Automating Bacterial Colony Segmentation with Deep Learning

Bachelor's thesis
in partial fulfilment of the requirements for the degree of
Bachelor of Science

University of Münster
Computer Science Department

Supervisor:
Prof. Dr. Xiaoyi Jiang

First assessor:
Prof. Dr. Xiaoyi Jiang

Second assessor:
Prof. Dr. Fritz Titgemeyer, FH Münster

Submitted by:
Leonhard Driesch

Münster, June 2024

Automating Bacterial Colony Segmentation with Deep Learning

Automatische Segmentierung von Bakterienkolonien durch Deep Learning

Abstract

Analysis of bacterial colony growth is an integral part of microbiology, which currently requires a lot of tedious work. Deep learning can help with detecting colonies to minimize the effort needed to count them. As part of this thesis, an efficient labeling technique is developed to quickly annotate images with instance segmentation masks. It is used to create a diverse dataset of 200 labeled images. Several object detection models are trained on this dataset with different configurations and data augmentations. Performance evaluation shows great potential for the use of object detection models in counting colonies and that performance can be greatly improved by incorporating various pre-training mechanisms. Everything is developed with ease-of-use in mind and the dataset can be grown further to improve detection performance in the future.

Acknowledgements

I'd like to thank Sebastian Fischer from the Department of Food, Nutrition and Facilities at FH Münster. He showed me around at the laboratory and explained the purpose of this project and his long term vision for it clearly. We worked closely together during creation of the dataset and he was always helpful and swift to reply.

The quick and constructive support from the Data Processing Centre at FH Münster, especially Sebastian Rennert, was also of great help. While using the campus cluster for training, they helped to sort out any hiccups with hard- or software swiftly and made my work with the computing resources a lot easier.

Contents

1 Introduction

Growing and analyzing bacterial colonies on Petri dishes is a core part of modern microbiology. It is used to quantify microbial populations present in a given sample, such as Salmonella in food or E. coli in water.

The process includes spreading a small sample onto nutrient-rich gel inside a petri dish, such as MRS agar, and then leaving the bacteria to grow for a couple of days. After that time, visible colonies of bacteria will show. They can then be counted to infer the approximate amount of bacteria present in the initial sample.

Analyzing these petri dishes is however a time consuming process. The number of colonies on a plate ranges anywhere from zero to hundreds of colonies, in some cases even thousands. Currently at FH Münster and many other microbiology labs, they are mostly counted manually, which often takes multiple minutes per plate.

The goal of this thesis is to prove that deep learning can help with the analysis of petri dishes by detecting colonies on images of plates. It also lays the foundation for future work to improve upon the techniques proposed here to fully automate the process. Since labeling images by hand is slow and expensive, this work also explores how to get the best performance for a limited dataset size.

The objective is purposely set on *detecting* colonies. This means the model should be able to locate colonies and their bounding boxes on the plate. This opens up possibilities for analyzing individual colonies, such as for classification. Therefore methods that are only capable of counting colonies, but not reason about their location and size on the image, were not taken into consideration here.

2 Literature Review

There are several different works on using deep learning to assist in bacterial colony counting and detection.

Ferrari *et al.* [1] combine traditional algorithm based colony segmentation with a convolutional neural network to count the number of colonies contained in the segmented confluent agglomerates. The CNN is trained on 17,000 images of agglomerates from their manually labeled dataset and classifies each image into having 1-6 colonies or being an outlier. It boasts an accuracy of 92.8% and beats other traditional segmentation approaches like the watershed transform.

Makrai *et al.* [2] introduce a bacterial colony detection dataset. It comprises 369 images with a total of 56,865 colonies. The focus is on diversity, with 24 different species being included in the images. Three different mobile phones are used to capture the images, on a mix of light and dark backgrounds. The images are labeled and curated by experts with a PhD in bacteriology using the COCO Annotator tool.

Majchrowska *et al.* [3] introduce the Annotated Germs for Automated Recognition dataset, AGAR for short. AGAR is a the first publicly available large-scale dataset for microbial colony detection, featuring a total of 18,000 images from 5 different microorganisms and various configurations. The images show full petri dishes in contrast to the images from Ferrari *et al.* [1] that show only image segments. This allows them to train end-to-end object detection models on the dataset, which don't rely on feature extraction from computer vision algorithms.

They train Faster R-CNN and Cascade R-CNN models on the dataset, while trying out different backbones. For the high-resolution subset of 6,990 images, Cascade R-CNN with an HRNet backbone performs best, if only by a small margin, with a mean Average Precision of 52.0% (mAP@.5:.95:.05) and a mean absolute error of 4.31 for the colony counting task.

Whipp *et al.* [4] use a subset of the AGAR dataset, specifically the 1,240 images containing the S. Aureus bacteria, to train and evaluate models from the YOLO family. YOLO is a family of object detection models that are single-shot detectors, i.e. they treat object detection as a single-pass regression problem. From the YOLOv5 family, 4 different model sizes are evaluated. Results show that performance only differs by a small amount for the different sizes. Different resolutions are

also evaluated, with the highest (1920x1920 px) offering slightly better detection for a larger cost in efficiency. The best results obtained were an mAP@.5:.95:.05 of 66.1% and a recall of 97.7% at an IoU of 0.5. Counting metrics are not reported.

Majchrowska *et al.* [5] compare different object detection architectures for both efficiency and performance on the AGAR dataset. They evaluate two-stage (Faster R-CNN, Cascade R-CNN, Libra R-CNN, CBNet v2), one-stage (YOLOv4, EfficientDet-D2) and transformer architectures (Deformable DETR, Faster R-CNN with XCiT backbone), all with pre-trained backbones.

While detection performance does not differ much between architectures, YOLOv4 performs the best with an mAP@.5:.95:.05 of 52.9% and a mean absolute counting error of 4.18. It also has the best inference efficiency, being at least twice as fast compared to the other architectures. Transformer architectures perform generally worse than the other detectors.

Yang *et al.* [6] also work with the AGAR dataset, but using a subset of only 100 images, which are augmented via style transfer to obtain more than 4,000 image patches of 512x512 px resolution. They then use a novel backbone, the Shifted Window (Swin) Transformer, in combination with Cascade R-CNN to get an improved performance of 61.4% mAP@.5:.95:.05, 2.2% more than Cascade R-CNN with HRNet backbone.

They also try a pre-trained YOLOv8x model on their created dataset, which reaches a 76.7% mAP showing very good detection performance on the dataset.

Zhang *et al.* [7] propose a novel data augmentation approach, which they use to generate a dataset of 970 petri dish images from a base of only 5 annotated images. They use a Random Cover Target Algorithm to mask colonies out from the images by filling the area with pixels of the background color. They train various Tiny YOLOv3 and Improved YOLOv3 models on this dataset, optimized for deployment on an edge device.

After evaluation on a test set of 60 independent images, the models obtain 98.8% precision and a recall of 98.5%. They show that a small number of labeled images can be sufficient to train a reliable detection system for images of a specific style.

Pawłowski *et al.* [8] use deep learning style transfer based on VGG19 to create a synthetic dataset from 100 base images. To generate new images, segmented colonies are placed on images of empty dishes. Then the stylizing adjusts these patchwork images to match the style of real data.

They compare the performance of Faster R-CNN and Cascade R-CNN for training on the AGAR high-resolution dataset vs. their synthetic dataset. Without the stylization process, only a 20.1% mAP@.5:.95:.05 is reached with significant counting error. Stylization brings the mAP to 41.5% (compared to 52% for the 70x larger AGAR dataset) and mean absolute counting error to 4.49

(compared to 4.31 for AGAR). They show that performance of larger datasets can almost be matched by synthesizing a dataset with neural style transfer from a small number of images.

3 Methodology

3.1 Data Collection

In this work a total of three datasets are used. The dataset used in all model training (referred to as the photobox dataset) is a dataset created in collaboration with the microbiology lab of FH Münster. It contains 529 images of unique agar plates, of which 200 are labeled with instance segmentation masks (see Figure 3.5). The images were taken in a photo apparatus (see Figure 3.1) which is constructed to capture images of agar plates under constant conditions. The two-piece design with a light box and a diffuser minimizes reflections and glare on its photos. [9] The camera framing the petri dish, a Raspberry Pi High Quality Camera, [10] takes images of 3040x3040 px (9.24 megapixels) in resolution.

