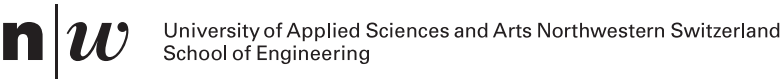
Biomedical Image Segmentation using Generative Adversarial Networks

Michael Aerni, Patrick Del Conte

A thesis presented for the degree of
Bachelor of Science FHNW in Computer Science



Advisors: Martin Melchior, Gabriele Torre
Institute of 4D Technologies
University of Applied Sciences and Arts Northwestern Switzerland, School of Engineering
Windisch, August 18, 2017

Abstract

Generative adversarial networks (GANs) are powerful unsupervised representation learning models. While recent publications propose the usage of GANs in supervised settings, adversarial loss is only used as an extension. No models solely based on adversarial loss have been applied to supervised tasks yet. In this thesis, we evaluate the performance of adversarial models on supervised tasks while omitting traditional supervised loss formulations. We create a biomedical image segmentation network, trained on skin lesion segmentations. We additionally propose a modified GAN objective by introducing context-aware discriminator loss. Experiments demonstrate the potential of supervised adversarial-only models and show how a more sophisticated discriminator loss formulation benefits training stability as well as the results.

Preface

This thesis underwent some singificant changes before it even started. It was originally titled *Automated diagnostic tool for Melanoma* and had the goal to create an automated machine learning classifier which predicts a diagnosis for skin lesion images as either malignant melanoma or benign melanocytic nevus. This idea stemmed from one of our advisors and had the goal to aid in early melanoma detection and prevention.

Shortly before the start of our thesis, Stanford University published a new machine learning model which was able to outperform trained dermatologists at lesion classification [1]. As Stanford's data set and budget far exceeded ours, it was deemed unrealistic to reproduce their results.

We discovered skin lesion segmentation from the same source our original goal stemmed. As the Stanford publication did not cover skin segmentation, we decided to focus on it instead. We were further involved in generative adversarial networks at that time and decided to use the opportunity to research them with our use case.

While the change in subjects delayed the start of our actual thesis a few weeks, we utilized the time to enhance both our technical and domain knowledge, which ultimately helped us to finish our work in time.

We would like to thank our advisors for supporting us during this thesis by providing valuable discussions and feedback to our work. Due to this fact, we did not just finish a product for some customer but gain a lot of knowledge.

Lastly, we would also like to thank the *Institute of 4D Technologies* for providing us with computational resources. We were able to utilize one of their computers as well as Amazon cloud instances for the training of our models. Without those resources, we would not have been able to produce even half of our experiments.

Contents

1. Introduction

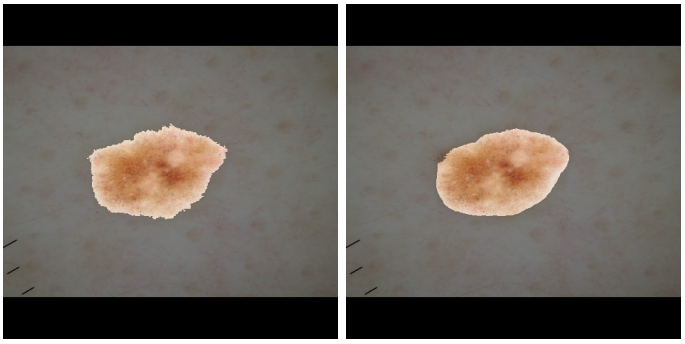
Generative adversarial networks (GANs) [2] denominate a new type of neural network models. They are trained with a set of input samples and learn to generate similar outputs. For example, a GAN trained with enough images of cats will learn to generate completely new and realistic cat images on its own [3]. Previously, similar models had to be individually developed for each new task. In contrast, GANs can be applied to a multitude of generative problems without substantial modifications. As of today, they produce state-of-the-art results for tasks such as image-synthesis [4], image super-resolution [5] or even audio-synthesis [6]. Even though the original GAN authors only considered unsupervised tasks, GANs have successfully been applied to supervised problems too [7]. However, in those cases, GANs are always used as an extension to an otherwise supervised model. To the best of our knowledge, no fully-supervised model solely utilizing GANs exists yet.

Artificial intelligence models like GANs and other neural networks have become an important tool in a variety of medical applications such as skin cancer detection. As of today, skin cancer is the most common type of cancer in humans. Melanoma denotes the most aggressive form, affecting over 2000 people and claiming hundreds of lifes in Switzerland alone every year [8]. In other countries with reduced healthcare, those numbers are significantly higher. However, if recognized and treated in its early stages, melanoma can easily be treated [9]. The availability of smartphones and portable dermatoscopes, in combination with automated artificial intelligence diagnostic algorithms, makes self-diagnosis accessible to a large fraction of the public and thus aids in early melanoma detection and treatment.

The initial step in manual and automated melanoma detection is image segmentation. Segmentation denotes the process of identifying relevant pixels in a biomedical image. In the case of skin cancer, this signifies masking all pixels belonging to a skin lesion, resulting in a so-called segmentation mask. Segmentation masks aid doctors and automated algorithms in diagnosis.

We develop a completely automated skin lesion segmentation model based on GANs. Figure 1.1 displays a segmentation created by a trained dermatologist and one created by our model. In contrast to other models solving this task, we completely omit traditional supervised loss formulations and rely only on GANs. As no comparable publications for this approach exist yet, we answer the questions if GANs can be used in a fully-supervised way and how they perform compared to traditional models.

Our contribution is to show that GANs can be used without additional supervised techniques. We propose a modified GAN optimization objective and empirically show that it reduces training times and improves results in fully-supervised scenarios. We further contribute an adversarial-only skin lesion segmentation model and a corresponding



(a) Expert segmentation (b) Generated segmentation

Figure 1.1.: Lesion segmentation created by an expert and our model

command line application. We term our resulting model *Molanet* due to its correlation with melanoma detection. Our model does not match the results of competitors yet but could in a further step be improved to do so.

Biomedical image segmentation is an important diagnosis step in a multitude of fields, not exclusively skin cancer prevention. Computer-assisted segmentation techniques benefit a variety of medical use cases such as hippocampus MRIs in Alzheimer's Disease detection and lower abdominal CT scans used to uncover liver tumors. Automated segmentation algorithms in most areas face similar problems:

- Training data is scarcely available and lacks unified imaging standards.
- Lesions often exhibit diffuse shapes, making them hard to differentiate from the background.
- Biomedical images contain an array of visual artifacts such as blood vessels, light reflections, and other domain-specific relics.

Therefore, improved techniques in one area benefit other areas too. For that reason, the international research community is fostering biomedical image segmentation progress. This is evident in the annual challenges hosted by the *IEEE International Symposium on Biomedical Imaging* (ISBI) conference. Our use case is based on one of the four challenges, the *ISBI Challenge on Skin Lesion Analysis Towards Melanoma Detection*. Together with the *International Skin Imaging Collaboration* (ISIC) the challenges provide the *ISIC Archive* [10] containing over 15000 freely available skin lesion images, each accompanied by one or more segmentation masks. The ISIC archive was one of the major resons for us to choose skin lesion segmentation as our use case.

As of today, skin lesion segmentation techniques suffer various problems. Performing the masking manually is very time consuming. While automated approaches exist, their algorithms exhibit a multitude of specialized parameters. Both approaches require in-depth expert knowledge and thus limit their accessibility to a fraction of the general public. While artificial intelligence models, especially neural networks, pose fully automated approaches, their development is aggravated by a lack of data. Only few publicly

available data sets exist, all consistently exhibiting an absence of imaging standards. This makes the development of machine learning models very difficult, even though various recent breakthroughs in computer-vision showcase their effectiveness. Motivated by the recent success of GANs, we use an adversarial approach to compensate for the lack of standard in ground truth data while still providing a fully automated process. We expect GANs to inherit the powerful computer-vision capabilities of convolutional neural networks while being resistant to the poor ground truth quality.

We approach this task by first acquiring an in-depth theoretical understanding of GANs as they are still actively researched and not fully understood yet. We identified the most important papers as the original formulation of generative adversarial networks by Goodfellow, Pouget-Abadie, Mirza, *et al.* [2], their thoughts on improving GAN training [11] as well as the successful application of GANs to convolutional neural networks by Radford, Metz, and Chintala [4]. Chapter 2 provides a theoretical explanation of GANs and an introduction to their state of research, establishing our theoretical foundation. Chapter 3 presents the data used in this thesis, followed by a qualitative and quantitative analysis. This concludes in the strategy used to partition our data into multiple sets. Finally, we answer how such data can efficiently be stored and handled. We then develop a lightweight framework which allows us to quickly create and compare different GAN-based models. The framework is written in *Python* and uses the *TensorFlow* numerical computation library, as documented in chapter B. Based on this framework, we create a proof-of-concept model to answer the question if our task is feasible using GANs. This proof-of-concept further establishes our base architecture, discussed and analyzed in chapter 4. Chapter 5 then explores three extensions to the base architecture. We evaluate and compare these extensions, concluding in our final architecture. Finally, our results are presented in chapter 6. We first discuss our final architecture and afterwards compare it to the top *ISBI Challenge on Skin Lesion Analysis Towards Melanoma Detection* submissions of the last two years.

2. Generative Adversarial Networks

2.1. Generative Models

In the last few years, machine learning algorithms, especially convolutional neural networks, have vastly outperformed traditional algorithms at image classification [12]. On the other hand, the reverse problem of generating realistic images has proven more challenging and fuels a very active area of research.

From a mathematical perspective the formulation of supervised problems, such as classification, is strongly supported by a clear notion of *right* and *wrong*. In contrast the formal definition of what constitutes a realistic image is very hard.

A generative model is a statistical model used to generate examples that belong to a distribution. These models have in common that they are defined in terms of likelihood. Evaluating a generative model's performance, which is needed for training, requires some knowledge about what constitutes a realistic example in the first place. In particular, a distance metric or first order approximation which defines the distance between the real distribution and the approximated one is needed. The formulation of such distance metric is a non-trivial task and usually distinguishes different types of generative models.

2.1.1. Generative Adversarial Networks

Recently a new kind of generative model, generative adversarial networks (GANs), has shown promising results because it learns an appropriate notion of *right* and *wrong* i.e. a loss function from training examples. It uses this learned loss function to improve its own generation method to produce more realistic looking samples.

In brief GANs consist of two parts, a generator and a discriminator network. The discriminator learns to differentiate real samples from generated ones. It learns the loss function that is usually so hard to define. During training the discriminator receives either the generator's output or real samples as input. The generator learns to produce an image the discriminator will classify as real. They are called adversarial networks because generator actively tries to fool the discriminator while in turn the discriminator tries to make as few mistakes as possible. The intuition is, if the generator is trained very well then the discriminator will have to guess whether an example is real, at that point the generator produces realistic output.

Generative Adversarial Networks found applications in a variety of fields such as images-synthesis [4], image super-resolution [5] or even audio-synthesis [6].

2.2. GAN Theory

Let us define x as a datapoint drawn from the probability distribution $\mathbb{P}_{data}$ and z as a datapoint drawn from the probability distribution $\mathbb{P}_z$. $\mathbb{P}_{data}$ is the distribution of the data that should be learned by the generator while $\mathbb{P}_z$ is usually defined as a random distribution. In a sense we want to use the generator to approximate a function with domain $\mathbb{P}_z$ and range $\mathbb{P}_{data}$ that can be used to infer the distribution of $\mathbb{P}_{data}$, a so-called estimator.

Let $D(\hat{x})$ be the discriminator function and $G(z)$ the generator function, both implemented by a neural network parameterized by learnable parameters θ_D and θ_G respectively. The generator function emits a tensor with the same shape as x. The discriminator function outputs a probability that $\hat{x}$ is from $\mathbb{P}_{data}$ rather than generated by G.

Equation (2.1) describes a two player minimax game played by the generator and discrimantor in a generative adversarial network [2]. A minimax game is a game for two players where both aim to minimize their own loss. More in detail, in this formulation GANs play the so called zero-sum minimax game in which the minimization of one player's loss leads to the maximization the other player's loss.

$$\min_G \max_D \mathrm{GAN}(D, G) = \mathbb{E}_{x \sim \mathbb{P}_{data}}[\log(D(x))] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D(G(z))] \tag{2.1}$$

The GANs likelihood function is defined as the linear combination of the expected value of $log(D(x))$ and the expected value of $log(1 - D(G(z)))$. The minimax game is set up by choosing opposite optimization objectives. While the discriminator maximises the probability to assign the correct label to examples from the data and produced by the generator, the generator tries to generate samples which are labeled as realistic looking by the discriminator [2].

It has been proven that using the above optimization objective the generator minimizes the Jenson-Shannon divergence between its output and the probability density function of the data [2]. The Jenson-Shannon divergence is a measure of difference between two probability density functions. Therefore G acquires implicit knowledge about the distribution $\mathbb{P}_{data}$ since the generated samples seem to be drawn from it.

Algorithm 1 lists the original formulation of the GAN training[2]. The algorithm iteratively trains towards the objective defined in eq. (2.1), first updating the discriminator k times and then the generator once. The iterative updating incentivizes the discriminator to focus on the weak spots of the generator, since it will be more confident in those areas, which in turn leads to improvements in the responsible parameters in the generator. Computing k discriminator updates before each generator update may provide faster convergence and more accurate results at the cost of additional computation time [2]. It has been proven that using a perfect discriminator, which would mean $k = \infty$, leads to

convergence [13].

foreach number of training iterations **do**

  foreach k discriminator updates **do**

    - Sample minibatch of m samples $\{z^{(1)}, ..., z^{(m)}\}$ from $\mathbb{P}_z$
    - Sample minibatch of m examples $\{x^{(1)}, ..., x^{(m)}\}$ from data generating distribution $\mathbb{P}_{data}$.
    - Update the discriminator by ascending its stochastic gradient:

$$-\nabla_{\theta_D} = \frac{1}{m} \sum_{i=1}^{m} \left[\frac{1}{2} \log \left(D(x^{(i)}) \right) + \frac{1}{2} \log \left(1 - D(G(z^{(i)})) \right) \right]$$

    .

  end

  - Sample minibatch of m samples $\{z^{(1)}, ..., z^{(m)}\}$ from $\mathbb{P}_z$
  - Update the generator by descending its stochastic gradient:

$$-\nabla_{\theta_G} = \frac{1}{m} \sum_{i=1}^{m} \left[\log \left(D(G(z^{(i)})) \right) \right]$$

  .

end

The gradient-based updates can be performed with any standard gradient-based learning rule.

Algorithm 1: Mini-Batch stochastic gradient descent training of generative adversarial networks.

2.2.1. Conditional Generation

With the approach discussed so far, it is possible, in principle, to generate an example belonging to any distribution. However, no further predictions about the output can be made. For example if GANs are used to produce images to mammals there is no way to specify whether lions or zebra should be generated. Naturally, such a conditioning of the generator would be highly desirable.

