further illustrate the limitations. Each grid search run took approximately two weeks. A single cycle iteration took six hours,

with 20 out of the 24 total hours being human labor. This is a huge amount of additional time compared to the 2 1/2 hours it took the initial U-Net model to achieve the same performance without human intervention. If the framework were used in this state, it would only waste time and money instead of fulfilling its original goal.

These findings lead to the conclusion that, in its current state, the semi-automated labeling framework does not effectively reduce human effort or accelerate the labeling process for semantic segmentation tasks.

But why is that?

6.1.2 Potential error sources

Several error sources that contributed to the limited success of the framework in its current form have been indicated.

A machine learning model is only as good as the data it is training on. That means the accuracy of the labels as well as the variance in the images and the dataset size. This thesis focused on using as few images as possible while training a model that is able to properly label the images by itself. This is the opposite of what is often the case in machine learning, as more data means a better fine-tuned model that is still able to generalize [32]. Therefore, even the 12% training images might not have been enough to train the model to a point where the hand corrections only have to correct small errors which are present in the whole dataset.

Furthermore, did the dataset present significant visual variability, including differences in lighting, bloom colors, and overlapping objects. The absence of consistent bloom characteristics, coupled with the Full-HD resolution of the images, posed challenges even for human labelers. Blooms have neither a consistent shape nor a constant color. Some of them are white, others pink, some of them have yellow ovarys or red spots. Figure 6.5 displays an image present in the final cycle training together with three zoomed parts. On one of them is clearly a bloom. The oracle was unable to tell which class the other two examples belong to during the experiments. These factors likely reduced the model's ability to learn effectively from the corrections.

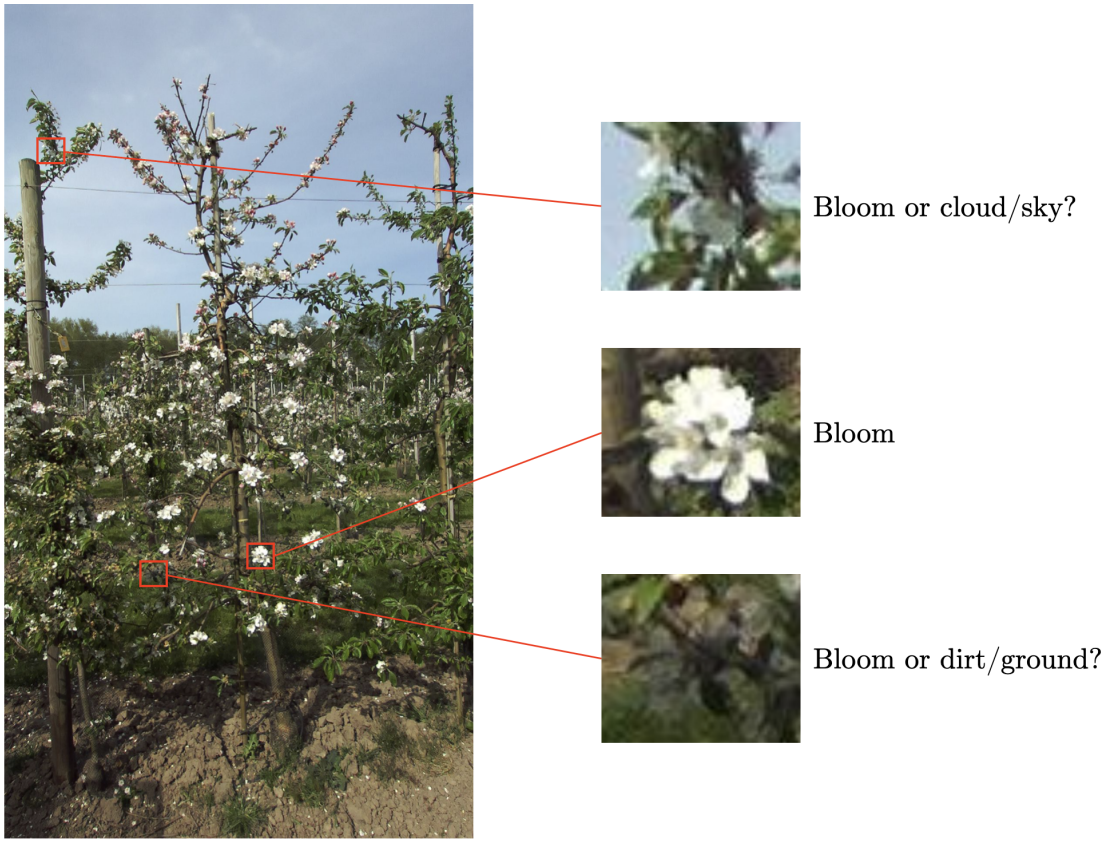

Figure 6.5: Image from the final cycle training with areas that are hard to identify.