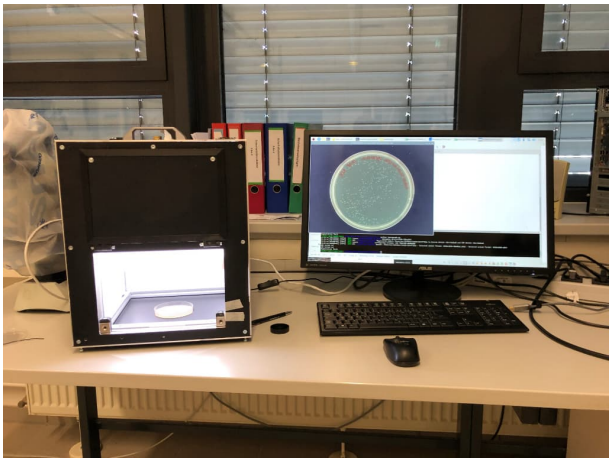


Figure 3.1: Photo apparatus used to take images for the photobox dataset [9]

The dataset contains plate images with different types of bacteria on different types of Agar. The colonies vary in shape, size, color and count. Some bacteria have a circular shape while others have mycelial growth (see top-right of Figure 3.2). Some of the bacterial colonies have grown together, which makes it more difficult to segment them and count them accurately (in the center of top-middle in Figure 3.2). Example images and the distribution of different types of images can be seen in Figure 3.2 and Figure 3.3.

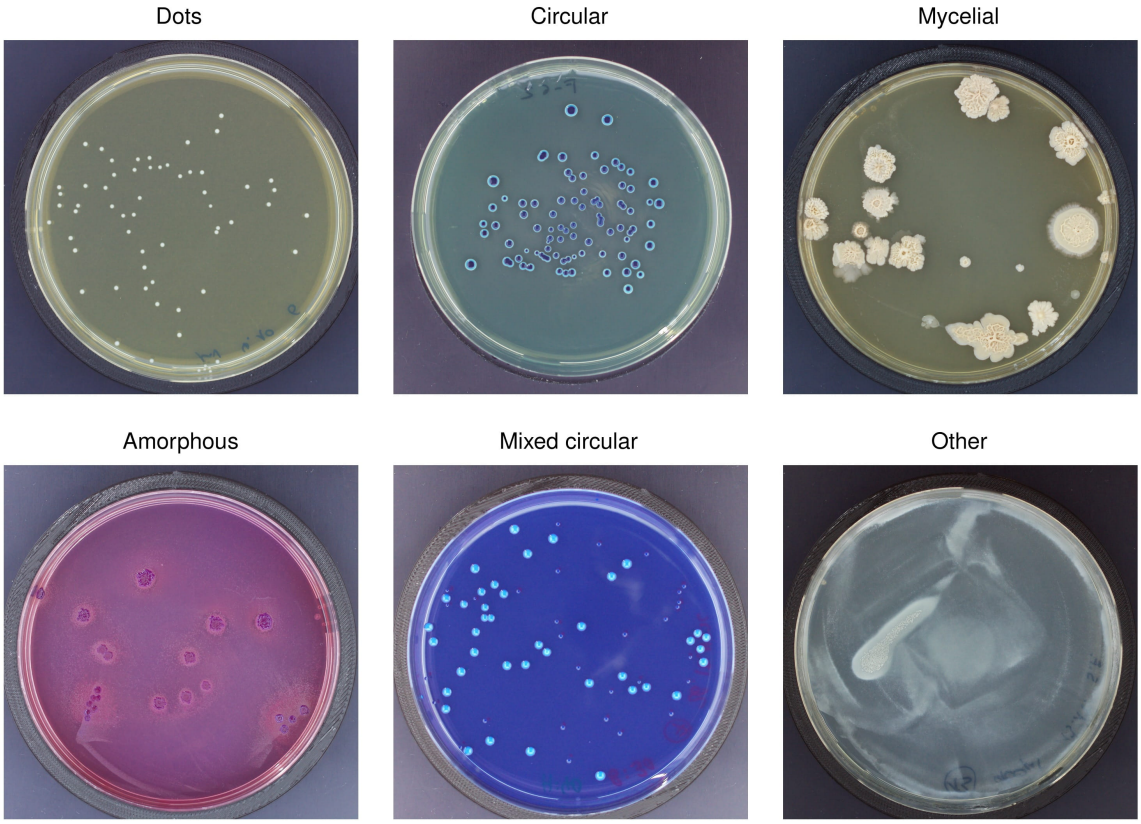


Figure 3.2: Overview of different types of images present in the photobox dataset

Since the photobox dataset is of limited size, two other datasets were also used during model development. The first one is Microsoft COCO (Common Objects in Context), a large-scale dataset for object detection. It contains more than 200,000 annotated images in which 91 different object types are labeled with class labels, bounding boxes and segmentation masks. The categories include bench, suitcase, plate and sandwich and many more. [11] See Figure 3.4 for example images and masks.

Fine-tuning an object detection network pre-trained on the COCO dataset is a widely used technique to enhance model performance for small datasets, especially in medicine. [12] [13]

The second auxiliary dataset is the Annotated Germs for Automated Recognition dataset, AGAR for short. It contains 18,000 photos of five different microorganisms under different lighting conditions and in different resolutions. [3] This dataset is highly related to the domain of this thesis.

Only the high-resolution subset of the AGAR dataset was used. There are 5242 train images and 1748 images for validation in this subset, each with a resolution of approximately 4000x4000 px (16 megapixels).

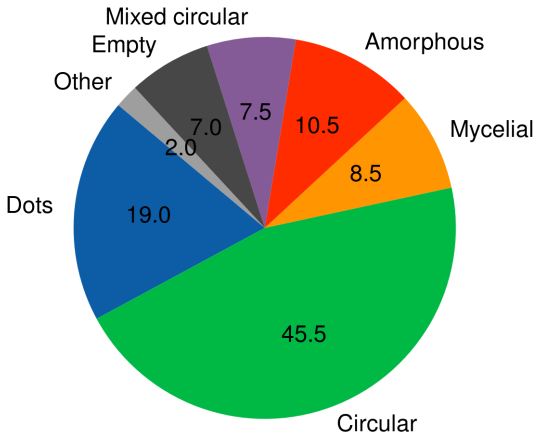


Figure 3.3: Distribution of different types of images present in the photobox dataset

Figure 3.4: Example images with their corresponding masks overlayed [14]

3.2 Data Labeling

Instance segmentation masks are selected as the type of label that is most useful for the photobox dataset. An instance segmentation mask here is a PNG image that has a black background with colored spots on it (see Figure 3.5). The colors are randomly chosen so no two colors are ever the same on a mask image and represent the individual instances (colonies).

Instance segmentation masks capture a lot of detail and can be trivially transformed into other types of datasets like bounding boxes, semantic segmentation masks or keypoints. This leaves room for experimentation, since the model architecture was not decided on yet.

To label the images of the photobox dataset, a web application was built as part of this thesis. The app (seen in Figure 3.7) allows bulk upload of images on the left hand side. When an image is selected in the image browser on the left, it is analyzed on the server. Then accurate masks for each colony can be automatically created by clicking on the colony. By repeatedly clicking on every colony in the image, the full instance segmentation mask is created.

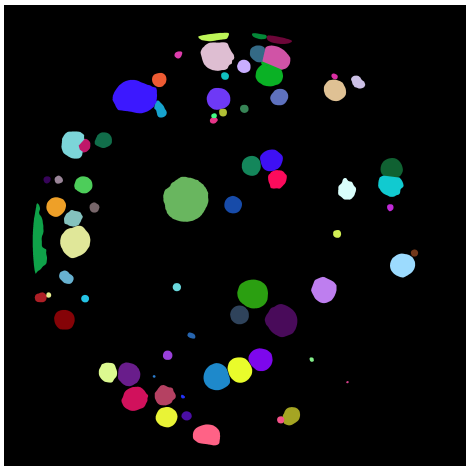


Figure 3.5: Example of an instance segmentation mask

The automatic mask creation is built around the Segment Anything Model, SAM for short. SAM is a zero-shot image segmentation model for creating accurate segmentation masks from various prompts (see Figure 3.6). [15] It can mask objects from point locations, create a mask for an object contained in a bounding box or refine a rough mask to be more precise. The prompt for creating a mask when clicking on a colony in the labeling app is a point-prompt, which can be thought of as "please mask the object I am pointing at".