A condition can be encoded in the input of the generator and discriminator [14]. The simplest way to achieve this would be to concatenate the a numeric representation of the condition to the inputs. The generator must learn to respect these structures, because it will be penalized much more if it doesn't since the output is much easier to dismiss as fake if it has the wrong condition. More formally, the discriminator outputs the conditional probability of it's input being real, given that it exhibits the condition that was encoded in the input.

2.3. Advantages and Disadvantages

GANs and adversarial models in general are still a very active area of research with a continuous stream of publications. So far, no architecture has prevailed above all others but many advances were made. Different optimization objectives, such as minimizing the Wasserstein distance instead of the Jenson-Shannon divergence, have been proposed and verified by empirical results [15][16]. Slowly, the theoretical understanding of adversarial dynamics is catching up, leading to even better methods. Especially the use of deep convolutional techniques has lead to more powerful architectures for image synthesis [4].

Regardless of the promising theoretical foundation, GANs have gained a reputation for being unstable to train in practice [11]. Some of the most frequent problems are

Mode dropping meaning the generator learns to produce only a small subset of the true distribution of the data. This can be observed as oscillating behavior in the generator, where it always generates one of a few outputs [11][15].

Vanishing gradients meaning the norm of the discriminator's gradient becomes very small. This is often the case when the discriminator performs too well. Once the gradient become vanishingly small, the generator updates become meaningless.

Even though many theoretical solution to the above problems have been proposed, there is no single solution which eliminates them in all cases [13][11].

The main advantage of using GANs is that the loss function is learned by the network and doesn't have to be engineered by hand, which would be impractical for all but the simplest examples. This makes them very flexible because, in principle, no expert knowledge is required to train a GAN in a new domain as everything is inferred from the training data.

Compared to existing generative models, GANs have a more tractable gradient function. This means GANs can be efficiently trained with standard gradient descent techniques and thus leverage the capabilities of modern GPUs.

In conclusion, despite the mentioned stability issues, GANs have matured to a level which allows them to meet the challenge of biomedical segmentation.

3. Data

3.1. Data Sources

The collection of a medical image data set is a difficult task as accurately summarized by [17]:

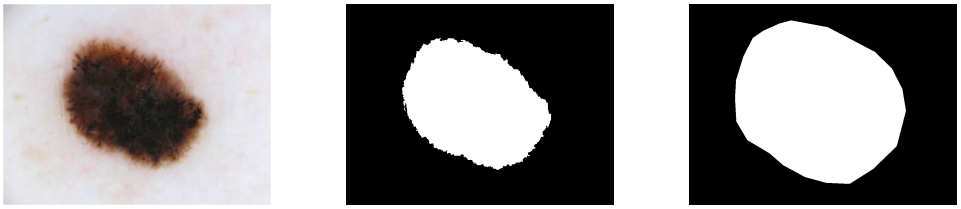
> *The current era of smart-phone technology presents an exciting opportunity for digital photography as an aid to early diagnosis of melanoma. However, the promise of digitally assisted melanoma diagnosis is currently hampered by:*
>
> - *A lack of standards for dermatologic imaging, and*
> - *Limited access of educators and developers to large numbers of high quality clinically annotated skin lesion images.*

This not only holds true for melanoma diagnosis but all biomedical imaging tasks, including but not limited to segmentation. As a lack of standards can be compensated using preprocessing mechanisms, the latter aspect was the deciding factor in chosing a demonstration use case for our network. We primarily chose *skin lesion segmentation* because of the public availability of two data sources which combined contain over 15000 samples.

The primary data source used for training and evaluation is the *ISIC Archive* [10]. It is an open source archive of skin lesion images and meta data, supervised by the *International Skin Imaging Collaboration: Melanoma Project*. Its main goal is to encourage the research behind automated melanoma detection. At the time of writing the data set consists of 13786 skin lesion images. All images are annotated with meta data, including diagnosis and segmentations. Segmentations were created using either flood-fill algorithms or manually drawn shapes and afterwards reviewed and approved. Approvers are categorized as either novices or experts. The advantages of the ISIC Archive are its public availability and free usage. However, it lacks image standards, e.g. normalized brightness, contrast levels or resolutions.

To enlarge our trainig data, we also used the *Dermofit Image Library 3.00* [18]. It is provided by the *University Court of the University of Edinburgh* and available under a non-free licence for academic use only. This library is more focused on image quality and only contains 1300 images at the time of writing. Each sample consists of a lesion image, a diagnosis and a segmentation mask. Masks are exclusively created with flood-fill algorithms.

The data set used in this thesis is the combination of the *ISIC Archive* and the *Dermofit Image Library 3.00*.



(a) Skin lesion (b) Flood-filled mask (c) Manually drawn mask

Figure 3.1.: Sample lesion and its corresponding segmentation masks, taken from [10]

As our focus lies on segmentation, we ignore most of the additional meta data. Images from both data sources are provided in the RGB color space. Skin lesion images are just a photograph of the relevant skin region. Segmentation mask images are binary images where the information is for display purposes encoded as

Black pixels describing background, i.e. skin.

White pixels describing foreground, i.e. lesions.

The segmentation masks can be categorized into two distinct types:

Flood-filled shapes produced by specialized algorithms such as [19]. Those algorithms specify multiple parameters which are fine-tuned by dermatologists. This leads to highly accurate results but requires expert knowledge about the impact of each parameter.

Manual shapes drawn by dermatologists. Those shapes require only dermatologic knowledge and no algortihm knowlege. Manual shapes are counterintuitively less accurate than flood-filled shapes because the effort required to capture fine border structures of a lesion grows proportional to the image size.

Figure 3.1 shows an example of a skin lesion and its corresponding segmentation mask. One mask is generated by a flood-fill algorithm, the other by manually drawing a shape.

3.2. Analysis

As previously mentioned, public data sets often lack standards regarding image format and quality. Inaccurate ground truth data is in many cases associated to inaccurate segmentations and distorted model evaluation results. Such inaccuracies as well as other issues are present in both data sources considered in this work.

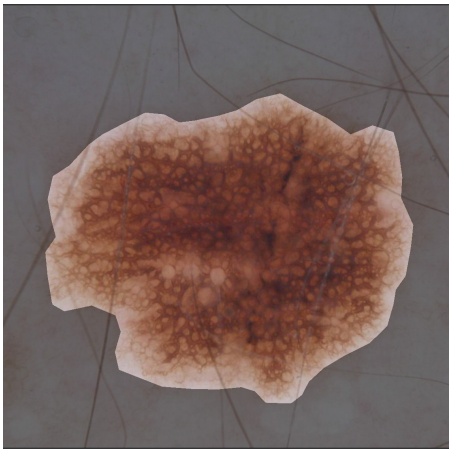
The most prevalent issues of the ISIC Archive are

Inaccurate manual shapes used as segmentation masks. Figure 3.2a shows an example. Darkened parts correspond to the background, brighter parts to the foreground. Such masks may negatively influence post-training evaluation and lead to an incorrect optimization target.

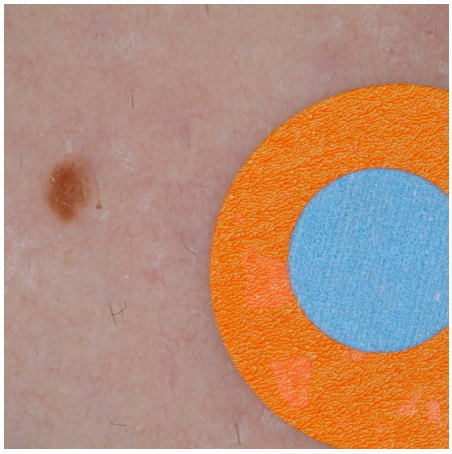
Visual artifacts such as plasters or rulers. Figure 3.2b and fig. 3.2c each show an example. The resulting network may detect the artifacts and wrongly segment them as lesions. However, if learned correctly, artifacts can also aid in generalization and semantic understanding of the resulting network.

Hair overlaying skin lesions. Especially thick hair as seen in fig. 3.2d obstructs parts of the skin lesions and hides information. The resulting network must learn to ignore hair which aggrevates training but serves as a form of regularization.

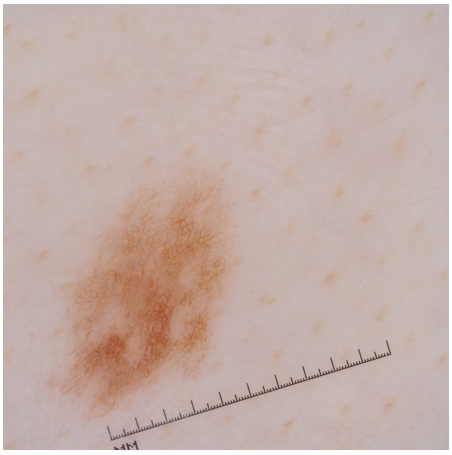
Different resolutions and aspect ratios of the images. There are a total of 323 different resolutions in the ISIC Archive data set, ranging from 576×768 pixels to 6780×4480 pixels. This requires the development of a method which allows arbitrary sized images as segmentation input.



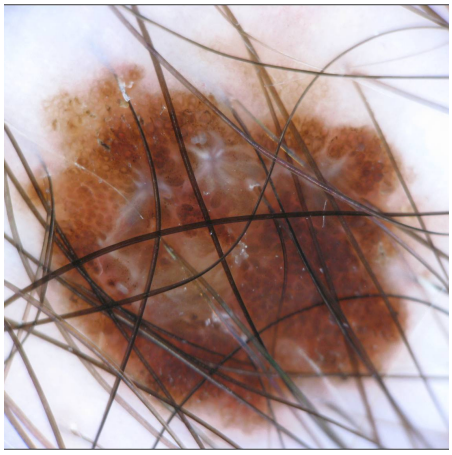
(a) Inaccurate segmentation mask



(b) Image containing a colored plaster



(c) Image containing a ruler



(d) Thick hair overlaying a lesion

Figure 3.2.: ISIC Archive most prevalent issues

As the Dermofit images are generally of higher quality, there are less issues with the lesion images but still some with their corresponding segmentation mask. We identify the following most prevalent issues in the Dermofit Library:

Rare diagnoses not available in the ISIC Archive. Those represent a minority which cannot be learned easily. Figure 3.3a shows a *Basal Cell Carcinoma* diagnosed lesion. Those only make up less than 0.02% of the combined data set.

Inaccurate manual shapes can be found but are less common than in the ISIC Archive. Figure 3.3b shows an example.

Segmented hair is hair masked by the original segmentation algorithm even though it does not overlay any lesion parts. This might hinder the generalization ability of the resulting network because we want it to ignore hair which does not belong to a lesion. The segmented hair might be a result of an inaccurately parameterized flood-fill algorithm as shown in fig. 3.3c.

Low resolution segmentations as in fig. 3.3d. They could be the result of segmenting an image on a low resolution and then upscalig it up to the original image resolution. This procedure produces masks which are less accurate than their full resolution counterparts.

The heterogenity of our data set and the listed issues require a careful sample selection for partitioning the data without introducing any significant bias. Having a biased sample distribution over the test, training and cross-validation sets introduces multiple problems.

Firstly, the network perfomance evaluation may lead to wrong conclusions. If the test set contains only few minority class samples, their errors are negligible in proportion to the majority class sample errors. This prevents reliable conclusions over wether the network generalizes well over the minority classes or fails to correctly segment them. Inaccurate segmentation masks may lead to two possible effects: If the network performs well, it might be wrongly penalized for generating segmentation masks which are better than the ground truth. In contrast, if the network produces inaccurate masks, they might better match the inaccurate ground which results in an overvalued evaluation. Another type of evaluation bias is introduced if a sample image is present in both the training set and the evaluation sets as it prevents the detection of overfitting.

Secondly, the actual training can also be distorted. If not enough minority class samples are present in the training set, the network might not be able to generalize enough. Inaccurate masks can aggravate network training as the presence of conflicting ground truth leads to slow convergence. Inaccuracies might also require a bigger network architecture to be compensated and may even lead to learning a wrong mapping function.

The mentioned problems call for an in-depth analysis of the samples from both data sources.

3.2.1. Methods

To quantify the quality variations in both data sources, we defined a set of features to capture statistics over all samples. For each lesion image and segmentation mask, the

(a) Basal Cell Carcinoma

(b) Inaccurate segmentation mask

(c) Segmentation mask containing hair

(d) Low resolution segmentation mask

Figure 3.3.: Dermofit Library most prevalent issues. For all segmentations darkened parts correspond to the background, brighter parts to the foreground.

listed features are calculated and compared. The results also provide good indications on how we can split our data into training, cross-validation and test sets without introducing significant bias.

For each lesion image, the following features are calculated:

Brightness level approximated by calculating the average and median brightness intensities over all pixels of a grayscale version of the image.

Contrast level approximated by calculating the standard deviation of the brightness intensities over all pixels of a grayscale version of the image.

Plaster presence in the image. This is done by a skin-color mapping, modified from the method of [20]. The image is converted into the *YCbCr* color space and split into patches of equal size. If the confidence of being skin for any patch is lower than an empirically chosen threshold, the whole image is considered to contain plasters or other visual artifacts. A patch is considered skin if most pixels' chroma values reside in a range proposed in [20]. This method does not work for very bright or dark image patches as their chroma values are distorted. Therefore, if the patch exhibits either an unusual mean brightness or very close chroma channel values, it is also considered to be skin.

Hair presence in the image. This is achieved by perfoming a morphological closing operation, converting its output to grayscale and running a Canny Edge Detector [21] on the result. Finally, a probabilistic Hough transform for line detection [22] is computed on the edge image. If the number of resulting lines exceeds an empirically chosen threshold, the image is considered to contain hair. This method assumes that hair in an image introduces many edges and therefore many lines. If the Hough transform reports a lot of lines, the probability of it being a hairy image is high. This method is not perfectly accurate but deemed good enough for quantitive analysis.

Diagnosis of the skin lesion.

The significance of most features was already discussed. Brightness and contrast levels are analyzed to determine how much they differ over the data set as it is expected due to the lack of imaging standards.

For each segmentation mask, the following features are calculated:

Absolute lesion size in a segmentation mask, calculated by counting the number of foreground pixels.

Relative lesion size in a segmentation mask, calculated by dividing the absolute lesion size by the total number of pixels in a mask.

The relative lesion size is only relevant for rescaled versions of the segmentation masks, the absolute lesion size only for full resolution versions. As the image and mask sizes are heterogenous over our data set, it cannot be assumed that the relative and absolute lesion sizes directly correlate. The lesion sizes are important because it is assumed that the network performance differs between big and small lesion sizes.