Additionally, limiting the selection to the 10 most uncertain images per iteration may have constrained the effectiveness of the corrections. Although the framework aimed to focus on areas of high uncertainty, this narrow scope potentially left broader dataset issues unaddressed. During the cycle, the 10 most uncertain images often looked similar as they proposed the same challenge. Expanding this selection could improve the results, but would increase human effort and time.

The previous sources of error certainly played a role. But the main issue is the feedback given to the model. Before the loss-function adaption, the human feedback was overvalued and always resulted in an image that was almost completely labeled as *Bloom* or *Not_Bloom* in the long run. The model just forgot what it had learned prior and overfitted its weights to the labels of the 10 correction images, resulting in poorer performance in the rest of the images. But after the appropriate changes to overcome catastrophic

forgetting, the oracle feedback was essentially ignored. In this work, it was not possible to find the golden mean between the two extremes to utilize the theoretical advantages of the framework.

Moreover, the semi-supervised learning by Desmond et al. [7] unfortunately does not properly describe its training process. But it seems like he and his colleagues added the oracle-corrected images to the labeled part of the dataset and trained on all labeled images during every iteration. Therefore, they required that the entire image be perfectly labeled, which is not possible in the context of this thesis without diminishing the purpose of saving time. That difference in training seems to be the second big issue, apart from the dataset issues, that explains why the framework did not work for semantic segmentation.

And finally, even if the framework had succeeded in improving the model, the reliance on manual corrections remains a bottleneck. Especially since the CVAT tool, used for correcting labels, exhibited performance issues, particularly when handling large masks. The lag during these corrections significantly slowed the workflow and would impact the overall time efficiency of the framework if it were to produce proper results.

7 Conclusion and future work

7.1 Conclusion

The research carried out explored the adaptation of Desmond et al.'s [7] semi-automated labeling framework to the domain of semantic segmentation. The objective was to evaluate whether this approach could effectively reduce the human effort required for dataset labeling while maintaining or improving the accuracy of the resulting segmentation models.

The implementation involved significant modifications to the original framework, including adjustments for pixel-level annotations, the development of a novel feedback mechanism for human corrections, and the incorporation of metrics tailored to semantic segmentation tasks. Extensive experiments were conducted to assess the performance of the adapted framework in various configurations, using metrics such as the dice coefficient and loss values to quantify results.

The findings reveal that, while the framework successfully addressed some of the challenges inherent to semantic segmentation, it fell short of achieving its primary objective. Specifically, the following observations were made:

1. Limited Performance Improvements: Despite iterative training and the inclusion of human feedback, the final dice scores and loss metrics achieved by the framework were comparable to those of the ground truth training. In several cases, the original model performed better.

2. Time and Effort Considerations: Cycle training required significantly more time and human interaction compared to baseline training that just kept going for 70 more epochs. The effort invested did not result in proportional gains in model performance, undermining the framework's viability as a time-saving solution.

3. Challenges in Semantic Segmentation: The variability in image characteristics, the inherent complexity of pixel-level annotations, and the limitations of the tools used further constrained the effectiveness of the framework.

In conclusion, the adaptation of the semi-automated labeling framework to semantic segmentation proved insufficient in reducing human effort or improving labeling efficiency. The results suggest that the framework, in its current form, is not suitable to address the unique demands of semantic segmentation tasks.

7.2 Future work

7.2.1 Dataset improvements

The dataset used in this study presented challenges due to its relatively low quality, which occasionally led to uncertainty even for human labelers. Future work should prioritize the acquisition of a higher-resolution dataset to enhance the accuracy and reliability of the annotations. In addition, efforts should focus on balancing the dataset more effectively by including a wider range of examples. This could involve capturing more images with varying lighting conditions, diverse bloom colors, and different background scenarios to ensure an even distribution of image types. For example, the current data set includes only three very bright images, highlighting the need for greater diversity to mitigate bias and improve model generalizability.