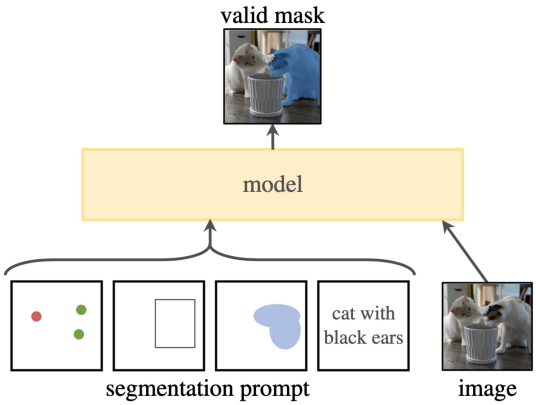


Figure 3.6: Tasks SAM can perform [15]

This works well for most cases, however SAM can have difficulties fully masking a colony if it has an irregular shape or is large in size (see top-right of Figure 3.2). Normally, a new region is created for every click, which correlates to a new instance (i.e. not the same colony). But when multiple clicks are needed to capture a single colony because SAM fails to mask the full object, holding shift while clicking will add the newly masked area to the previous region (i.e. same colony).

Colonies that grew together are often collectively masked by SAM, but have to be accounted for

Figure 3.7: Screenshot of the labeling app

independently. To separate this region it is possible to click inside of an existing mask to split it into two. Every pixel is then assigned to the mask of the click point it is closest to by using a k-d tree.

Other conveniences such as an undo button and zooming are also available. This setup of features allows every image in the photobox dataset to be labeled accurately. Manual correction of the masks, for example through photo editing software, is not needed.

The app is also deployed on a server at FH Münster, so their microbiology team can continue to grow their dataset independently.

Figure 3.8 shows statistics on the distribution of the created photobox dataset. The first histogram shows how many colonies are present on the images. It highlights a bias towards images with lower numbers of colonies, images with more than 200 colonies are rarely present.

The second histogram shows the size of colonies in relation to their frequency in the dataset. It demonstrates that most of the colonies are of similar, smaller size, with a few outliers that are much larger, up to roughly 800x800 px. Such large colonies can be seen in the top-right image of Figure 3.2.

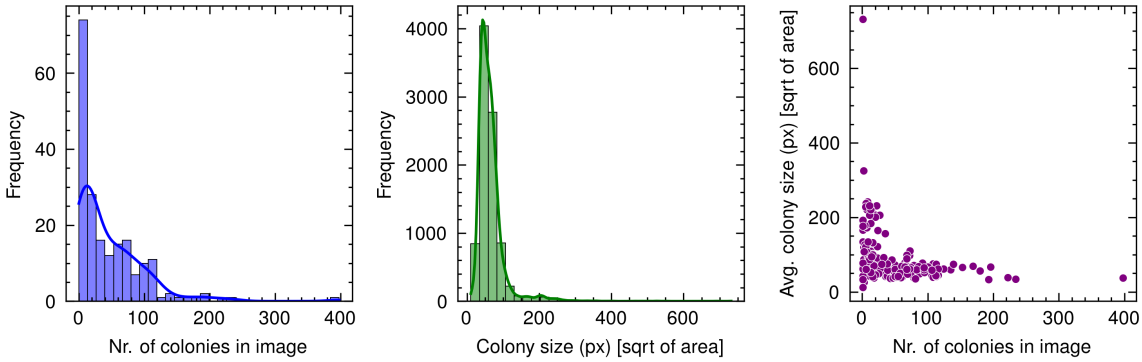


Figure 3.8: Distribution of the labeled data

The scatter plot on the right of Figure 3.8 shows the relation of the number of colonies in an image and the average size of the colonies in the image. It reveals a trend where images with fewer colonies can have colonies that are larger in size, but colonies on images with high counts are generally smaller.

3.3 Data Preprocessing

The photobox dataset is split into train, validation and test sets with a 50%/25%/25% split. With the 200 labeled images that leaves 100 for training, 50 for validation and 50 for testing (performance evaluation).

The annotated instance segmentation masks are converted into a bounding box style dataset, similar to the AGAR dataset. Since each colony has a different color in the mask, it is easy to draw a bounding box around every region of unique color in the image. The pseudo code for this algorithm is outlined in 1.

Algorithm 1 Convert instance segmentation masks to bounding boxes

```
for each mask in masks do
    unique_colors ← get_unique_colors(mask)
    for each color in unique_colors do
        x1 ← find_first_index(mask, color, 0)
        y1 ← find_first_index(mask, color, 1)
        x2 ← find_last_index(mask, color, 0)
        y2 ← find_last_index(mask, color, 1)
        save_bounding_box(mask_id(mask), x1, y1, x2, y2)
    end for
end for
```

For computer vision models it is common to use pre-processing image transformations, called augmentations, to artificially increase the dataset size. Here, several data augmentations are experimented with:

1. Horizontal flip (50% probability)
2. Vertical flip (50% probability)
3. Random rotation of 0, 90, 180 or 270 degrees
4. Shift brightness and contrast by up to 10%

A newly developed augmentation was also experimented with, inspired by the Random Cover Targets Algorithm proposed in Zhang *et al.* [7]. It is based around the idea of covering an area of the image that contains colonies and inpainting this area with the plate background. The new inpainted area now does not contain colonies and the corresponding mask can be filled with black in the inpaint area. This augmentation is referred to as the inpainting augmentation.

First, suitable areas for covering have to be found. The algorithm for finding these areas works by randomly positioning a circle on the image. It then checks if

1. the circle is fully inside the area where colonies can occur (inside the plate, and not on the edge or in the corners of the image)
2. the circle contains colonies (this is done by checking if there are non-black pixels in the corresponding mask)
3. the circle does not "cut off" any colonies, i.e. a colony is half inside the circle and half outside
4. the circle does not overlap more than 20% with another circle that has been found

The algorithm continuously tries to find such circles in an image. If no suitable circle was found within 1,000 tries, the circle finding program goes on to the next image. In total, 1,757 circles were found for the 100 images in the train set, on average 17.57 per image.

This brute force approach is fast enough, with somewhat optimized code it takes about 1 hour to find suitable circles for all images. It also finds enough circles for most images in its 1,000 tries and is statistically random, so each circle is as likely as any other to be generated (there is e.g. no bias towards circles that are closer to the center).

In the next step, for each image, each proposed circle would be covered individually. Then an image generation model was used to fill the covered area, a process known as inpainting. The goal of this step is to fill the area with background texture that matches the rest of the image.

The specific model used for inpainting is Kandinsky 2.2, specifically the inpaint variant published on Hugging Face. [16] Kandinsky is a generative text-to-image model based around CLIP as the text and image encoder and latent diffusion. [17] It can create realistic images from a text prompt, similar to DALL·E 2 [18] and Stable Diffusion [19].

The text-to-image generation is guided by a positive prompt, which incentivizes certain elements to appear in the final image, and a negative prompt which specifies which elements should not be in the final image. When inpainting, the text-to-image model can only "control" the pixels included in the inpainting mask, so the context of the image is also important in guiding the generation.

The specific text prompts used for inpainting here are *a clean petri dish with nutrient agar medium under laboratory lighting* for the positive and *bacteria colony, bacterial colonies, bacteria, circles, dots, microbial spots* for the negative prompt. These prompts were constructed through CLIP interrogation and experimentation. CLIP interrogation takes an image and gives a text prompt that when fed to a text-to-image model is likely to produce a similar image.