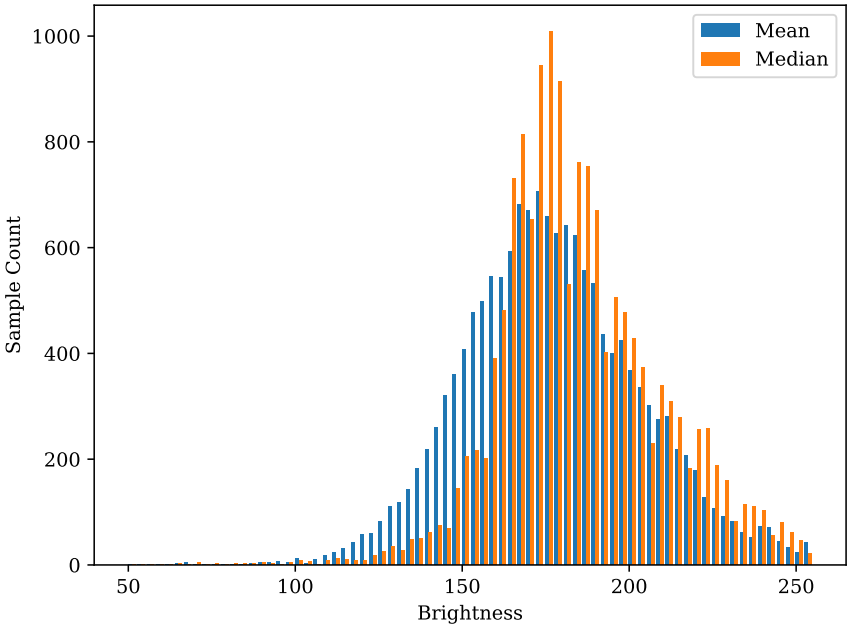


Figure 3.4.: Mean and median brightness histogram

3.2.2. Results

Figure 3.4 shows the distribution of image brightness levels. The average brightness is normally distributed. Some outliers are present on the upper end of the brightness scale. There are no images which are too dark.

Figure 3.5 shows the distribution of the image contrast levels. The contrast is overall rather low. This confirms previous assumptions about the general image quality. Low contrast may be a problem for our network as the borders between lesions and skin become less clear in a low contrast setting.

Finally, fig. 3.6 displays the correlation between brightness and contrast over all lesion images. Darker spots indicate more samples which have similar brightness and contrast levels. The cluster in the bottom right corner belongs to the very bright and low contrast images. As almost all very bright images also lack contrast, it can be assumed that those are results of unsatisfying image acquisition methods.

Figure 3.7 shows the proportion of samples containing certain anomalies. It has to be noted that both results contain many false positives as the utilized methods rely on assumptions and rough empirical testing. Figure 3.7a shows how many samples are classified as containing hair. False positives in this category stem mostly from images containing a lot of thin hair. This results in many detected lines and a positive classification. Figure 3.7b displays approximately how many images contain plasters. False positives are introduced by very bright or low-contrast images as the skin colors become distorted and are reported as being non-skin. Even when accounting for false positive,

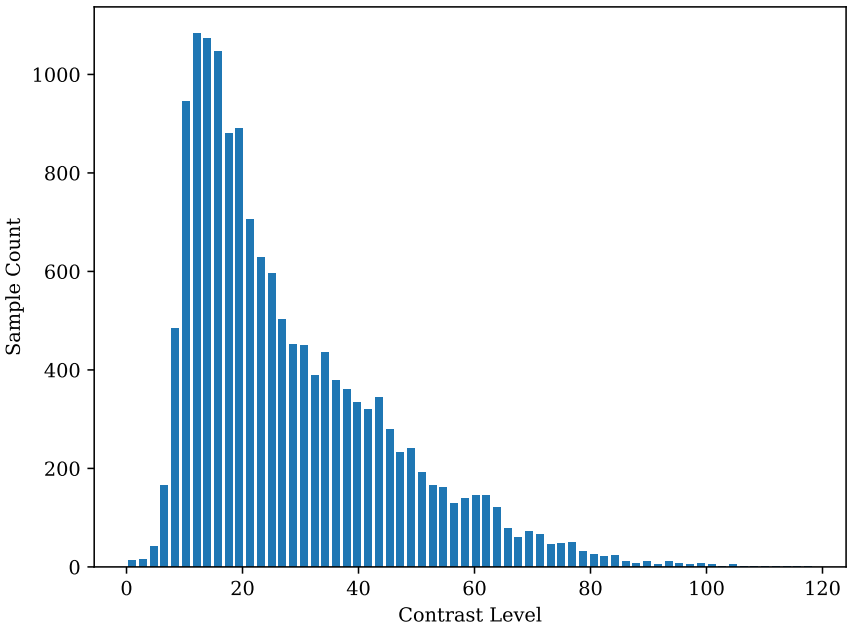


Figure 3.5.: Contrast histogram

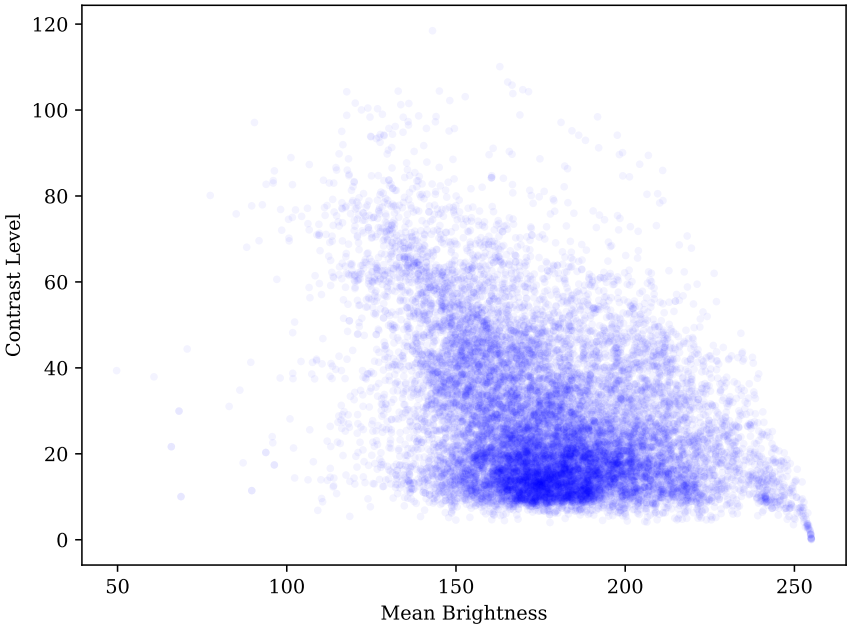
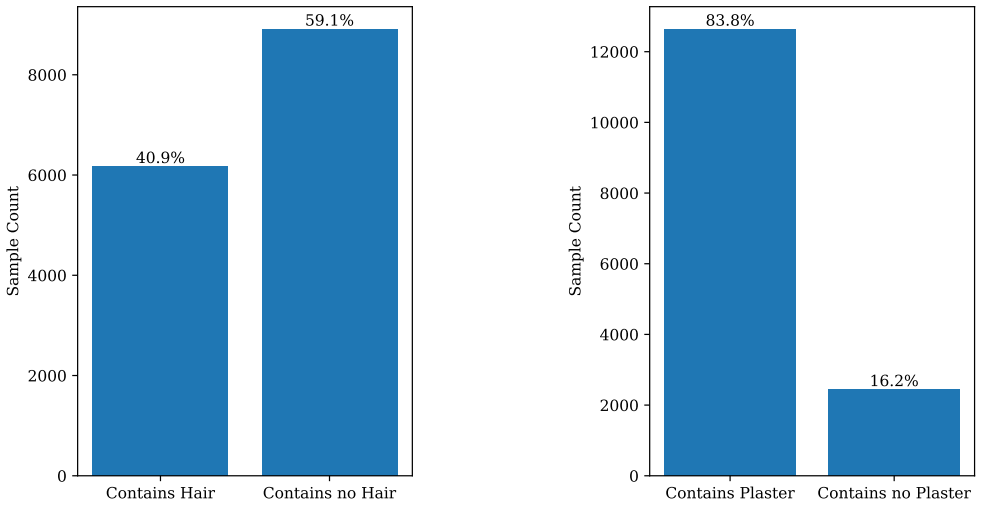


Figure 3.6.: Correlation between brightness and contrast



(a) Proportion of images containing hair (b) Proportion of images containing plasters

Figure 3.7.: Anomaly data set statistics

it can safely be assumed that around a third of all images contain significant amounts of hair and approximately two thirds contain a plaster or other visual artifact.

The final part of the lesion image analysis is the diagnosis distribution. Table 3.1 shows the total and relative occurence per diagnosis over all lesion images in our data set. This confirms the existence of clear minority classes as *Nevus*, *Melanoma* and *Seborrheic Keratosis* already make up over 92% of all lesions while other classes contain only a single image.

Figure 3.8 displays the relative and absolute distribution of segmentation mask sizes. The absolute mask sizes are logarithmically scaled to address the broad range of displayed numbers and a bias value of 1 is added since some segmentation masks do not contain any foreground pixels. Both relative and absolute mask sizes are generally small, apart from some extreme outliers. The outliers in the absolute mask sizes can be explained by the varying image resolutions. Figure 3.9 shows how the relative and absolute mask sizes correlate. Correlation values are distributed along multiple diagonals, indicating a non-global correlation between relative and absolute mask sizes.

3.3. Set Partitioning

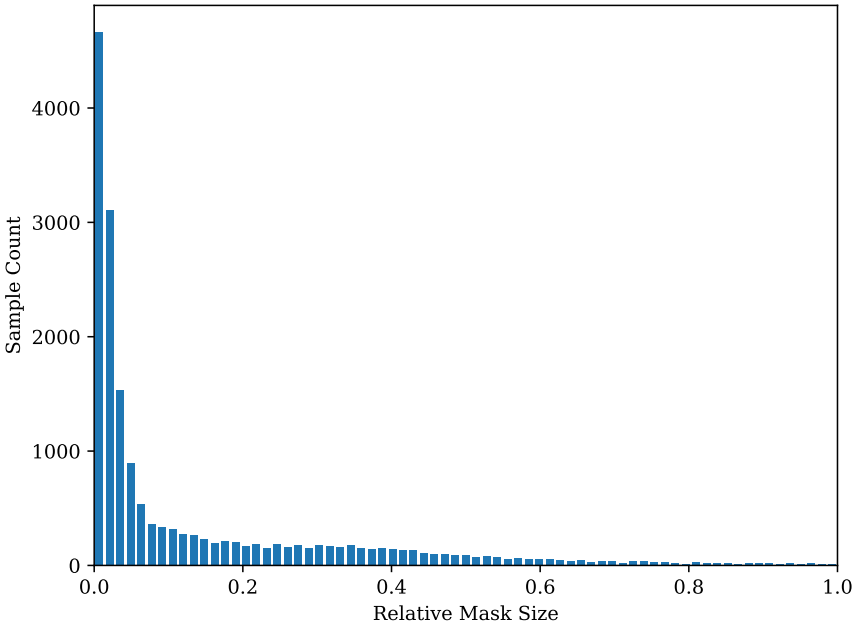
We partition our data into three data sets, each consisting of a disjunct list of samples. A sample is a combination of a lesion image and segmentation mask. As lesion images can have multiple masks assigned, a single lesion image may be part of multiple distinct samples.

We use the following three sets:

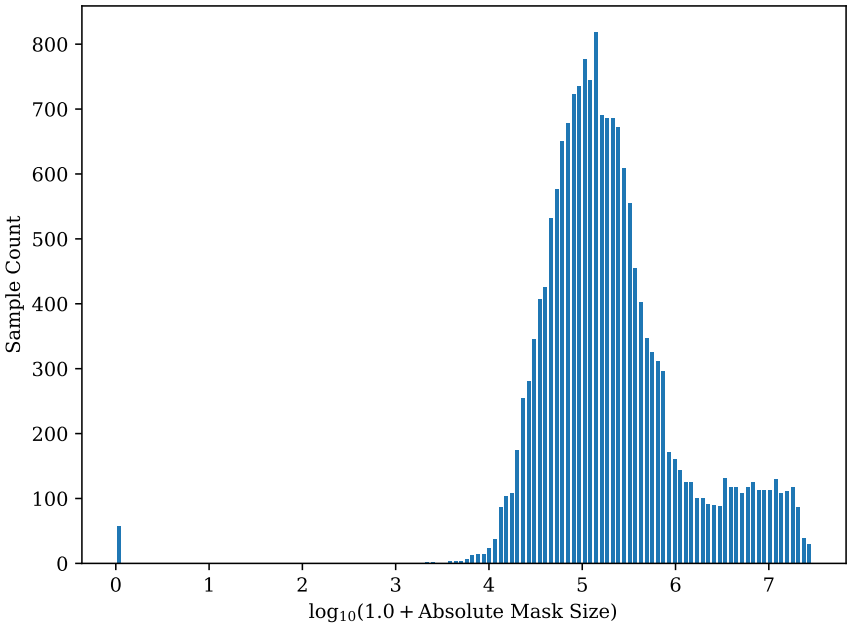
Training set containing the samples used to train our different neural networks.

Diagnosis	Total Occurences	Relative Percentage
Nevus	12192	80.83%
Melanoma	1095	7.26%
Seborrheic Keratosis	676	4.48%
Basal Cell Carcinoma	272	1.80%
Unknown	244	1.62%
Haemangioma	97	0.64%
Squamous Cell Carcinoma	93	0.62%
Intraepithelial Carcinoma	77	0.51%
Dermatofibroma	72	0.48%
Lentigo Nos	71	0.47%
Solar Lentigo	57	0.38%
Actinic Keratosis	47	0.31%
Lentigo Simplex	27	0.18%
Pyogenic Granuloma	24	0.16%
Angioma	14	0.09%
Atypical Melanocytic Proliferation	13	0.08%
Other	10	0.06%
Lichenoid Keratosis	1	0.01%
Scar	1	0.01%
Angiofibroma Or Fibrous Papule	1	0.01%

Table 3.1.: Occurence of Diagnoses in our Data Set



(a) Relative mask size histogram



(b) Absolute mask size histogram

Figure 3.8.: Segmentation Mask Size Statistics

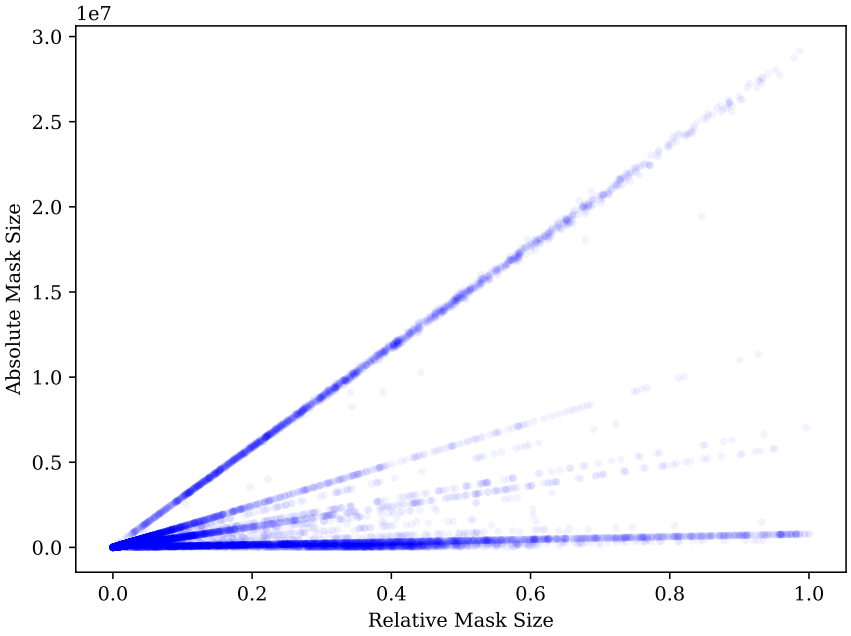


Figure 3.9.: Correlation between relative and absolute mask sizes

Cross-validation set containing the samples evaluated during training to measure changes in network performance and detect overfitting. The cross-validation set is further utilized post-training to compare different architectures and identify which samples were segmented insufficiently.