7.2.2 Adaptations to Labeling Assistance

Future improvements to the labeling assistance process could focus on optimizing the user interface for efficiency. One potential enhancement involves selecting the most challenging pixels for correction and displaying the most probable labels for each. This approach could leverage the idea of the PixelPick [28] Framework, where the highlighted pixel or area is shown in red, and the user navigates and corrects labels using keyboard inputs. This modification would eliminate the reliance on the mouse, significantly accelerating the labeling process and reducing the manual effort. This change would also have the benefit of eliminating the use of the CVAT tool and therefore the lag it brought with it.

7.2.3 General Framework adaptions

A couple ideas for improvement and expansion of the framework can be explored in future work. The first one is increasing uncertainty image selection. Increasing the number of uncertain images selected per cycle iteration could provide the model with a more comprehensive understanding of challenging cases, potentially leading to better performance.

After building the general framework, the issue of catastrophic forgetting heavily influenced this thesis. Therefore, developing and testing alternative strategies to mitigate catastrophic forgetting could enhance the framework's ability to retain previously learned information. This is the most promising but also the most difficult improvement, as it is directly connected with the oracle feedback.

Furthermore, an alternative loss function could generate the proper results. Experimenting with a dice coefficient-based loss function instead of cross-entropy would align the optimization process more closely with the primary evaluation metric and improve segmentation accuracy. This is especially interesting, as the model had trouble improving during the last experiments, seemingly not being able to make the proper adaptions based on the loss.

Bibliography

[1] Mohamed Amgad et al. "NuCLS: A scalable crowdsourcing approach and dataset for nucleus classification and segmentation in breast cancer". In: *GigaScience* 11 (2022), giac037. DOI: 10.1093/gigascience/giac037 (cit. on pp. 2, 3).

[2] Uddhav Bhattarai et al. "Automatic Blossom Detection in Apple Trees using Deep Learning". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 15810–15815. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.216 (cit. on p. 3).

[3] Garrick Brazil, Xi Yin, and Xiaoming Liu. "Illuminating pedestrians via simultaneous detection & segmentation". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 4950–4959. DOI: 10.1109/ICCV.2017.530 (cit. on p. 9).

[4] Samuel Budd, Emma C Robinson, and Bernhard Kainz. "A survey on active learning and human-in-the-loop deep learning for medical image analysis". In: *Medical image analysis* 71 (2021), p. 102062. DOI: 10.48550/arXiv.2108.00941 (cit. on p. 30).

[5] Fabio Cermelli et al. "Modeling the background for incremental learning in semantic segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9233–9242. DOI: 10.48550/arXiv.2002.00718 (cit. on p. 31).

[6] Michael Desmond et al. "Increasing the Speed and Accuracy of Data Labeling Through an AI Assisted Interface". In: *Proceedings of the 26th International Conference on Intelligent User Interfaces*. IUI '21. , College Station, TX, USA, Association for Computing Machinery, 2021, pp. 392–401. ISBN: 9781450380171. DOI: 10.1145/3397481.3450698 (cit. on pp. 2, 3, 7).

[7] Michael Desmond et al. "Semi-Automated Data Labeling". In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Ed. by Hugo Jair Escalante and Katja Hofmann. Vol. 133. Proceedings of Machine Learning Research. PMLR, June 2021, pp. 156–169. URL: https://proceedings.mlr.press/v133/desmond21a.html (cit. on pp. 1–3, 7, 15, 17, 19, 22, 46, 47).

[8] Arthur Douillard et al. "PLOP: Learning without Forgetting for Continual Semantic Segmentation". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 4039–4049. DOI: 10.1109/CVPR46437.2021.00403 (cit. on p. 31).

[9] Esteburg. *Website of Esteburg research facility*. URL: https://www.esteburg.de/research/mid_42991.html (cit. on p. 13).

[10] Teodor Fredriksson, Jan Bosch, and Helena Holmström Olsson. "Machine Learning Models for Automatic Labeling: A Systematic Literature Review." In: *ICSOFT* (2020), pp. 552–561. DOI: 10.5220/0009972705520561 (cit. on pp. 1, 5).

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (cit. on p. 14).

[12] Google. *Google Colab*. URL: https://colab.google/ (cit. on p. 24).

[13] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A brief survey on semantic segmentation with deep learning". In: *Neurocomputing* 406 (2020), pp. 302–321. DOI: 10.1016/j.neucom.2019.11.118 (cit. on pp. 1, 9).

[14] Moritz Hentzschel. *Experteninterview über den Obstanbau*. personal communication. 17. April, Auf dem Gelände der Esteburg. 2024 (cit. on p. 1).