For the inpainting process the images are scaled down to 768x768 px, because that is the recommended image resolution for Kadinsky 2.2. They are upscaled to the original resolution of the images in the photobox dataset (3040x3040 px) using bicubic interpolation. The algorithmic upscaling has a blurring effect, but this can be neglected here since the area of interest from the generated image only contains background texture and no objects whose details would suffer from blurring.

The image generation often produces undesirable outcomes. A desirable inpainting result produces a texture that blends nicely into the background of the plate. However often the model will generate additional colonies into the image or even produce artifacts (see Figure 3.9). Therefore each generated image has to be checked, which takes about 2.5 hours to sort manually for the 1,757 generated images.

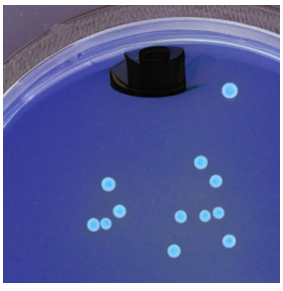
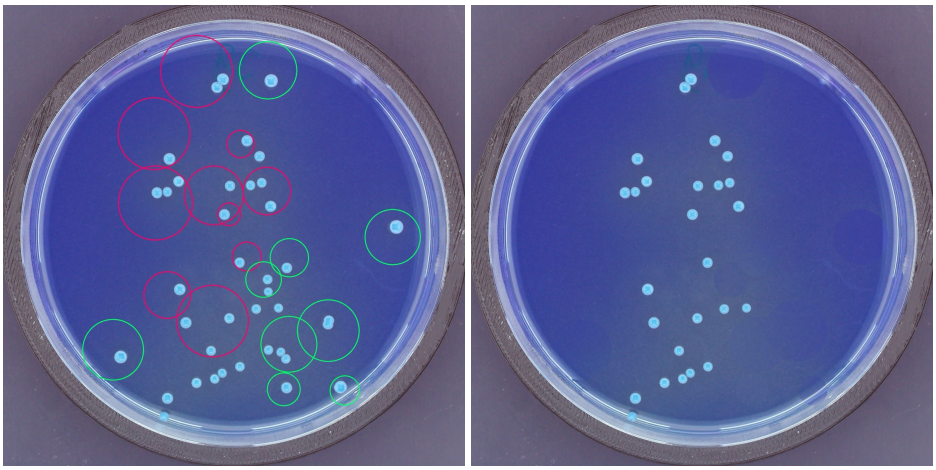


Figure 3.9: Example of an artifact generated during the inpainting process

Sorting out the images with undesirable outcomes leaves 934 images which can be used for augmenting the dataset. These can now be used to cover up the colonies in an original image, by



(a) Potential inpainting circles. Green ones can be kept, red ones are undesirable.

(b) All successful inpainted circles are copied onto an original image.

Figure 3.10: Patch augmentation pipeline for an example image

copying in the inpainted area from a generated image. An example of this process can be seen in Figure 3.10.

During training, when an image is loaded for a forward pass through the model, each patch is loaded with a chance of 50%. When a patch is loaded, the bounding boxes contained in the circle area are removed from the labels. Loading a patch can be thought off as toggling a layer in photo editing software.

3.4 Model Development

As explained in the introduction, the goal of this work is to localize all colonies on the plate individually and being able to count them.

An object detection model is best suited for this job because it is able to localize bounds of each colony, which a regression based (directly predicting the number of colonies from an image) or density map based counting model could not. It is also able to distinguish individual colonies, even if they overlap, which a semantic segmentation model (is this pixel a colony or not?) could not. Instance segmentation models, which create output masks just like the labels of the photobox dataset (see Figure 3.5) would also work. But object detection models are easier to train and the mask output is not needed.

From the family of object detection models, Faster R-CNN was selected as the architecture for this project. Faster R-CNN is a two-stage object detection model. A region proposal network and a

classifier work on feature maps created by a backbone model (here ResNet-50). The region proposal network creates proposals for regions that could contain an object of interest, the classifier then decides if the object belongs to the background or any of the specified classes. [20] Figure 3.11 shows the architecture in a diagram. In this work, there are only two classes, one for background (not of interest) and one for colony. This could however be easily expanded to classify colonies by species during the detection process.

Faster R-CNN has a readily available PyTorch implementation [21] which includes pre-trained weights for the COCO dataset based on the training approach from Li *et al.* [22]. As shown in Majchrowska *et al.* [5], the difference in performance between state-of-the-art object detection models on the task of microbial colony detection is minimal. So Faster R-CNN is a good choice since the effort of implementing a training script [23] and using pre-trained weights [24] is much reduced.

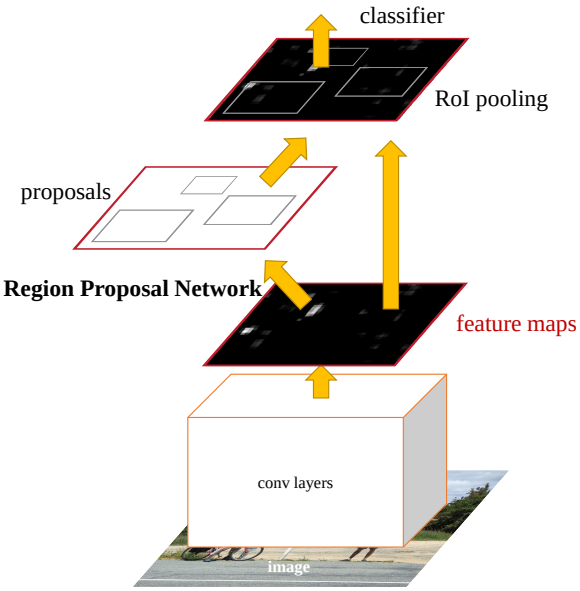


Figure 3.11: Overview of the Faster R-CNN architecture [20]

The model is optimized by stochastic gradient descent, using a momentum of 0.9 and a weight decay of 0.0005. The learning rate is initialized to 0.005 for a batch size of 8. The learning rate was scheduled with a ReduceLROnPlateau strategy, which would cut the learning rate by 90% after no improvements have been made to validation loss for 10 epochs. After the learning rate has been reduced two times, training stops. This eliminates n_{epochs} as a hyperparameter that needs tuning and makes experiments easier.

All training runs are conducted on single nodes from the Campus Cluster of FH Münster equipped with Tesla V100 16GB graphics cards.

Evaluation Metrics

Before discussing the different experiments, the metrics used for evaluating an experiment have to be laid out. Accurately counting the colonies is the primary focus of this work and precisely locating them is of secondary interest, the main metrics used for evaluating are comparing the number of detected colonies vs the ground truth. They are adapted from Majchrowska *et al.* [3].

The symmetric mean absolute percentage error metric, sMAPE for short, measures relative counting error in relation to the amount of colonies on the plate.

$$\text{sMAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|x_i - \tilde{x}_i|}{|x_i + \tilde{x}_i|}$$

Here N is the number of images, x_i is the number of colonies on the image and $\tilde{x}_i$ is the colonies in a prediction. For the case of $x_i = \tilde{x}_i = 0$, meaning there are no colonies on the image and the predicted colony count is 0, nothing is added to the sum as the prediction is correct.

The mean absolute error, MAE for short, is the mean over all absolute counting errors for predictions on images of the test set, so

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{n}|x_i - \tilde{x}_i|$$

As an object detection metric, mean average precision, mAP for short, was used. The average precision is the area under the precision-recall-curve. To draw a precision-recall-curve, these values have to be calculated:

True positives (TP) are predictions the model has made, which are correct (match a ground truth bounding box).

False positives (FP) are predictions the model has made, which are incorrect (do not match a ground truth bounding box, are duplicates, etc.).

Precision is $\frac{TP}{TP+FP}$, i.e. how much of the predictions are actually correct.

False negatives (FN) are cases where a prediction should have been made, but is missing.

Recall is $\frac{TP}{TP+FN}$, i.e. how much of what should have been found has been found.

The precision recall curve is drawn by going through each predicted bounding box sorted by confidence score, and calculating precision and recall for the subset of bounding boxes that have been "visited". When plotting these values onto a graph with precision on Y and recall on X axis and connecting all of them, the area under the curve is the average precision.