Test set containing samples used for the final analysis and interpretation of an architecture.

The presented partitioning scheme leads to multiple advantages. As the training and cross-validation sets do not overlap, overfitting generates a divergence between training and cross-validation errors which can be detected. As the test set is never used to determine possible network improvement points, any test set evaluations are unbiased.

Cross-validation can be performed in various ways [23]. Our cross-validation method statically assigns samples to a cross-validation set which stays the same for all different training runs and architectures. We do not utilize *k-fold cross-validation* as it leads to unsatisfactory training times due to the characteristics of our data.

3.3.1. Concept

As previously discussed, our data is partitioned into three disjunct static data sets. We utilize the quantitative data set analysis results to specify constraints in order to avoid bias in the training, cross-validation and test sets.

We initially discard all samples with an uncommon diagnosis and only use *Nevus*, *Melanoma* and *Seborrheic Keratosis* diagnosed lesions. All other diagnosis types are too rare to be learned and look different than most lesions. As our goal is to evaluate biomedical image segmentation using GANs, not to create a lesion segmentation network which works for all types of skin lesions, those minority diagnoses can be dismissed safely.

Another hard constraint applies to lesion images with multiple segmentation masks. Those images are not used in the test or cross-validation set as they distort the results. Multiple possible segmentation masks will always lead to an increased evaluation error even if the network matches one of the masks perfectly. The samples from lesion images with multiple segmentations are therefore always assigned to the training set.

Finally, all sets should contain lesion images and segmentation masks ranging over all analyzed features. Especially samples with rare features should be equally present in each set. If the training set is missing samples with uncommon features or does only contain few of them, generalization is hindered as most likely not enough data is present to learn segmentation of all kinds of lesions. If the test or cross-evaluation do not contain enough of the samples with less common features, failure to correctly segment those samples cannot be reliably detected. In the cross-evaluation set, this leads to wrong optimizations, in the test it might even result in a score which is better than the actual network performance.

Algorithm 2 shows how we concretely partition our data. The histogram of each feature is manually analyzed prior to running the algorithm. Manually choose the number of bins per feature. Each bin is then assigned a minimum number of samples which is chosen chose as $\min(\frac{\text{number of samples in bin}}{3}, 100)$

3.3.2. Resulting Sets

We applied the developed partitioning algorithm to our data. Table 3.2 contains the relative and absolute number of samples per data set. The cross-validation set is relatively small as its size heavily influences network training times. The chosen number of cross-validation samples is low enough to not retard training while still being high enough to provide statistically relevant evaluation data.

Table 3.3 displays high-level statistics over all three sets. It is observed that our algorithm leads to an equal unbiased distribution.

3.4. Data Management

Because our data set stems from different sources and requires homogenous meta data, a structured data organization and storage approch is required. We define a unified data and storage model which allows for various optimizations.

3.4.1. Data Model

Figure 3.10 shows a conceptual overview over the data model. Table B.5, Table B.6 and Table B.7 describe the attributes of `mole_samples`, `segmentations` and `set_entries`

```
input  : sample features, number of samples to put in test set, number of samples to
         put in cross-validation set, MinBinSize
output: test set, cross-validation set, training set, discarded samples
discarded samples ← discard samples not matching initial constraints;
training set ← samples for lesions with multiple masks;
FullfillContstraints(cross-validation set);
FullfillContstraints(test set);
while Size(cross-validation set) ≤ number of samples to put in cross-validation set
 do
    cross-validation set ← take random sample;
end
while Size(test set) ≤ number of samples to put in test set do
    test set ← take random sample;
end
training set ← remaining samples;
Procedure FullfillContstraints(set)
    foreach bin ∈ histogram bins do
        while CalcHistogram( set, histogram bins)[bin] ≤ MinBinSize[bin] do
            set ← take random sample;
        end
    end
    return;
```

Algorithm 2: Data Set Partitioning Algorithm

Data Set	Non-Discarded Samples	Sample Count
Training	70%	11382
Cross-validation	10%	1626
Test	20%	3252
Discarded	-	1262

Table 3.2.: Sample count per data set

Data Set	Plaster Images	Hair Images	Brightness	Contrast
Training	9801 (86.1%)	4925 (43.3%)	177.44/26.01	28.04/16.77
Cross-validation	1420 (87.3%)	705 (43.4%)	174.52/28.04	30.01/19.43
Test	2828 (87.0%)	1385 (42.6%)	176.86/26.87	29.04/17.66

Table 3.3.: High-level statistics per data set. Brightness and contrast each show mean and standard deviation.

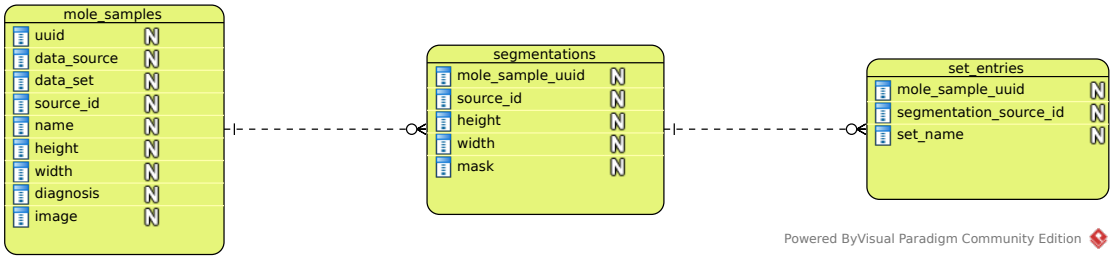


Figure 3.10.: Data model

respectively.

This data model enables

Re-importing from a source. The samples of a source can be identified by their `data_set` and `source_id` attributes. If matched against those, updating is possible without breaking references. The same holds true for segmentations with their `source_id` attribute.

Unified referencing of samples in the complete system. A `uuid` is generated at the first import, independent of the original sample source. Each sample is always referenced by its `uuid`, even if the sample is re-imported. Unified identifiers furthermore enable additional optimizations in the data pipeline as described in section 3.5.1.

Creation of multiple partitions over the complete data set. A partition only needs a unique name. Each partition can be used as a separate data set, a single segmentation can be in multiple data sets at the same time. This is not needed for the training, cross-validation and test split but enables the creation of smaller sets for debugging purposes.

3.4.2. Storage

During this work, various relational and NoSQL database management systems were evaluated. The requirements for a suitable database management system are as follows:

Storage of large binary data to enable storing images and segmentation masks. In particular, a single uncompressed skin image takes up $width \cdot height \cdot 3$ bytes which cannot be efficiently represented as text.

Storage of structured meta data to create a unified view over samples from the different data sources. Structure helps to avoid import errors and provides guarantees over the data format for all steps in the data pipeline.

Referencing entries of different types for linking lesion images to masks and masks to data sets.

A general problem is the storage of images as large binary arrays or *BLOBs*. We did not find any NoSQL databases capable of supporting our use case with out-of-the-box

solutions not suffering major performance impacts. Relational databases come with support for BLOBs but also experience performance degradation. We are using *PostgreSQL* as our database management system because our data model has relational structures, PostgreSQL performs reasonably well with BLOBs and does not need special configuration to work in our use case.

We do not pursue a filesystem based approach because having images on the filesystem and meta data in a database management system requires manual management of the references between filesystem and database. Moreover, a completely file-based approach does not allow the storage of structured meta data. This would introduce too much organziational overhead compared to the possible performance improvements.

Table B.8 contains the mapping of conceptual types to PostgreSQL types. Table B.9 lists all primary key constraints. Finally, listing C.1 shows the SQL required to create the full database schema.

3.5. Data Processing

How data is served to a deep learning algorithm has a big impact on the training performance [24]. Training with large images requires considering memory implications as a single sample already takes up multiple megabytes. The goal of the input pipeline is to keep the graphics card utilized 100% by ensuring no time is spent waiting for new data.

Our pipeline from the database to the graphics card is optimized to be as fast as possible within the memory limitations imposed by the hardware. As loading samples directly from the database requires slow network access, the pipeline is divided into two parts. The first part loads samples from the database to the local filesystem and stores them in an efficient way. The second part loads the samples from the filesystem to the graphics card.

3.5.1. Database Pipeline

As previously discussed, loading samples over the internet is a time consuming procedure. To improve it, we load the samples once for each training machine and store them on its local filesystem. This step is only performed once, minimizing the computational overhead imposed by preprocessing steps and further improving training speed.

Samples are stored in the *Example protocol buffer* format because TensorFlow utilizes it as the default and provides built-in functionality for saving and loading. Protocol buffers already contain serialized matrices, obviating the need to decode images during training, resulting in a further speed improvement. The values of the stored image and mask matrices are further converted into the range of the *hyperbolic tangent* (*tanh*), the input scale of our network.

We store each sample in a separate file. This is against the best practices [25] which recommend utilizing as large files as possible to avoid filesystem induced bottlenecks. We do not follow best practices because it makes random shuffling training samples impossible in our use case. As protocol buffers do not support random access, shuffling their entries is done by loading as many entries as possible into memory and shuffle them

in-memory. A single one of our data points already consumes a fairly large amount of memory which does not allow having many samples loaded at the same time. Therefore, we cannot utilize the in-memory shuffling method and have to resort to shuffling the sample ids.

Keeping thousands of files in a single directory degrades file access performance. We utilize a special directory structure inspired by the Git object storage model [26] to avoid filesystem induced performance impacts. Each protocol buffer is stored in a subdirectory, named after the first byte of the lesion's 128 bit random UUID. As the UUIDs are uniformly distributed, this leads to 256 subfolders which each contain approximately $\frac{1}{256}$th of each set, a size easily manageable by most modern filesystems.

Algorithm 3 shows the complete loading process.

input : data set, should rescale
output: Data set entries and list of ids stored on the filesystem
segmentation ids, lesion uuids ← load all lesion and segmentation ids from data set
foreach current lesion uuid ∈ lesion uuids **do**
  lesion image matrix ← load and deserialize lesion image for current lesion uuid;
  segmentation matrices ← load and deserialize all segmentations associated to current lesion uuid;
  if should rescale **then**
    lesion image matrix, segmentation matrices ← rescale all images;
  end
  lesion image matrix, segmentation matrices ← convert values into *tanh* range $[-1.0, 1.0]$;
  foreach segmentation matrix ∈ segmentation matrices **do**
    file name ← `Concatenate`(*first two bytes of* current lesion uuid *as hex, '/',* current lesion uuid*, '_', id of* segmentation matrix*, '.tfrecord'*);
    write protocol buffer containing (lesion image matrix, segmentation matrix) → file name;
    write `Concatenate`(current lesion uuid*, '_', id of* segmentation matrix) → meta data file;
  end
end

Algorithm 3: Database Pipeline Algorithm

3.5.2. Graph Pipeline

The graph pipeline handles sample transfer from the filesystem to the actual execution of our network. Samples are loaded from their protocol buffer files, transformed on the fly, grouped in batches and put into a queue. Transformation optionally consists of color space conversion and data augmentation. The goal is to always keep the queue above a minimum number of entries as otherwise the graphics card has to wait for new samples. It is also important to minimize context switches between the CPU and graphics card as

each switch takes some time. We therefore perform all loading and adjustment tasks on the CPU so the graphics card is fully used for matrix multiplication tasks.

Input pipelines in TensorFlow heavily utilize multi-threaded queues [27]. A thread dequeues a sample from one queue, performs operations on the data, and enqueues the result into another queue. The TensorFlow input pipeline pattern allows for scalability, is supported throughout the library and therefore well suited for our input pipelines.

The pipeline used during training needs to be as fast as possible, shuffling the samples and augmenting them. On the contrary, the pipeline used for evaluation can be slower but must provide an absolutely deterministic order of samples. This results in the choice of using two different pipelines, one for training and one for evaluation.

As previosuly discussed, the top goal of the training pipeline is to optimize for graphics card utilization. This is achieved with the following process:

1. The full list of sample ids of a training set is provided to the input queue.
2. The actual file names of the sample protocol buffers are reconstructed from the ids.
3. Those filenames are then continuously shuffled and put into the file name queue. This queue allows for an inifinite number of dequeuing operations while guaranteeing that, each epoch, all file names are dequeued once. The file name order is randomized every epoch.
4. A configurable number of threads is then started. Each thread repeatedly
   a) Dequeues a single file name from the file name queue.
   b) Reads the protocol buffer sample from disk.
   c) Deserializes the stored lesion image matrix and segmentation mask matrix.
   d) Applies a configured list of augmentation functions to the lesion and/or the segmentation mask. Data augmentation is discussed in section 3.5.3.
   e) Converts the samples into a configured color space.
   f) Enqueues the transformed pair into the batch queue.
5. The images from the batch queue are finally grouped into sample batches and provided as the result queue.

This queue design optimizes performance as samples are loaded from disk in parallel. If slow tasks like data augmentation or color space conversion result in an occasionally empty result queue, the number of threads can be increased. The resulting sample order is deliberately random because the ids are shuffled and the order in which the threads insert their samples into the batch queue is undefined. This helps preventing overfitting of the network but results in the side effect that a single sample can be present in the same epoch twice. Because many epochs are performed and the sample count is high, this effect can be ignored.

The evaluation pipeline shares most parts with the training pipeline. However, as a deterministic order of samples in the resulting batches is more important than performance,

some tradeoffs are made. All connections between queues in the evalution pipeline utilize only one thread each. As the ids are not shuffled between epochs, this guarantees that the output order of samples is always the same between epochs and even different runs. Furthermore, no data augmentation is performed which allows for a comparable deterministic segmentation performance analysis.