[15] David Joon Ho et al. "Deep Interactive Learning-based ovarian cancer segmentation of H&E-stained whole slide images to study morphological patterns of BRCA mutation". In: *Journal of Pathology Informatics* 14 (2023), p. 100160. DOI: 10.1016/j.jpi.2022.100160 (cit. on p. 3).

[16] David Joon Ho et al. "Deep interactive learning: an efficient labeling approach for deep learning-based osteosarcoma treatment response assessment". In: *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part V 23*. Springer. 2020, pp. 540–549. DOI: 10.48550/arXiv.2007.01383 (cit. on p. 3).

[17] Andreas Holzinger. "Interactive machine learning for health informatics: when do we need the human-in-the-loop?" In: *Brain informatics* 3.2 (2016), pp. 119–131. DOI: 10.1007/s40708-016-0042-6 (cit. on pp. 6, 7).

[18] Tobias Kalb and Jürgen Beyerer. "Causes of Catastrophic Forgetting in Class-Incremental Semantic Segmentation". In: *Proceedings of the Asian Conference on Computer Vision (ACCV)*. Dec. 2022, pp. 56–73. DOI: 10.48550/arXiv.2209.08010 (cit. on p. 31).

[19] Ronald Kemker et al. "Measuring catastrophic forgetting in neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018. DOI: 10.48550/arXiv.1708.02072 (cit. on p. 30).

[20] *Keras documentation*. URL: https://keras.io/ (cit. on p. 24).

[21] Baojun Li et al. "Real-time object detection and semantic segmentation for autonomous driving". In: *MIPPR 2017: Automatic Target Recognition and Navigation*. Vol. 10608. SPIE. 2018, pp. 167–174. DOI: 10.1117/12.2288713 (cit. on p. 9).

[22] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165. DOI: 10.1016/S0079-7421(08)60536-8 (cit. on p. 30).

[23] Samreen Naeem et al. "An Unsupervised Machine Learning Algorithms: Comprehensive Review". In: *IJCDS Journal* 13 (Apr. 2023), pp. 911–921. DOI: 10.12785/ijcds/130172 (cit. on p. 5).

[24] Vladimir Nasteski. "An overview of the supervised machine learning methods". In: *Horizons. b* 4.51-62 (2017), p. 56. URL: https://www.researchgate.net/publication/328146111_An_overview_of_the_supervised_machine_learning_methods (cit. on p. 5).

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/arXiv.1505.04597. arXiv: 1505.04597 [cs.CV] (cit. on pp. 11, 12, 17).

[26] Bhavani Sambaturu et al. "Efficient and generic interactive segmentation framework to correct mispredictions during clinical evaluation of medical images". In: *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part II 24*. Springer. 2021, pp. 625–635. DOI: 10.48550/arXiv.2108.02996 (cit. on p. 3).

[27] Burr Settles. *Active learning literature survey*. 2009. URL: https://minds.wisconsin.edu/handle/1793/60660 (cit. on p. 6).

[28] Gyungin Shin, Weidi Xie, and Samuel Albanie. *All you need are a few pixels: semantic segmentation with PixelPick*. 2021. DOI: 10.1109/ICCVW54120.2021.00194. eprint: 2104.06394 (cit. on p. 48).

[29] *TensorFlow documentation*. URL: https://www.tensorflow.org/ (cit. on p. 24).

[30] Guotai Wang et al. "Interactive medical image segmentation using deep learning with image-specific fine tuning". In: *IEEE transactions on medical imaging* 37.7 (2018), pp. 1562–1573. DOI: 10.48550/arXiv.1710.04043 (cit. on pp. 31, 32).

[31] Xingjiao Wu et al. "A survey of human-in-the-loop for machine learning". In: *Future Generation Computer Systems* 135 (2022), pp. 364–381. DOI: 10.48550/arXiv.2108.00941 (cit. on p. 30).

[32] Lina Zhou et al. "Machine learning on big data: Opportunities and challenges". In: *Neurocomputing* 237 (2017), pp. 350–361. DOI: 10.1016/j.neucom.2017.01.026 (cit. on p. 44).

[33] Jiayuan Zhu and Junde Wu. "MedUHIP: Towards Human-In-the-Loop Medical Segmentation". In: *arXiv preprint arXiv:2408.01620* (2024). DOI: 10.48550/arXiv.2408.01620 (cit. on p. 31).

[34] Xiaojin Jerry Zhu. *Semi-supervised learning literature survey*. 2005. URL: https://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf (cit. on p. 6).

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original