Deciding between a true and a false positives for an object detection model requires setting an IoU threshold. The Intersection over Union, short IoU, is the area of the overlap between two boxes divided by the area of the union of both boxes. The IoU threshold is a fixed number that determines whether a detected box is correct.

Mean average precision is the mean of the average precision at different IoU thresholds, here it is every IoU threshold between 0.5 and 0.95 with steps of 0.05. This is also the primary metric for comparing performance of object detection models on the COCO dataset. [25] It's score is indicative of how well a model can detect the objects it is supposed to detect, and how precisely it can locate them.

Experiments

For the experiments, evaluation metrics are calculated for the validation set. Comparisons are always against a model trained from random initial weights.

First, the rotation augmentation was tested for its impact on performance. As mentioned before, the random rotation augmentation randomly rotates an image during training by 0, 90, 180 or 270 degrees.

	mAP	MAE	sMAPE
Rotation enabled	0.6869	11.36	0.2743
No Rotation	**0.6919**	**9.7**	**0.2492**

Table 3.1: Comparison of metrics with and without rotation

But as seen in Table 3.1, rotation does not improve any of the metrics, in fact it decreases every one. The deterioration however is not large, but indicative of the fact that random rotation does not improve generalization of the model.

Next, the inpainting augmentation was tested.

	mAP	MAE	sMAPE
inpainting augmentation	0.6591	11.46	0.2782
no inpainting augmentation	**0.6919**	**9.7**	**0.2492**

Table 3.2: Comparison of metrics with and without inpainting augmentation

As with the rotation augmentation, the inpainting augmentation does not yield any performance benefits, it decreases the mAP even further than the rotation augmentation (see Table 3.2).

This indicates that the model is more sensitive to images that are not directly inside its domain than it needs more data. The rotation augmentation for example changes the angle the lighting on the images comes from, which is not exactly uniform because of the door in the photo apparatus (see Figure 3.1). So this augmentation does not improve performance on the validation set, because there all lighting is uniform. The inpainting augmentation produces images that are not exactly in the same domain as images in the validation set, because the inpainting process is not perfect (upscaling, edges are not perfect) and produces visible artifacts as seen in Figure 3.10b.

Next, the impact of using the different datasets was evaluated. The photobox dataset is always used as the last step in training but with different strategies for pre-training the model. As a baseline, the Faster R-CNN model is initialized with random weights and then trained with the photobox dataset (referred to as "Random"). For another run, pre-trained weights on the COCO dataset were used instead of randomly initialized weights weights (referred to as "COCO"). Both weight initializations are evaluated with and without using a training step for the AGAR dataset ("+AGAR").

The trained model weights from Majchrowska *et al.* [3] are not public, so training on the AGAR dataset has to be performed as well. Training on the AGAR dataset uses the same training recipe as the aforementioned training for the photobox dataset. Only the patience for reducing the learning rate was set to 3 epochs instead of 10, because of the larger volume per epoch. The resulting performance is comparable with the results obtained in the paper for Faster R-CNN training on the higher resolution subset (see Table 3.3).

	Epochs	mAP	MAE	sMAPE
Results from [3]	**20**	0.493	4.75	0.0532
Replicated results	57	0.574	4.45	0.0721
Replicated results using COCO pre-trained model	36	**0.5955**	**3.96**	**0.053**

Table 3.3: Comparison of metrics on AGAR higher resolution validation set

The increase in average precision can be explained with the increased epochs training on a lower learning rate, which fine-tunes the box coordinate placement.

As visible in Figure 3.12, pre-trained models converge much faster. As explained earlier, the learning rate scheduler stops training after validation loss stagnates for some time. For the COCO+AGAR run that happens already after 51 epochs. For runs that use only one of these datasets for pre-training it happens after roughly 70 epochs, while the randomly initialized run takes more than 150 epochs to finish.

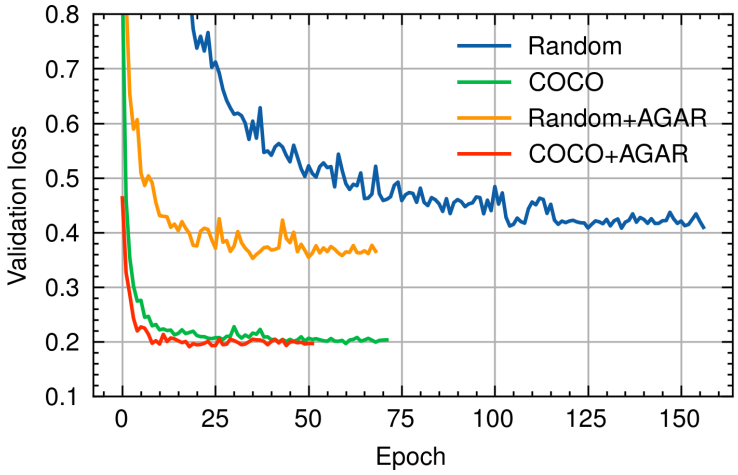


Figure 3.12: Validation loss plot for different dataset combinations

	Epochs	mAP	MAE	sMAPE
Random	156	0.6919	9.7	0.2492
COCO	71	0.7607	6.92	0.1842
Random+AGAR	68	0.7065	8	0.1608
COCO+AGAR	**51**	**0.7908**	**4.64**	**0.1356**

Table 3.4: Comparison of validation metrics for different dataset combinations

Not only do they converge faster, but as seen in Table 3.4 and Figure 3.13, they also perform better. The combination of COCO+AGAR leads to the highest precision and best counting metrics, needing the fewest epochs for the photobox dataset.

Using pre-trained COCO weights instead of randomly initialized weights seems to have a positive impact on precision, improving it by 9.94%. Using the AGAR dataset for pre-training seems to have an especially positive impact on the primary counting metric sMAPE, improving it by 35.5%. Combining these two datasets for pre-training leads to an improvement in precision by 14.3% and sMAPE by 45.6%, while only having to train for approximately one third of the epochs.

Inspecting the performance visually (see Table 3.5) shows that the inclusion of domain specific data (AGAR dataset) and non domain specific data (COCO dataset) in pre-training helps to reduce false positives. Also detection of smaller colonies and colonies that are close together improves. Small colonies in large numbers still propose a problem to the network, as seen in row 3 of the aforementioned table.

Other experiments included:

- **Adjusting batch size and gradient clipping** did not yield any better results. The batch size