The process of the evaluation pipeline is as follows:

1. The full list of sample ids of a training set is provided to the input queue.
2. The actual file names of the sample protocol buffers are reconstructed from the ids.
3. Those filenames are then continuously put into the file name queue. This queue allows an inifinite number of dequeuing operations while guaranteeing that, each epoch, all file names are dequeued exactly once. The initial order of file names is preserved.
4. A single thread repeatedly
   a) Dequeues a single file name from the file name queue.
   b) Reads the protocol buffer sample from disk.
   c) Deserializes the stored lesion image matrix and segmentation mask matrix.
   d) Converts the samples into a configured color space.
   e) Enqueues the transformed pair into the batch queue.
5. The images from the batch queue are finally grouped into sample batches and provided as the result queue. The batching can happen with one or more threads. If multiple threads are used, the order of samples in a single batch will be nondeterministic, however, the samples contained in a single epoch still stays deterministic.

It has to be noted that, due to the TensorFlow graph model, both resulting queues can also be used to perform training and evaluation on the CPU.

3.5.3. Data Augmentation

As observed during the analysis of our data set, there are no standards for lesion image quality. This leads to the conclusion that, to be usable in practice, our network needs to be mostly invariant against specific properties of image acquisition like brightness and contrast levels. Such an invariance also promotes learning higher level features instead of simple brightness patterns. To achieve such goal, various data augmentation techniques are used. They are applied to each image every time it is used in training. Data augmentation slightly transforms every image to increase the variance of the augmented features [28]. Higher variance requires the network to become invariant against the augmented features in order to minimize the objective function.

The following data augmentation techniques are used:

Random rotate and flip an image. We use a technique which allows all possible transformations and mirrorings without requiring interpolation. This results in not losing any information while still promoting rotation invariance. First, the image is rotated using a rotation angle $\varphi \in \{0°, 90°, 180°, 270°\}$, randomly choosen. Afterwards, with a probability of 50%, the image is horizontally mirrored.

Random brightness adjustment is applied to an image. Two parameters have to be provided: Δ_{min} and Δ_{max} specifying the minimum and maximum adjustment respectively. A $\Delta \sim U[\Delta_{min}, \Delta_{max}]$ is randomly chosen and added to each channel of each pixel. This results in a global brightness increase (if Δ is positive) or decrease (if Δ is negative). The channel values are then clipped to avoid value overflow and underflow.

Random contrast adjustment is applied to an image. Two parameters have to be provided: Δ_{min} and Δ_{max} specifying the minimum and maximum adjustment respectively. A $\Delta \sim U[\Delta_{min}, \Delta_{max}]$ is randomly chosen as the adjustment. Afterwards, the global means μ_r, μ_g, μ_b for each channel are calculated. The contrast adjustment is then applied to each pixel x as $x_{channel} = (x_{channel} - \mu_{channel}) \cdot \Delta + \mu_{channel}$. This results in a global contrast adjustment where $\Delta > 1$ increases contrast and $\Delta < 1$ decreases contrast. The channel values are then clipped to avoid value overflow and underflow.

4. Base Architecture

Biomedical image segmentation denotes the task of transforming a lesion image into a segmentation mask, thus qualifying it as an image-to-image translation problem. The adversarial approach utilizes GANs in a conditional setting by providing input images as conditions [29]. An alternative approach additionally uses input images as generator input while omitting random vectors completely. This way, a surjective mapping between input and output distribution is learned. We use the first approach to define a base architecture and explore the latter approach in our extensions.

The base architecture serves two purposes: Firstly, it allows us to discover if image segmentation is possible in our scenario. To the best of our knowledge, we are the first to explore semantic segmentation using GANs while completely bypassing supervised loss functions, relying solely on a discriminator output to train a generator. Secondly, the base architecture serves as foundation for architectural expansions so their impacts on the segmentation performance can reliably be compared.

4.1. Network Architectures

We derive our base generator and discriminator from an existing model known as *Pix2Pix* [29]. Pix2Pix is engineered to be a general-purpose image-to-image translation GAN. It applies a modified version of the *U-Net* architecture [30] as its generator. This serves our purpose well as the U-Net architecture is tailored to biomedical image segmentation.

The base architecture operates on images with a fixed size of 512×512 pixels. In principal, this limit could be lifted to arbitrary spatial extents. We chose it to ease the development of a proof-of-concept architecture and reasoning about intermediate results.

Figure 4.1 displays a high-level overview of the base architecture. Section A.1 lists all base architecture generator and discriminator layers in detail.

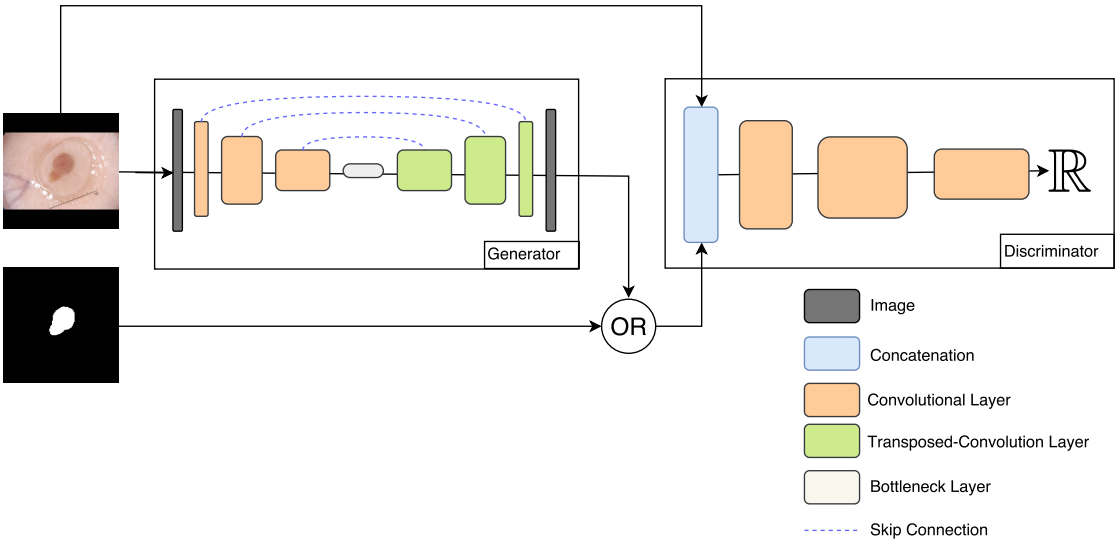


Figure 4.1.: Base architecture overview. Some intermediate layers are omitted for brevity.

4.1.1. Generator Network

The base generator follows an encoder-decoder structure [12]. Initially, an input image is provided in its full spatial size with a feature map consisting of the three color channels. A number of encoder layers half the spatial extents until a bottleneck size is reached. We choose 1×1 pixels as our bottleneck size. An equal number of decoder layers each double the spatial extents again until the original size is reached. The number of feature maps increases respectively decreases inversely proportional to the spatial size of each layer. The encoder acts as a feature extractor, combining local image statistics into global ones. The decoder then recreates a representation of the original image based on its global statistics. We use strided convolution for downsampling and strided transposed convolution for upsampling [12]. Leaky ReLu [31] with a slope $\alpha = 0.2$ is applied after each encoder layer, ReLu [32] after each decoder layer except the last one. The hyperbolic tangent function (tanh) [33] is applied to the output layer to produce values in the range $(-1, 1)$. Tanh is preferred over sigmoid as an activation function due to it being zero centered, thus avoiding bias shift. Batch normalization [34] is performed in each layer except for the first and last encoder layers and the last decoder layer.

The encoder-decoder structure is further extended with skip connections [35]. Skip connections concatenate the feature maps from each encoder layer to its corresponding decoder layer. This eliminates the need to learn an identity mapping as the original features from each encoder layer are directly provided to their decoder equivalents. As learning an identity mapping becomes harder the deeper a network, utilizing skip connections accelerates training and produces better results [35].

Due to the high dimensionality of images used as conditions, accompanying lower-dimensional input vectors are mostly ignored by generator networks [29]. We use dropout [36] in the first two decoder layers during training and inference as an alternative to intro-

duce random entropy. Dropout is directly applied to the connections between neurons, thus ignoring it becomes almost impossible.

All convolution inputs in the generator and discriminator are zero-padded. This allows for simple skip connections as the spatial extents are equal for each encoder-decoder level.

4.1.2. Discriminator Network

The discriminator follows a straightforward architecture. Initially, the original input image's channels are concatenated to either the ground truth or generated segmentation mask. Each discriminator layer then performs a strided convolution with a stride of 2, followed by a leaky ReLu activation. After a spatial size of 2×2 is reached, a final convolution is applied to produce a single value. No batch normalization is performed in the discriminator.

As our input images in the base architecture exhibit constant spatial sizes, they can reliably be downsampled to a single value without using special pooling techniques. This results in a discriminator with a receptive field size encompassing the full input, enabling it to discriminate based on global statistics.

4.2. Loss Function

We modify the usual GAN objective into a *Wasserstein GAN with gradient penality* objective [37]. This introduces a new loss function which adds an additional term to the discriminator loss. Each training step, the discriminator gradient with respect to a combination of real and generated samples is calculated. The squared difference between the gradient norm and 1 is then scaled by a parameter λ_{gp} and added to the discriminator loss.

Equation (4.1) shows the modified discriminator loss function defined in [37]. D refers to the discriminator network with fixed weights, $\tilde{x}$ and x to generated and real samples respectively. $\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}$ is a random combination of generated and real samples where $\epsilon \sim U[0, 1]$ is randomly chosen for each pair of $\tilde{x}$ and x.

$$L \leftarrow D(\tilde{x}) - D(x) + \lambda_{gp} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2 \tag{4.1}$$

The additional term is further referred to as *gradient penalty* because it penalizes exploding and vanishing gradients. Gradient penalty restricts the change in discriminator parameters between training iterations and thus forces a lipschitz constraint on the discriminator [37]. This results in more stable training as the discriminator fluctuates less and provides more consistent feedback to the generator.

The original Pix2Pix model further introduces an additional L_1-distance term to the generator loss. We omit this additional term in our base architecture as it would reduce discriminator impact on generator training and thus distort our evaluation.

4.3. Experiments

We train the base architecture networks with stochastic mini-batch gradient descent and the *Adam* optimizer [38]. We choose a batch size of 1 due to memory limitations. Adam is used because it outperforms similar optimizers in most cases and renders learning rate tuning almost superfluous.

We parameterize our optimizer with $\alpha = 0.0001$, $\beta_1 = 0.5$ and $\beta_2 = 0.9$ where α represents the learning rate, β_1 and β_2 the 1st and 2nd moment estimates respectively.

The network is trained for 25000 iterations, each performing 5 discriminator updates followed by 1 generator update. We use a new sample in each update to prevent overfitting and support generalization.

4.3.1. Evaluation

The evaluation of our models focuses on quantitative metrics. We calculate and average those metrics over both the test and cross-validation sets. This enables a comparison of various performance properties between the base architecture and extensions. We can further identify samples which are segmented particularly bad.

The cross-validation set metrics are used to evaluate and propose further steps in the development of our networks. In order to avoid bias, the test set metrics are solely used to retrospectively compare different models.

All evaluation metrics compare a ratio between positive and negative classifications. We determine the generated class for each pixel using a simple thresholding operation. A pixel is counted as positive if its value is greater than 0.5, negative otherwise. This threshold has to be adjusted for pratical application scenarios.

We utilize the following evaluation metrics:

Accuracy is the ratio between correctly segmented pixels and all pixels. It provides and overall picture of how well an image is segmented.

Precision is the ratio between true positives and all generated positives. The precision indicates how precise the ground truth mask is matched.

Recall is the ratio between true positives and ground truth positives. The recall indicates how much of the ground truth mask is recalled.

Specificity is the ratio between true negatives and all negatives.

F_1 Score combines precision and recall into a single metric, defined as $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

Jaccard Similarity Coefficient is the ratio between ground truth and generated mask intersections and their union. Let A be the set of positive ground truth pixels, B the set of positive generated pixels. Jaccard similarity is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

All metrics produce scalar score values in the range $[0, 1]$. Higher scores indicate better performance than lower ones.

Metric	Cross-Validation Set	Test Set
Accuracy	0.916	0.926
Precision	0.679	0.666
Recall	0.528	0.551
F_1 Score	0.484	0.498
Specificity	0.987	0.986
Jaccard Similarity	0.384	0.399

Table 4.1.: Base architecture evaluation metrics

Some metrics lead to deeper insights than others. Accuracy provides little information due to most parts of a segmentation mask being negative. Classifying each pixel as skin already achieves accuracy scores around 90%. Precision, recall and specificity scores provide valuable hints about the segmentation strategy utilized by a model. Over time diverging precision and recall can further indicate architectural problems. The F_1 score balances precision and recall, thus providing a general idea of the overall results. Jaccard similarity is often specifically used to evaluate segmentation results as it directly measures the similarity between two masks. Thus, the jaccard similarity coefficient provides direct feedback regarding generated mask quality.

Table 4.1 lists the discussed metrics for the cross-validation and test sets.

A general problem in GAN training is non-interpretable loss development. Figure 4.2 displays generator and discriminator loss by the number of iterations on both training and cross-validation sets. The loss values are heavily smoothed and the scaling is adjusted to ignore outliers. Non-smoothed loss values are semi-transparently plotted in the background. Coincidentally, segmentation quality for a single sample can heavily oscillate between training iterations. Figure 4.3 lists the generated masks of a single sample after various iterations. We choose iteration 18000 for this evaluation as it exhibits the best overall scores.

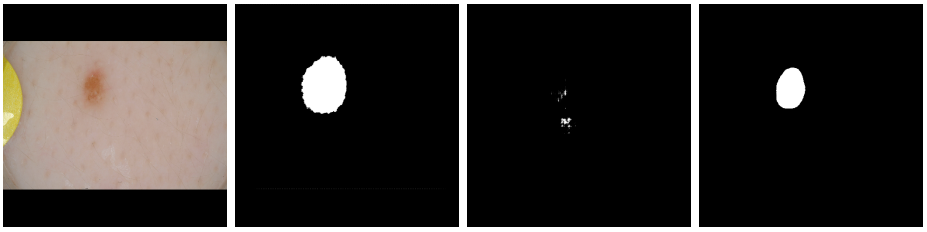


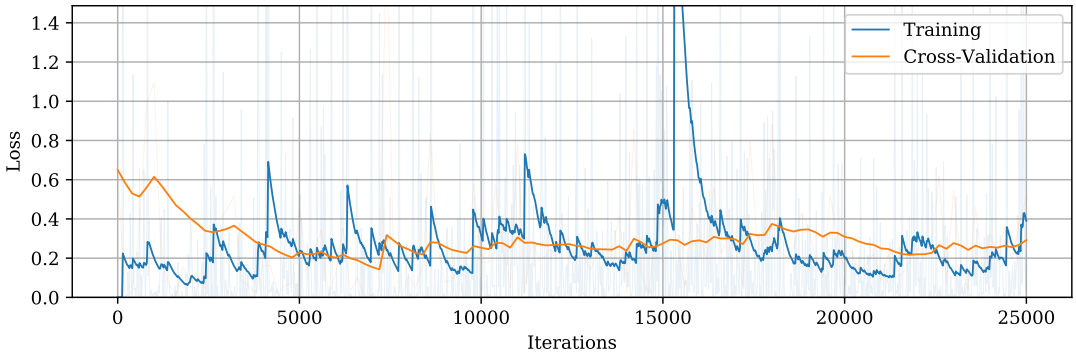
Figure 4.3.: Generated masks by number of iterations. The first image shows a lesion, the others generated masks after 4000, 14600 and 19000 iterations respectively.