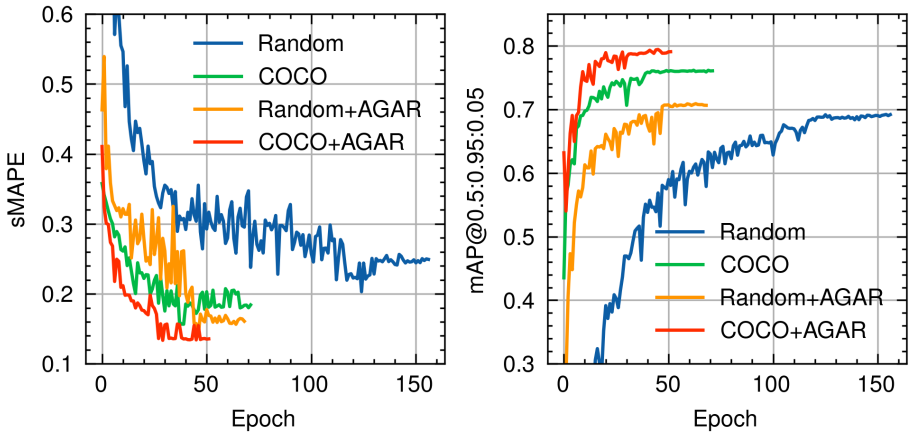
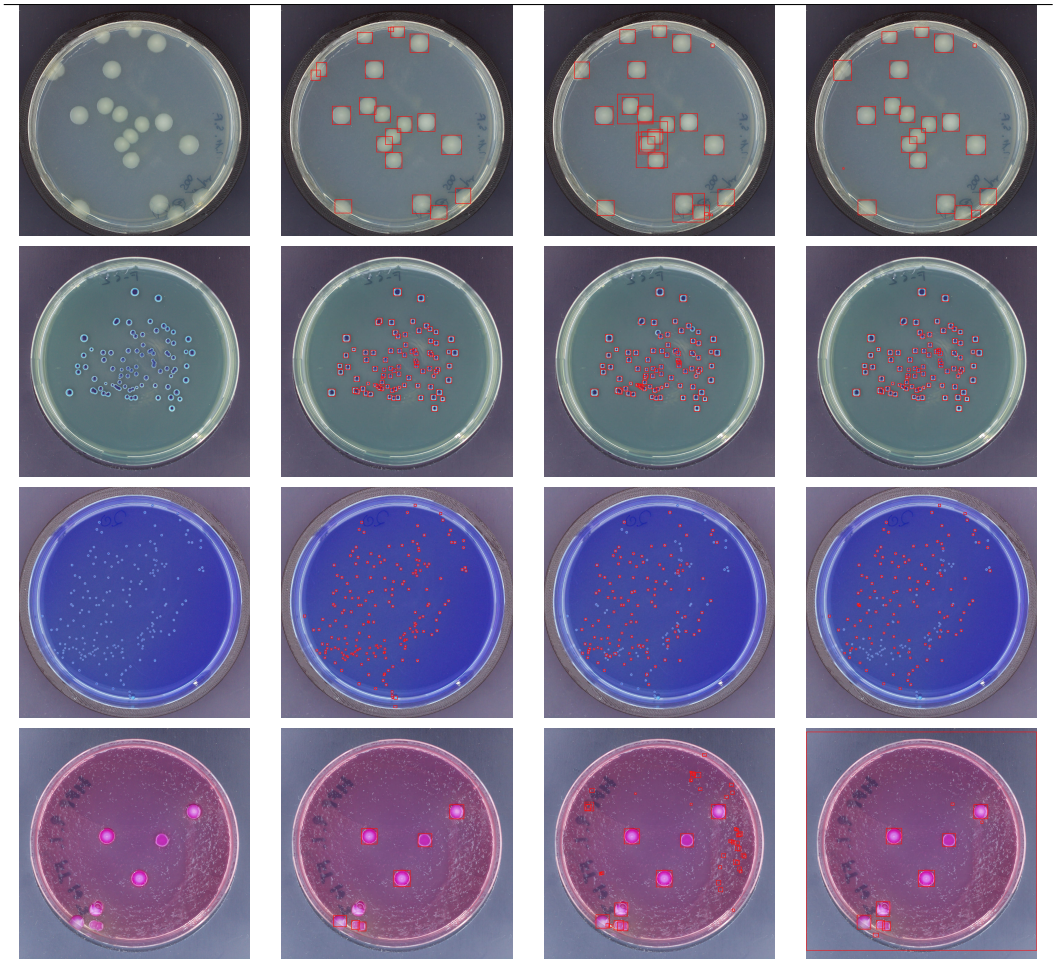


Figure 3.13: Validation metrics on the photobox validation set for different dataset combinations

that fills the available GPU memory (8) worked best and gradient clipping was not necessary there as training is already quite stable at that batch size. Gradient clipping only slows down convergence.

- **Trying out different image resolutions for Faster R-CNN** was also not beneficial. The standard configuration of resizing to 800x800px worked best, using higher resolutions did not improve the performance.
- **Using different optimizers** like Adam and AdamW did not yield better performance. Also their higher memory requirements meant the batch size had to be reduced, which introduced instability to the training.

Table 3.5: Comparison of detections for the different runs

Image	Ground Truth	Random	COCO+AGAR
			

4 Results and Discussion

4.1 Quantitative Analysis

For evaluating the performance of the model on unseen images we use the photobox test set, comprising 50 labeled images. The model only detects up to 300 colonies, so all metrics are calculated with 300 as the upper boundary for ground truth boxes.

An IoU (intersection over union) of 0.5 was chosen for deciding between true and false positives. As seen in Figure 4.1, an IoU of 0.5 requires a quite clear overlap in both boxes to count as a correct prediction, but is also not too strict, because accurate counting is more important than precise localisation.

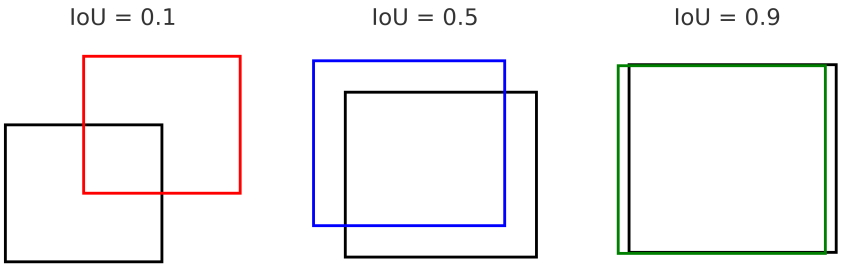
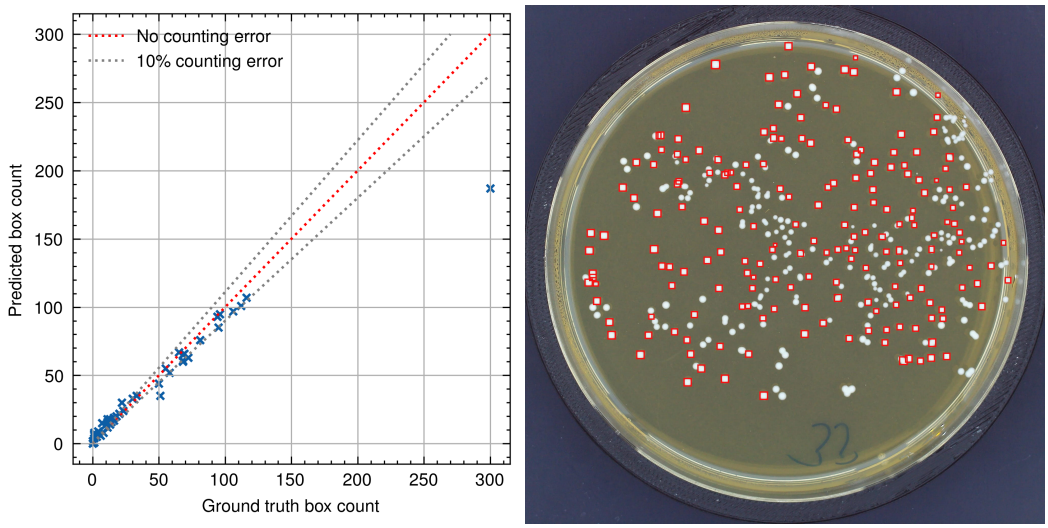


Figure 4.1: Examples of IoU values

Of all the 1,924 found boxes on the test set, **92.9%** were in fact colonies (precision over all boxes in test set). Also **87.3%** of the 2,145 boxes present in the labels of the test set were predicted (recall over all boxes in test set). The mean absolute error for counting the test set is **6.26**, the sMAPE is **0.153**.

When taking a mean over the individual precision and recall values for each of the 50 images in the test set, mean precision is 78.1% and mean recall is 90.1% (this is *not* related to the mAP metric used earlier). Interestingly, when taking these values while including only images that contain 2 or more colonies, mean precision is 86.6%. This shows that false positives on empty dishes are a problem. When taking only images with more than 50 colonies into account, mean precision rises to 97.7%, which shows that false positives almost never occur for images with higher numbers of colonies.



(a) Chart showing counting error

(b) Outlier image in test set

Figure 4.2: Counting performance on test set

This is trend is also visible in Figure 4.2a. The model counts too many colonies for low colony counts (lower precision) but too few for images with more colonies (lower recall). The outlier seen in the chart is image Figure 4.2b, where only 187 of 398 colonies are detected. This shows the model has significant recall problems for large colony counts.

4.2 Qualitative Analysis

To investigate the strengths and weaknesses of the model, all 276 unlabeled images from the photobox dataset were given to the model. Some of the results are shown in Figure 4.3.

Generally, the model is able to precisely localize colonies when isolated. It generalizes beyond one style of image, having no problems finding colonies on different backgrounds or colonies of different shape and color.

As seen in the top right example of Figure 4.3, the model can also segment colonies which have grown together a lot. It is also decent at detecting dishes as empty, with some false positive detections that have very low confidence scores. To combat this issue, confidence scores should be sharpened through more training (on more data) and the confidence threshold should be tuned.

It is really good at colony detection on images that contain only a single species with a light background, such as the bottom right images of Figure 4.3. This is to be expected since a large part of the training dataset comprises similar images. The model seems to be fine for mixed species too,

detecting two different types of species even if they differ in size, shape and color (see 9th image on Figure 4.3).

The 4th image of Figure 4.3 shows that detection also goes beyond circle shapes. Even odd shapes can be detected and amorphous colonies, such as colonies that grow on the edge of the plate, are reliably detected as well.

The main problem of the model is detection in clusters of densely packed colonies, such as the top left image of Figure 4.3. The boxes for these colonies are very close together and very small. This makes these images very inefficient to label, especially with the SAM-assisted labeling method described earlier, which is why they are rarely present in the photobox training data. Since they are very difficult to count, even for humans, some images of this type are marked as "uncountable" in the AGAR dataset. [3]

As seen in the quantitative evaluation, the model has difficulties finding all colonies when there are a lot of them on the image. The photobox dataset is biased towards lower numbers of colonies per image, so this only makes sense. A more balanced dataset would most likely help combat this weakness.

Large false positive detections are also a minor issue, such as in the 4th image of Figure 4.3. Since they usually have a very low confidence score, these should be fixable through post-processing and longer training schedules on more data.

4.3 Discussion

The precision of 92.9% and recall of 87.3% over the test set indicate strong performance for detecting colonies, however they leave room for improvements. The mean absolute error of 6.26 shows that the model cannot yet perfectly count the colonies in an image, but can approximate the number of colonies in an image pretty well.

This means the model can aid in counting and greatly reduce the time taken to count the colonies on a plate, since model errors are easy to spot. But if a precise count is needed, the model cannot yet be used in a fully automated fashion.

The impact of false positives on the detections for images with a low number of colonies on them is quite high. These false positives however mostly have a low confidence score. If the confidence score could be sharpened, these false positives would be sorted out easier. This could be improved by training longer on a larger dataset or finding reliable ways to augment the dataset.

For an average image of the test set, 78.1% of predictions made on the image are correct and 21.9% are false positives (precision). For recall, it correctly predicts 90.1% of colonies and misses 9.9%. Having a higher recall than precision is desirable here, since false positives are easier to dismiss than having to manually add missing bounding boxes.

The qualitative analysis matches the quantitative results, and shows which weaknesses hurt performance metrics. The weaknesses can be mostly traced back to imbalance in the training set, which is something that has to be improved for better general performance.

The difference in performance also brings up the question if specifically tuned models for different types of plates and species could yield better performance, since the model is very capable for certain types of images (spaced out colonies on clean backgrounds), while having trouble with other plates, which have artifacts on them, irregular shapes or just generally colonies of large area with odd shapes e.g. mold.

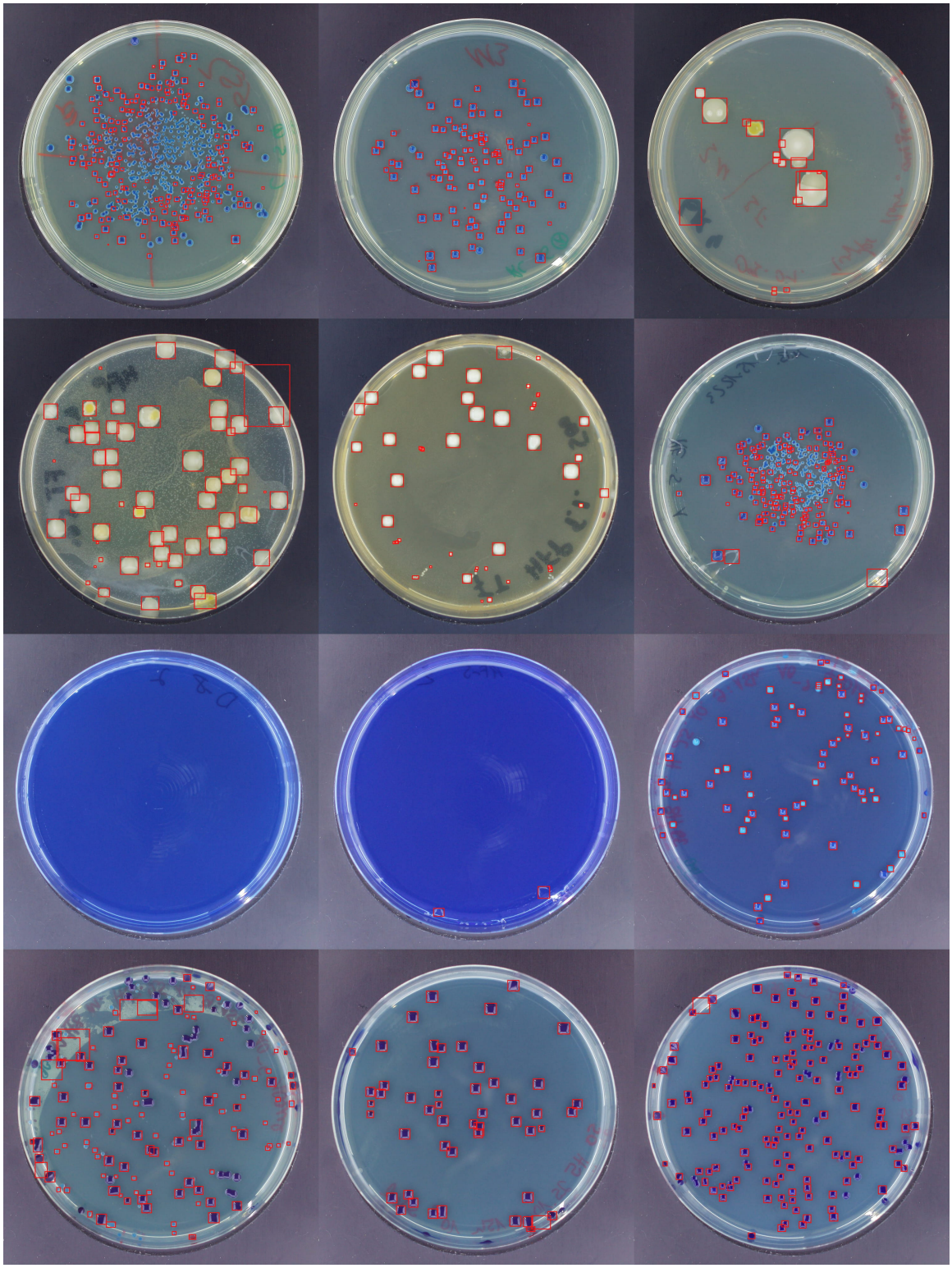


Figure 4.3: Examples of predictions for unlabeled images from the photobox dataset

5 Conclusion and Outlook

5.1 Conclusion

This thesis introduces a dataset with photos taken from hardware at the microbiology lab of FH Münster. Along with this an application for labeling is developed to make it quick and easy to create instance segmentation masks for colonies on plate images. 200 images are labeled with the help of the app. The app can be used continuously to increase the size of this dataset.

A Faster R-CNN model for object detection is trained on different configurations, with different combinations of data. Enhancing the created dataset with additional data yields a significant performance improvement. Other techniques for augmenting the dataset such as creating new images through the use of image generation models are explored but do not yield performance benefits.

Evaluating the best model from the experiments shows promising performance even for the small dataset size, and can be used to speed up the time-consuming counting process for plates by providing predictions that are correct approximately 8 out of 10 times while finding about 90% of the colonies on an image.

5.2 Outlook

First of all, the model trained can be used via a simple web app as seen in Figure 5.1. The app is already deployed on a server at the FH Münster which performs the computation and can be accessed through a web browser on any device. So any photos taken with the photobox can be quickly counted with the help of the model.

One of the main areas for improvement is enlarging the dataset. It currently only holds 100 images for training and is biased towards certain image and colony types. This hurts general performance as previously shown. There are two main ways to increase the dataset size.