4.3.2. Conclusion

The metrics previously displayed in table 4.1 provide overall insights to the base architecture performance. Accuracy scores over both cross-validation and test set are high as

(a) Generator loss

(b) Discriminator loss on real images



(c) Discriminator loss on generated images

Figure 4.2.: Loss development by number of iterations

hypothesized. This, combined with specificity scores over 98%, provides a hint that the general idea of segmentation is learned by our network.

We from now on display generator output as four images. The first image displays the lesion image as provided to the generator. The second and third image encode the ground truth and generated masks respectively. Finally, the fourth image visualizes the difference between ground truth and generated mask. False positives are colored red, false negatives blue. Black pixels therefore indicate areas correctly classified by the generator.

Figure 4.4 displays a sample which is segmented almost perfectly. As the corresponding ground truth segmentation mask is upsampled from a lower resolution, this also indicates that the network is capable of generalization and thus provides a subjectively better matching mask than the ground truth.

Even though the general concept of segmentation is learned, many issues are still prevalent. Low recall scores indicate that the network generally underestimates lesion sizes. As skin pixels form a majority in most segmentation masks, false negatives are harder to detect than false positives. Thus, the generator fools the discriminator more easily by producing negative classifications. Figure 4.5 displays an image containing no lesion. Our network, preferring negative classifications, matches the empty ground truth mask perfectly, resulting in 100% precision and recall. Similarly, uncommonly large lesions are matched very poorly. Figure 4.6 shows one of the largest lesions in our cross-validation set. The generated segmentation mask covers less than one fourth of the ground truth mask. As no clear correlation between generated mask and lesion image can be seen, this further hints at a failure to generalize over large lesions.

Low precision scores can mostly be attributed to plaster presence in lesion images. Because the generator underestimates lesion sizes, it only classifies those pixels as positive which can be assigned to a lesion with high confidence. However, if the generator fails to distinguish between plasters and lesions, it perceives both as anomalies. This results in positive classifications of plaster images and thus, low precision scores. Figure 4.7 contains a sample where the lesion is completely overlooked and the plaster is masked instead. As the generated and ground truth masks do not overlap at all, the presented sample leads to 0% precision, recall and jaccard scores. We detect this behavior in approximately 9.4% of our cross-validation set samples.

In conclusion, the base architecture serves as proof-of-concept that learning a supervised task is not impossible by solely using adversarial loss. However, the resulting network is far behind other state-of-the-art models and unusable in practical application. We therefore propose and evaluate extensions based on this evaluation in order to improve the segmentation results.

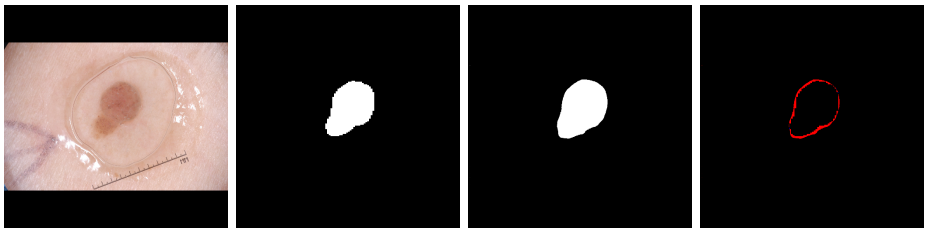


Figure 4.4.: More accurate generated mask than ground truth

Figure 4.5.: Empty mask

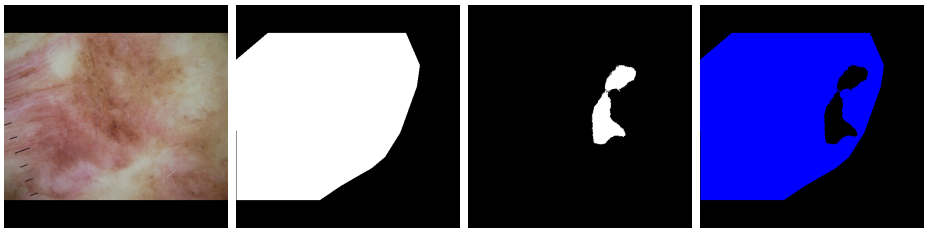


Figure 4.6.: Uncommonly large lesion

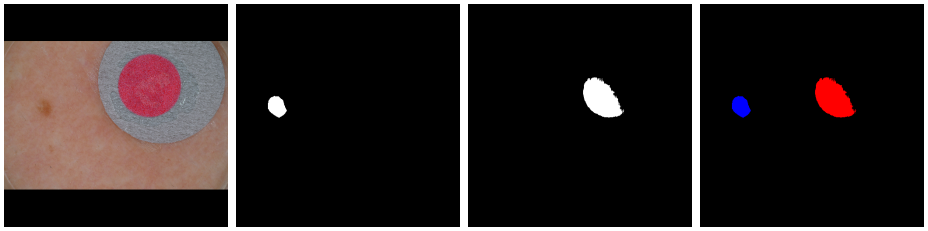


Figure 4.7.: Complete mismatch between generated and ground truth masks

5. Architectural Extensions

In this chapter, three base architecture extensions are proposed. We train and evaluate a model for each extension, comparing the results afterwards. The gained insights are used to propose a final architecture.

5.1. Enhanced U-Net Generator

Failure of the generator to distinguish plasters and other artifacts from lesions can be ascribed to various reasons. If the generator lacks global context information, it might not reliably differentiate between plaster and non-plaster regions. Furthermore, if the architecture restricts learning capabilities too much, the generator might not learn a semantic image understanding.

As the base architecture allows the generator to capture the complete image, absence of global context is deemed unlikely. We thus explore if increased capabilities improve segmentation results.

An increase in the number of feature maps intuitively leads to better segmentations as the presence of more parameters allows a neural network to learn more complex functions. Furthermore, as all generator layers change the spatial extents of their input, they need to perform resampling and feature extraction in a single step. Splitting this process into dedicated sampling and feature extraction layers simplifies the learning process, potentially resulting in the approximation of a more adequate mapping function.

We propose an enhanced generator architecture which follows the original U-Net more closely.

5.1.1. Enhancements

Figure 5.1 shows the original U-Net architecture in detail. Each blue box correspond to a multidimensional feature map where the spatial extents are printed on the left, the number of feature channels on top. White boxes denote concatenated features, blue dashed lines a cropping operation.

The base architecture applies the following modifications to the original U-Net architecture:

- For each level, multiple features maps are reduced to a single one.
- The bottleneck feature map is removed.
- The top level skip connection is omitted as no convolution is performed on the last discriminator level.

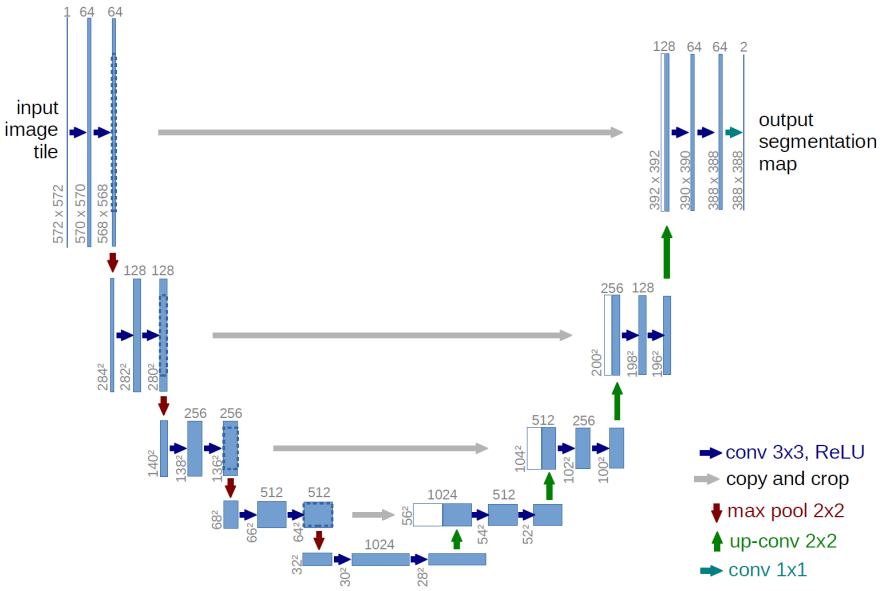


Figure 5.1.: U-Net architecture, taken from the original paper [30]

- Max pooling is replaced with strided convolutions.
- Zero padding is applied to keep the spatial extents constant.

We modify our base architecture to resemble the original U-Net more closely and thus re-introduce promising constructs. Combined with additional extensions, the following changes are proposed:

- A constant number of convolutions on each encoder and decoder level is performed prior to resampling.
- An additional 1×1 convolution is applied after the final decoder layer to produce results.
- The bottleneck feature map which connects encoder and decoder is re-introduced.
- Add an additional skip connection on the top level is added.
- Zero padding is replaced with mirrored padding.
- Dropout as a means to introduce random entropy is removed from the decoder.
- Batch normalization is performed after every layer.
- The bottleneck layer size is 32×32 pixels instead of 1×1.

As previously discussed, combining image resampling and feature extraction into a single step might hinder semantic learning. In the enhanced architecture, the additional

convolution layers on each level are not used for resampling. This allows them to learn parameters solely utilized for feature extraction. Likewise, strided convolution layers are not directly concerned with feature extraction anymore, thus more optimized resampling parameters can be learned.

Adding an extra 1×1 convolution for output generation, together with a skip connection on the highest level, might lead to finer segmentations because high level image representations can be directly combined with the original image. In contrast, the reintroduction of a bottleneck feature map allows the network to better operate on dense global image representations which might improve the localization of artifacts and lesions.

Zero padding keeping the generator's spatial shapes consistent might be a reason for bad segmentations around the border regions. Mirror padding reflects the image along all four borders. This combines the advantages of padding while removing the need for the generator to learn special image boundary handling.

As mentioned before, dropout is applied to the first two base architecture decoder layers to introduce random entropy during mask generation. In our case, it is sufficient if the generator always produces the same segmentation mask for the same input image. This makes random entropy superfluous. By removing dropout, the generator learns a direct mapping from a lesion image distribution to a segmentation mask distribution.

The two final enhancements are of a purely practical nature. We did not note any significant difference between omitting batch normalization or performing it after each layer. Thus, to decreas network training times, we perform batch normalization everywhere. Furthermore, the introduction of multiple convolutions per encoder-decoder level significantly impacts training speed and memory usage. We compensate this by omitting encoder and decoder layers with a spatial resolution lower than 64×64 pixels, resulting in the 32×32 pixels bottleneck size.

Figure 5.2 displays a high-level overview of the enhanced generator. Section A.2 lists all layers and their properties.

5.1.2. Experiments

We train the enhanced generator combined with the base architecture discriminator for 19000 iterations. The same optimizer parameters as in the base architecture are used. We choose iteration 10000 for the evaluation and comparison because the trend of performance metrics hints at model instabilities. Figure 5.3 displays the recall development over time. It can easily be seen that, after a short stable period, the recall steadily decreases.

Table 5.1 displays the evaluation metrics compared to the base architecture. All cross-validation scores of the enhanced generator are lower than their base architecture equivalents. While in most cases, scores between the two architectures differ less than 1%, the recall dropped significantly. This indicates that the enhanced and base architecture generators exhibit an equal overall performance but the ehanced generator cannot reliably be trained. A 4% decrease in Jaccard similarity correlates the decreased recall scores.

The percentage of generated masks not matching their related ground truth in our cross-validation set is lower than 1%. Even though this significantly is less compared to

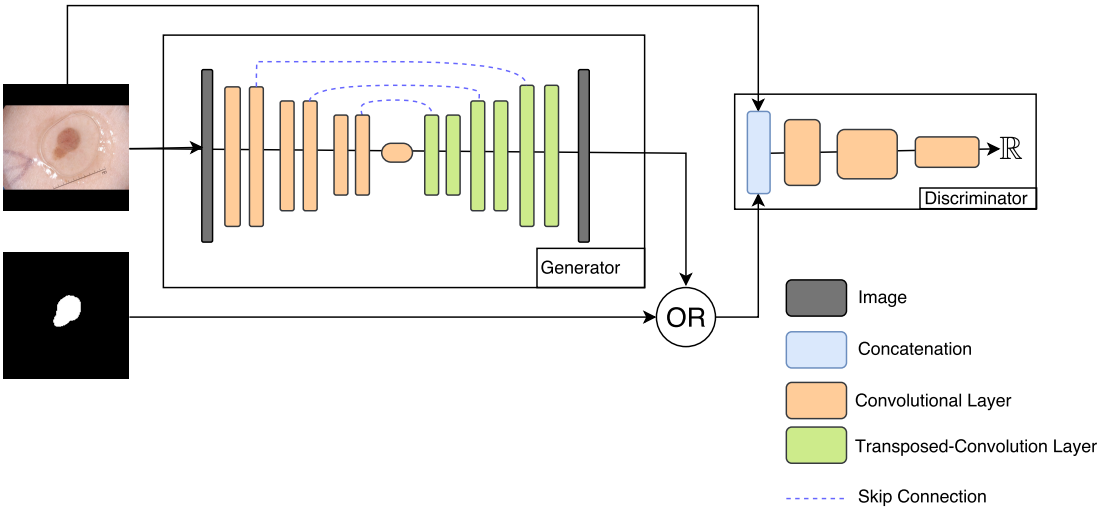


Figure 5.2.: Enhanced generator architecture overview. Some intermediate layers are omitted for brevity

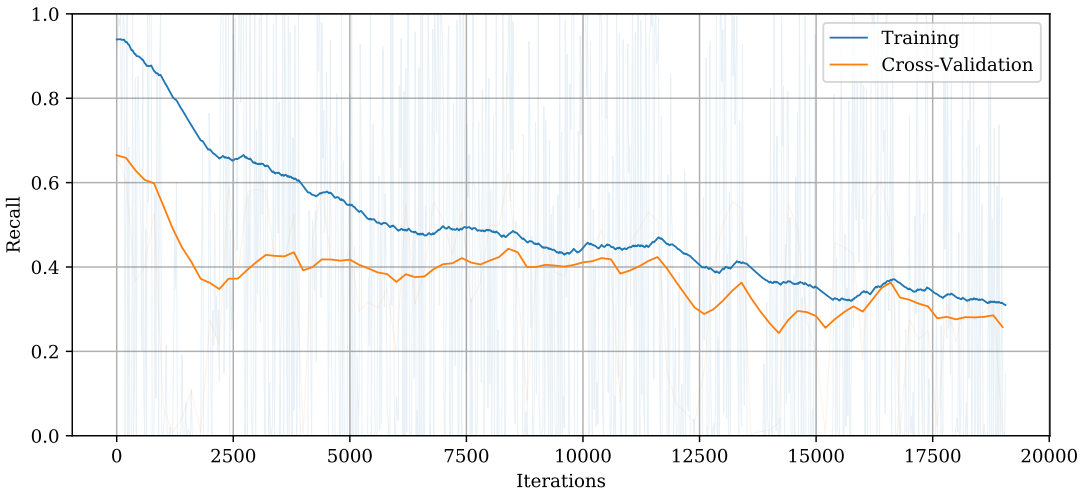


Figure 5.3.: Enhanced generator recall development by number of training iterations