Figure 5.1: Screenshot of the app used for detection

First, the labeling tool introduced earlier can be used instead of counting by hand. This creates segmentation masks as a byproduct of counting colonies, which is a task that has to be performed frequently anyways.

The second opportunity for increasing the dataset size is to close the loop between inference and training. When using the model today for quick counting, it will sometimes make incorrect predictions which have to be corrected by a human. If these corrections are recorded, a new entry for the dataset could be created, ever improving the model performance.

This work focuses mostly on creating a suitable dataset and pipeline to create a great foundation for an automated detection system. Therefore only one model architecture, Faster R-CNN, was tested to prove that great detection on this task is attainable. It is possible that different model configurations or architectures perform better on this task. To squeeze more performance out of the existing dataset, it could prove useful to try out different model backbones or try out a single-shot object detection model like YOLOv4, which provided the best performance in Majchrowska *et al.* [5].

Furthermore object detection architectures like Faster R-CNN support classification of objects, so it would be possible to not only locate colonies but also classify their species. For this, the dataset would have to be expanded by a classification layer, in addition to the segmentation masks.

And as a final remark, everything shown in here does not specifically rely on the photobox images.

Smartphone cameras can also be used to capture the petri dish images, the labeling software can also be used in the same way for all types of images, and the model does not necessarily need high resolution images to reliably detect the colonies.

So in the future, students at the microbiology lab from FH Münster could take pictures of dishes using their smartphones, count the colonies using the help of the model developed here and the labeling app, which would feed back into the dataset and continuously improve the detection performance.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] A. Ferrari, S. Lombardi, and A. Signoroni, "Bacterial colony counting by convolutional neural networks," vol. 2015, Aug. 2015, pp. 7458–7461. DOI: 10.1109/EMBC.2015.7320116 (cit. on p. 3).

[2] L. Makrai, B. Fodróczy, S. Á. Nagy, *et al.*, "Annotated dataset for deep-learning-based bacterial colony detection," *Scientific Data*, vol. 10, no. 1, p. 497, Jul. 2023, ISSN: 2052-4463. DOI: 10.1038/s41597-023-02404-8. [Online]. Available: https://doi.org/10.1038/s41597-023-02404-8 (cit. on p. 3).

[3] S. Majchrowska, J. Pawłowski, G. Guła, *et al.*, *Agar a microbial colony dataset for deep learning detection*, 2021. arXiv: 2108.01234 [cs.CV] (cit. on pp. 3, 8, 17, 19, 25).

[4] J. Whipp and A. Dong, "Yolo-based deep learning to automated bacterial colony counting," Dec. 2022, pp. 120–124. DOI: 10.1109/BigMM55396.2022.00028 (cit. on p. 3).

[5] S. Majchrowska, J. Pawłowski, N. Czerep, A. Górecki, J. Kuciński, and T. Golan, "Deep neural networks approach to microbial colony detection–a comparative analysis," in *Lecture Notes in Networks and Systems*. Springer International Publishing, 2022, pp. 98–106, ISBN: 9783031114328. DOI: 10.1007/978-3-031-11432-8_9. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-11432-8_9 (cit. on pp. 4, 16, 30).

[6] F. Yang, Y. Zhong, H. Yang, Y. Wan, Z. Hu, and S. Peng, "Microbial colony detection based on deep learning," *Applied Sciences*, vol. 13, no. 19, 2023, ISSN: 2076-3417. DOI: 10.3390/app131910568. [Online]. Available: https://www.mdpi.com/2076-3417/13/19/10568 (cit. on p. 4).

[7] B. Zhang, Z. Zhou, W. Cao, X. Qi, C. Xu, and W. Wen, "A new few-shot learning method of bacterial colony counting based on the edge computing device," *Biology*, vol. 11, no. 2, 2022, ISSN: 2079-7737. DOI: 10.3390/biology11020156. [Online]. Available: https://www.mdpi.com/2079-7737/11/2/156 (cit. on pp. 4, 13).

[8] J. Pawłowski, S. Majchrowska, and T. Golan, "Generation of microbial colonies dataset with deep learning style transfer," *Scientific Reports*, vol. 12, p. 5212, 2022. DOI: 10.1038/s41598-022-09264-z. [Online]. Available: https://doi.org/10.1038/s41598-022-09264-z (cit. on p. 4).

[9] E. L. Dehn, *Identification of lactic acid bacteria using colonies on mmrs-bpb and mmrs-agar: A novel experimental setting and a machine learning approach*, Bachelor Thesis, Münster, Germany, May 2023 (cit. on p. 7).

[10] Raspberry Pi Foundation. "Raspberry pi high quality camera." (2024), [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/ (visited on 06/20/2024) (cit. on p. 7).

[11] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV] (cit. on p. 8).

[12] B. Sheng, M. Zhou, M. Hu, Q. Li, L. Sun, and Y. Wen, "A blood cell dataset for lymphoma classification using faster r-cnn," *Biotechnology & Biotechnological Equipment*, vol. 34, no. 1, pp. 413–420, 2020. DOI: 10.1080/13102818.2020.1765871. [Online]. Available: https://doi.org/10.1080/13102818.2020.1765871 (cit. on p. 8).

[13] S. Koitka, A. Demircioglu, M. S. Kim, C. M. Friedrich, and F. Nensa, "Ossification area localization in pediatric hand radiographs using deep neural networks for object detection," *PLoS One*, vol. 13, no. 11, e0207496, 2018. DOI: 10.1371/journal.pone.0207496 (cit. on p. 8).

[14] "Coco dataset." (), [Online]. Available: https://cocodataset.org/#explore (visited on 06/04/2024) (cit. on p. 9).

[15] A. Kirillov, E. Mintun, N. Ravi, *et al.*, *Segment anything*, 2023. arXiv: 2304.02643 [cs.CV] (cit. on p. 10).

[16] "Kandinsky 2.2." (2023), [Online]. Available: https://huggingface.co/kandinsky-community/kandinsky-2-2-decoder-inpaint (visited on 06/04/2024) (cit. on p. 14).

[17] A. Razzhigaev, A. Shakhmatov, A. Maltseva, *et al.*, *Kandinsky: An improved text-to-image synthesis with image prior and latent diffusion*, 2023. arXiv: 2310.03502 [cs.CV] (cit. on p. 14).

[18] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022. arXiv: 2204.06125 [cs.CV] (cit. on p. 14).

[19] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2022. arXiv: 2112.10752 [cs.CV] (cit. on p. 14).

[20] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV] (cit. on p. 16).

[21] "Faster r-cnn – torchvision main documentation." (), [Online]. Available: https://pytorch.org/vision/main/models/faster_rcnn.html (visited on 06/07/2024) (cit. on p. 16).

[22] Y. Li, S. Xie, X. Chen, P. Dollar, K. He, and R. Girshick, *Benchmarking detection transfer learning with vision transformers*, 2021. arXiv: 2111.11429 [cs.CV] (cit. on p. 16).

[23] "Torchvision object detection finetuning tutorial." (), [Online]. Available: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html (visited on 06/07/2024) (cit. on p. 16).

[24] "Fasterrcnn_resnet50_fpn_v2 – torchvision main documentation." (), [Online]. Available: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn_v2.html#torchvision.models.detection.FasterRCNN_ResNet50_FPN_V2_Weights (visited on 06/07/2024) (cit. on p. 16).

[25] "Coco - common objects in context: Detection evaluation." Accessed: 2024-06-21. (2024), [Online]. Available: https://cocodataset.org/#detection-eval (cit. on p. 18).

Declaration of Academic Integrity

I hereby confirm that this thesis, entitled

Automating Bacterial Colony
Segmentation with Deep Learning

is solely my own work and that l have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited. I am aware that plagiarism is considered an act of deception which can result in sanction in accordance with the examination regulations.

Leonhard Driesch, Münster, June 26, 2024

I consent to having my thesis cross-checked with other texts to identify possible similarities and to having it stored in a database for this purpose. I confirm that I have not submitted the following thesis in part or whole as an examination paper before.

Leonhard Driesch, Münster, June 26, 2024