	Enhanced Generator		Base Architecture	
Metric	Cross-Validation Set	Test Set	Cross-Validation Set	Test Set
Accuracy	0.912	0.923	0.916	0.926
Precision	0.667	0.668	0.679	0.666
Recall	**0.465**	0.482	**0.528**	0.551
F_1 Score	0.465	0.480	0.484	0.498
Specificity	0.989	0.989	0.987	0.986
Jaccard Similarity	**0.344**	0.355	**0.384**	0.399

Table 5.1.: Enhanced generator evaluation metrics

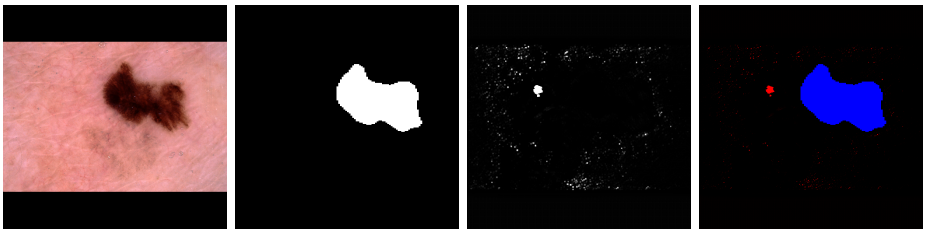


Figure 5.4.: Noisily segmented lesion

9.4% in the base architecture, we identified many other samples with a Jaccard similarity index lower than 1%. Figure 5.4 displays an example, the generated mask is dilated to better visualize fine noise. The generated segmentation mask does not optically correlate with the input image and contains widely distributed noise. This noise overlaps randomly intersects the ground truth mask and thus results in non-zero Jaccard similarity. While the base architecture generator seems to learn a semantic understanding of lesion images, the enhanced generator interprets the image in a wrong way, focusing on non-existing features. We detected this behavior in approximately 13.4% of our cross-validation samples, correlating with the decreased recall score.

We attribute the slightly aggravated overall segmentation performance to the limited amount of training iterations. Training for only 10000 iterations intuitively provides an insufficient amount of data points to the model and thus makes learning a semantic image understanding impossible. However, the steadily decreasing recall scores on the cross-validation and training sets also implies that performing more training iterations would not lead to a better result.

A reason for this model collapse might be the imbalance between generator and discriminator capabilities. The enhanced generator, exhibiting extended capabilities, might learn basic image segmentation after fewer iterations than its base architecture counterpart. In this case, the base architecture discriminator might not be able to differentiate between generated and real samples anymore, effectively preventing the discriminator from learning. As the generator solely learns from discriminator output, the optimization objective becomes invalid, resulting in decreasing generator performance.

Based on those observations, we conclude that the enhanced generator has to be combined with further model extensions in order to be reliably evaluated.

5.2. Enhanced Segmentation Discriminator

The base architecture discriminator issues are twofold: Firstly, the aforementioned lack of capacity might not only be prevalent in the generator but also in the discriminator. Secondly, in contrast to the generator, the base architecture discriminator is not designed with segmentation in mind.

Our discriminator computes the conditional probability of a mask stemming from the training data given the input image. Therefore, the discriminator must extract features which allow it to assess if the condition is met and the mask lookes realistic in the first place. Additionally to learning feature extraction, it needs to learn how to combine features into a dense representation during downsampling.

In the base architecture, the discriminator reduces the dimension on every layer by a factor of two. Earlier convolutional neural networks relied on pooling layers for downsampling, such as max-pooling or average pooling [39][40]. Strided convolution is considered superior [41] as it allows a network to learn custom downsampling techniques. However, this requires the dedication of network capacity to sampling parameters.

The base architecture often included large plasters in the segmentation mask. A good discriminator would output a high loss for such masks and force the generator to ignore plasters. As this is not happening, we conclude that the base architecture discriminator fails to differentiate plasters and lesions. Similarly to the base architecture generator, such a lack of semantic understanding might be attributed to missing capacity because the discriminator performs downsampling and feature extraction in a single step.

We propose an enhanced discriminator architecture, inspired by [7], which is tailored to image segmentation.

5.2.1. Enhancements

Based on the insights gained in [7], we propse the following enhancements to the base discriminator.

- The original mask and the lesion image are multiplied to produce a mask with additional color channels.
- The original image and the mask are convolved separately prior to being concatenated.
- An additional non-strided convolution is preformed before each down-sampling step.
- A 1×1 convolution is performed to produce a probability.

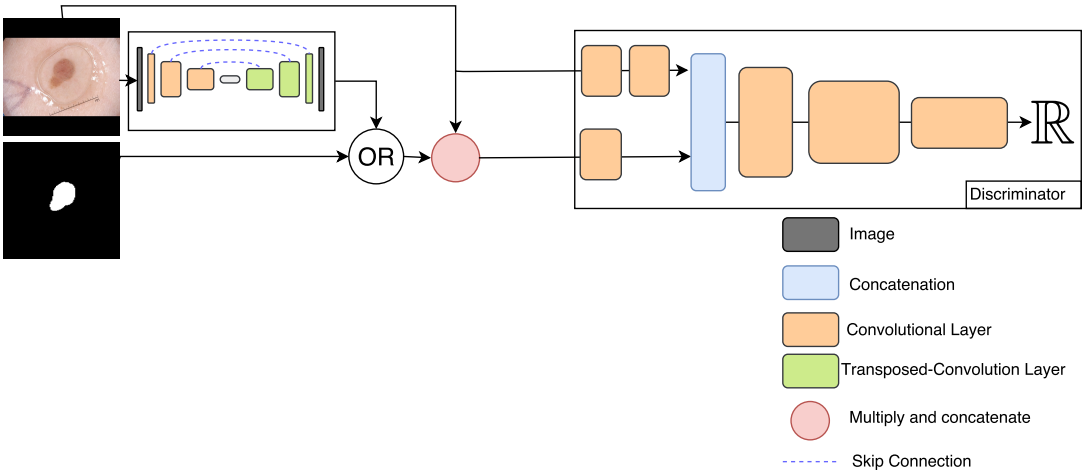


Figure 5.5.: Enhanced discriminator architecture overview. Some intermediate layers are omitted for brevity.

Mask and lesion image multiplication is an additional preprocessing step to aid the discriminator. Empirical experiments in [7] resulted in minor performance improvements. We additionally concatenate the original mask because otherwise the generator could label black regions of the lesion image arbitrarily without the discriminator noticing. Thus, the resulting input exhibits four channels.

The original input image is convolved twice, the mask once. This results in two separate branches which are afterwards concatenated. The reason for this is twofold: Firstly, inputs fed to the discriminator should have similar feature map depths as otherwise the higher-dimensional inputs may have an unfairly large impact [42]. Secondly, a more useful intermediate low-level representation of the mask and input image can be learned.

The concatenated features are repeatedly convolved with a 3×3 filter and downsampled by a strided convolution with filter size 3×3 and a stride of 2. The difference to the base architecture is the additional convolution before downsampling, following the same philosophy as the enhanced generator to give each layer more capacity so it learns higher-order statistics.

At the end, a 1×1 convolution produces real-valued outputs representing the probability of a mask being real.

Figure 5.5 shows an overview over the enhanced discriminator architecture a detailed list of the layers is in table A.4.

We expect the enhanced discriminator to be more sensitive towards plasters, punishing the generator for classifying them as lesions.

5.2.2. Experiments

We train the enhanced discriminator architecture for 20000 iterations with the usual set of optimizer parameters.

	Enhanced Discriminator		Base Architecture	
Metric	Cross-Validation Set	Test Set	Cross-Validation Set	Test Set
Accuracy	0.904	0.913	0.916	0.926
Precision	0.681	0.669	0.679	0.666
Recall	**0.331**	0.335	**0.528**	0.551
F_1 Score	0.376	0.382	0.484	0.498
Specificity	0.986	0.985	0.987	0.986
Jaccard Similarity	0.344	0.291	0.384	0.399

Table 5.2.: Enhanced discriminator evaluation metrics

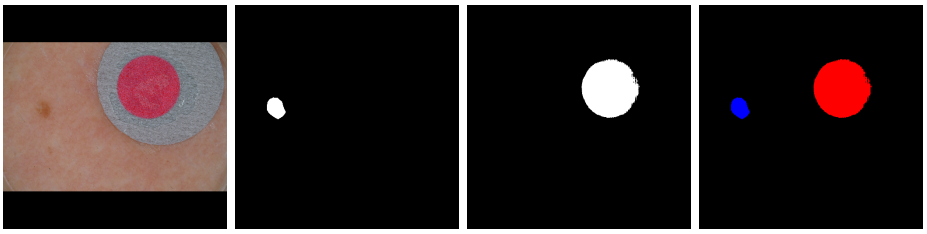


Figure 5.6.: Segmentation mask only capturing plaster

Table 5.2 lists the quantitative results of the enhanced discriminator architecture next to the base architecture. We choose iteration 15000 for the comparison as it exhibits the best scores. Unexpectedly, the recall is much worse, resulting in a lower F_1 score as well. This implies that masks containing only a fraction of the lesion are much more readily accepted by the discriminator. Figure 5.7 shows one of many examples.

Figure 5.6 shows an lesion image with a large plaster, contrary to our expectations the results are just as bad as with the base architecture. The generator is allowed to completely ignore the lesion and segment the plaster.

Figure 5.6 shows an lesion image with a large plaster. Contrary to our expectations, the results are equally bad as with the base architecture model. The generator completely ignores the lesion while masking the plaster.

This indicates that either the enhanced discriminator was not better fit for the tasks than the base discriminator, or it was not trained for enough iterations. Enhanced

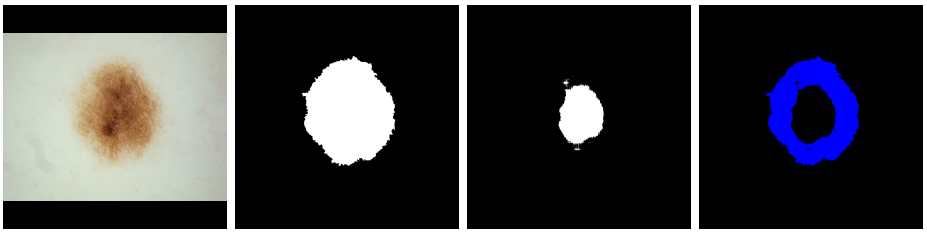


Figure 5.7.: Severely underestimated lesion size

discriminator training takes roughly twice as long compared to the base architecture. This comes as no surprise since multiple additional convolutions are performed on the full spatial size of any input image. Due to time constraints, we are unable to explore effects of longer training.

5.3. Context-Aware Discriminator Loss

Traditionally, GAN discriminators are trained to classify all generated samples equally as fake. In unsupervised representation learning, this is the sole possibility as no other loss function for generated samples exists.

However, this loss approximation of generated samples introduces various stability issues. Because the discriminator is trained to classify realistic and unrealistic generated samples equally as fake, its parameters are wrongly optimized as soon as the generator starts producing more realistic images. Generators in unsupervised scenarios can usually circumvent bad discriminator loss by producing different but equally realistic images. However, in a case like ours, each input corresponds to only a single correct output, thus the generator is unable to generate alternative outputs which still fool the discriminator.

Training the discriminator to classify all generated images as fake can be seen as an approximation for the actual unknown loss. We can choose a better approximation for the discriminator loss on generated samples due to the availability of comparable ground truth masks.

This approach slightly changes the GAN objective. The discriminator does not learn to predict the probability if a sample is real anymore, it instead learns to predict a quantization of a sample's quality.

By negating the prediction, it can be interpreted as a loss. The primary advantage of such a loss formulation is that high-quality generated samples should ideally receive a low loss, thus not changing the generator parameters much. High-quality samples are then mostly ignored during generator optimization while low-quality samples pose a larger impact on the parameters, resulting in a learning boost.

Furthermore, an optimized discriminator can be used to generate a loss over manually generated samples outside of the training scope. This could find practical application in various fields of biomedical image segmentation, for example to provide quality metrics of manually generated segmentations to doctors.

5.3.1. Jaccard Loss

In order to provide a loss aproximation to the discriminator, an appropriate distance metric has to be chosen. As previously discussed, the Jaccard similarity coefficient exhibits properties making it a distance metric well-suited for the comparison of segmentation masks. Jaccard similarity furhter punishes the discriminator more if at accepts maks-like but wrong shapes from the generator. For these reasons, we choose the inverse Jaccard similarity between a generated and real segmentation mask as our discriminator loss approximation.

Equation (5.1) list the original Jaccard similarity coefficient formulation where A and B each represent a set. The Jaccard similarity coefficient indicates the ratio between the intersection of two sets over their union. Two completely matching sets receive a coefficient of 1, two completely disjoint sets a coefficient of 0.

$$J(A,B) = \frac{A \cap B}{A \cup B} \tag{5.1}$$

In our case, the sets A and B are the positively classified pixels in the generated and ground truth masks respectively. We further extend the usual set semantics to accommodate for generated classifications. As the generator outputs values in the range $[0,1]$, we allow a pixel to be only partially contained in a set. The intersection between a generated image g and a ground truth image r is then defined as $\sum_{xy} \min(g_{xy}, r_{xy})$, the union as $\sum_{xy} \max(g_{xy}, r_{xy})$ where xy references a single pixel.

Due to ground truth labels always being 0 or 1, we can use a simplified formulation to calculate the Jaccard similarity between generated and ground truth masks. Equation (5.2) displays our modified Jaccard similarity $\hat{J}$

$$\hat{J}(g,r) = \frac{\sum_{xy} (g_{xy} \cdot r_{xy})}{\sum_{xy} \min(g_{xy} + r_{xy}, 1)} \tag{5.2}$$

Finally we choose new optimization objectives. The generator is trained to minimize discriminator output for generated samples. The discriminator is trained to predict 0 for ground truth masks r and $1 - \hat{J}(r,g)$ for generated masks g. We minimize the cross entropy between expected and actual values as it leads to benefits during training.

5.3.2. Experiments

We train the base architecture with the context-aware discriminator loss. This loss formulation leads to more steady generator development, thus we can train our networks for far more iterations than the base architecture. We halted training after 36000 iterations due to time constraints. At this point, the loss development was still steady, leading us to the assumption that more training iterations could have further improved the resulting segmentation performance. We choose iteration 32000 for the evaluation and comparison because it exhibits the best overall scores. The same optimizer parameters as in the base architecture are used for training.

Table 5.3 lists the evaluation metrics over the cross-validation and test sets compared to the original loss formulation. While accuracy and specificity scores remained similar compared to the base architecture, precision, recall, F_1 score and Jaccard similarity all improved around 10 percentage points.

We interpret the improved metrics as a confirmation of our hypothesis that a more appropriate loss for generated masks leads to better segementation results and more stable training.

	Context-Aware Loss		Original Loss	
Metric	Cross-Validation Set	Test Set	Cross-Validation Set	Test Set
Accuracy	0.928	0.939	0.916	0.926
Precision	**0.771**	0.770	**0.679**	0.666
Recall	**0.610**	0.638	**0.528**	0.551
F_1 Score	**0.599**	0.618	**0.484**	0.498
Specificity	0.990	0.989	0.987	0.986
Jaccard Similarity	**0.486**	0.504	**0.384**	0.399

Table 5.3.: Context-aware discriminator loss metrics

Figure 5.8.: Badly matching segmentation with context-aware discriminator loss

Still, around 9.8% of all cross-validation samples are segmented with a resulting Jaccard similarity of less than 1%. Figure 5.8 showcases an example. The ground truth mask is only minimally matched while another image region is falsely segmented and noise is present.

However, for regular samples, the network produces accurately matching segmentation masks. Figure 5.9 displays a segmentation resulting in a Jaccard similarity coefficient of over 94%. The only difference to the ground truth is a slightly under-segmented area in the lesion center and the lesion border. As the mask borders highly depend on the algorithm used to create the ground truth, we consider this example as almost perfectly segmented.

In conclusion, our conjecture regarding a context-aware discriminator loss seem to be empirically confirmed. Our network generated significantly better results by solely adjusting the loss formulation. We therefore consider this loss formulation for our final

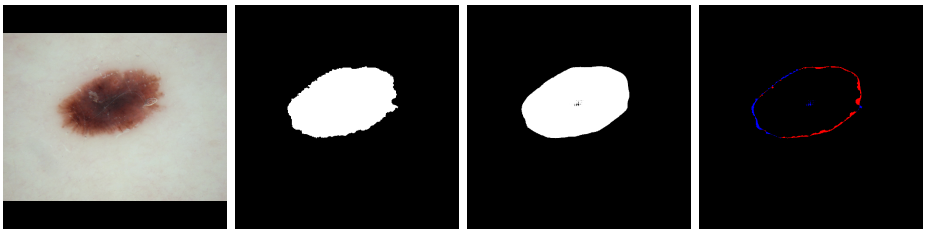


Figure 5.9.: Accurately matching segmentation with context-aware discriminator loss

Metric	Original Loss	Context-Aware Loss	Base Architecture
Accuracy	0.929	**0.930**	0.916
Precision	0.623	0.667	**0.679**
Recall	0.660	**0.673**	0.528
F_1 Score	0.564	**0.592**	0.484
Specificity	0.973	0.981	**0.987**
Jaccard Similarity	0.450	**0.458**	0.384

Table 5.4.: Comparison of architectural extensions and base architecture

architecture.

5.4. Comparison

We concluded in the enhanced generator that it needs to be blended with the enhanced discriminator in order to be reliably evaluated. Thus, we combine both enhancements into a single model and slightly downsize them due to memory and time constraints. The generator used in the combined model performs only one convolution on each encoder and decoder level. The discriminator performs only a single convolution on the lesion image branch instead of two and starts the downsampling process with 32 instead of 64 feature maps. We further reference this combined architecture as the *molanet* architecture. It has to be noted that, due to smaller filter sizes and less convolution operations, the molanet architecture contains only around $\frac{1}{5}$ of the base architecture weights.

We also concluded that context-aware discriminator loss improves training stability and result quality. However, as such effects do not apply equally to different network architectures, we compare a molanet architecture trained with traditional GAN loss to one trained with the context-aware discriminator loss.

We train the molanet architecture with original loss for 35000 iterations, with the context-aware discriminator loss for 28000 iterations. Both models trained stable and might have improved if trained for more iterations. However, due to budget limitations, longer training is not feasible in the scope of this thesis. We choose iterations 28000 and 26000 for the evaluation of the normal and context-aware loss respectively, based on their overall performance. The usual set of optimizer parameters was used during training.

Table 5.4 compares cross-evaluation metrics for the two combined models and the base architecture. Initially, it has to be noted that context-aware loss outperforms the original loss formulation in every metric despite being trained for less iterations. Furthermore, precision and specificity scores are lower than the base architecture ones. In contrast, the recall, F_1 and Jaccard similarity scores far exceed their base architecture equivalents. We interpret this as the model becoming more confident in lesion detection. Especially the recall increase of almost 15 percentage points indicates that a semantic image understanding was learned. The generator does not just try to fool the discriminator anymore but endeavors to produce well-matching segementation masks.

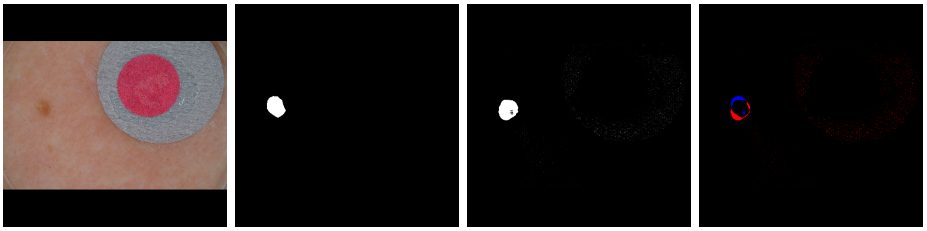


Figure 5.10.: Almost completely ignored plaster

We observe an enhanced invariance against plasters in the molanet architecture. Figure 5.10 showcases a lesion image which was already displayed for the base architecture in fig. 4.7. Back then, the actual lesion was completely ignored, while the red plaster part was masked. The figure now displays the same image, segmented by the context-aware molanet model. Few plaster pixels are still falsely masked, however, most of the generated mask only contains the actual lesion.

Plaster invariance is also reflected in the number of generated masks which exhibit a Jaccard similarity of 0%. The context-aware molanet model produces over the complete cross-validation set less than 1% such masks.

Based on this comparison, we propose a combination of all three extensions to form our final architecture.

6. Results

6.1. Final Architecture

We choose to use the molanet architecture and context-aware discriminator loss to form our final architecture. The molanet generator and discriminator are chosen because they slightly outperform their base architecture counterparts while exhibiting only around $\frac{1}{5}$ of the parameters. This implies that our network learns more semantic features, resulting in improved generalization. We choose the context-aware discriminator loss in our final architecture because it further enhances the performance of both the base architecture and molanet architecture networks.

Figure 6.1 provides a high-level overview over the complete final architecture model. Altough we train and evaluate it on 512×512 images, it can be used fully convolutional without any modifications.

The generator exhibits an encoder-decoder structure, convolving and downsampling an input image by factors of two until a bottleneck size is reached. This bottleneck is convolved twice, followed by upsampling and convolution steps. Skip connections are used to provide encoder activations to decoder inputs.

The discriminator intially performs a separate convolution on each the lesion image and a pixel-wise multiplication of lesion image and segmentation mask. The outputs are concatenated and followed by a series of convolution and downsampling steps. The discriminator is trained to predict our Jaccard loss for any combination of lesion image and segmentation.

Section A.2.3 lists all final architecture layers in detail. We further implement an end-to-end application which can be used to segment lesion images. Section B.3 describes this application.

We continue to evaluate the final architecture on our test set and compare the segmentation performance to the top ISBI challenge submissions of the years 2016 and 2017. We use the trained context-aware loss model from the extension comparison as described in section 5.4.

6.2. Segmentation Performance

We evaluate the final architecture on our test set. Table 6.1 lists the resulting metrics.

Our network reaches very high accuracy and specificity scores. As previously discussed, accuracy does not contribute much information as most pixels in segmentation masks correspond to skin and a completely black image already results in high accuracy. High specificity indicates that our network selects mole areas very carefully and does not

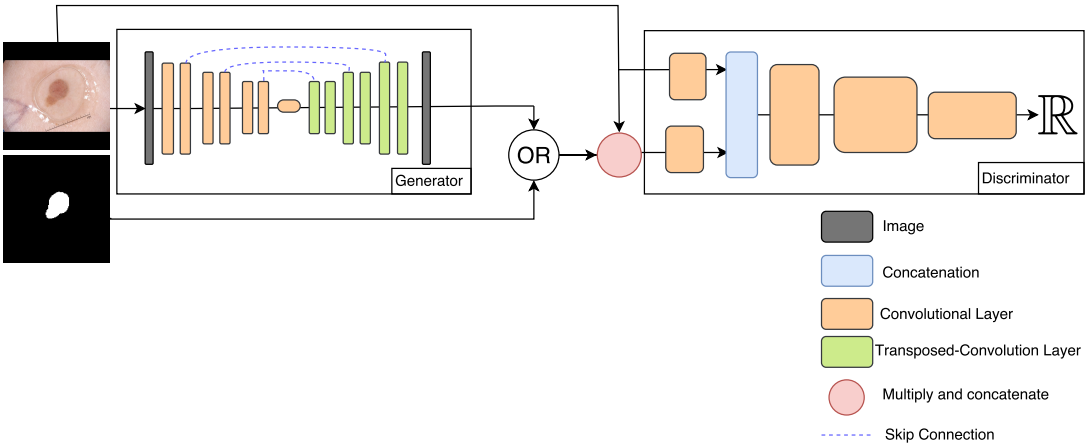


Figure 6.1.: Final architecture overview. Some intermediate layers are omitted for brevity.

Metric	Score
Accuracy	0.940
Precision	0.668
Recall	0.688
F_1 Score	0.597
Specificity	0.982
Jaccard Similarity	**0.460**

Table 6.1.: Final architecture test set metrics

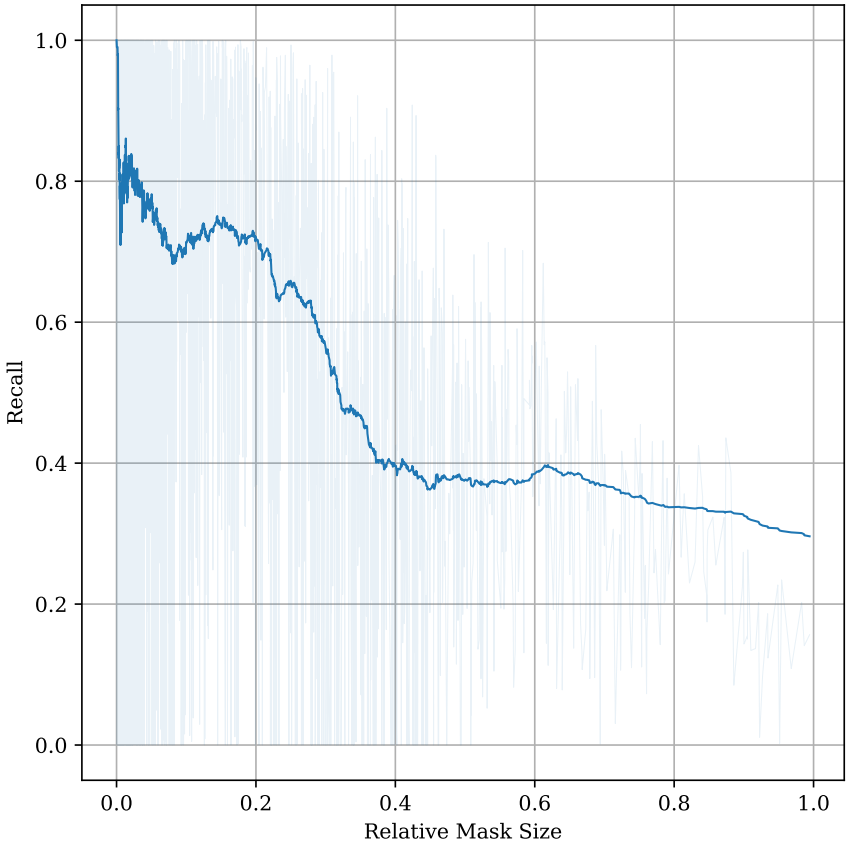


Figure 6.2.: Correlation between relative lesion size and recall

produce noisy outputs.

The precision score does not correlate with the specificity. We interpret the significantly lower precision as our network still detecting visual artifacts as lesions and wrongly segmenting them. However, it further indicates that our generator does not just produce mask-like shapes to fool the discriminator but tries to identify lesions and produce masks matching as close as possible.

Low recall scores might be attributed to overly large lesions. Figure 6.2 plots the correlation between ground truth mask size and recall. The curve is heavily smoothed, original values are displayed semi-transparently in the background. It can easily be seen how recall declines proportionally to increasing segmentation mask sizes. As large masks form minority classes in our data sets, our model did not learn handling them well enough.

Jaccard similarity as an overall segmentation mask quality indicator is unsatisfactorily low. On average, generated and ground truth segmentation masks overlap less than half of their areas. Besides just weak segmentation performance, faulty ground truth might be another cause. By manually observing our data sets, we found a multitude of ground truth masks noticeably missing their corresponding skin lesions. Figure 6.3 showcases

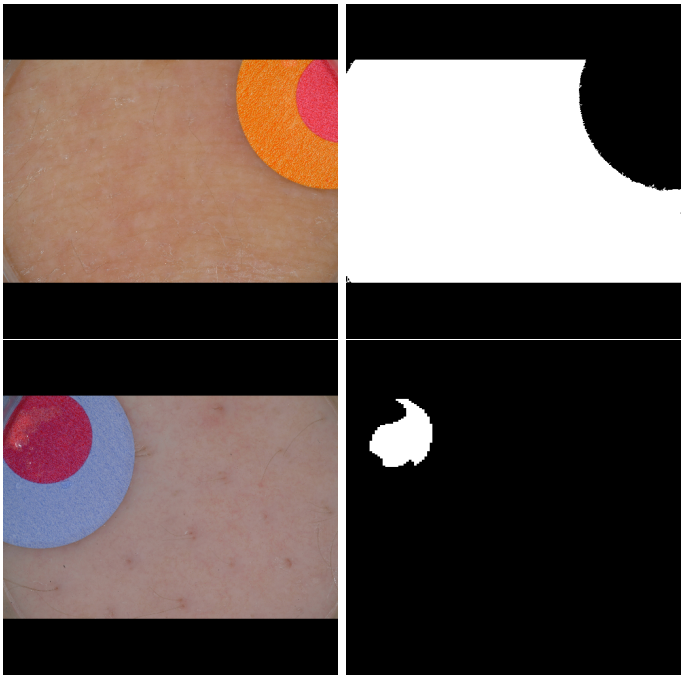
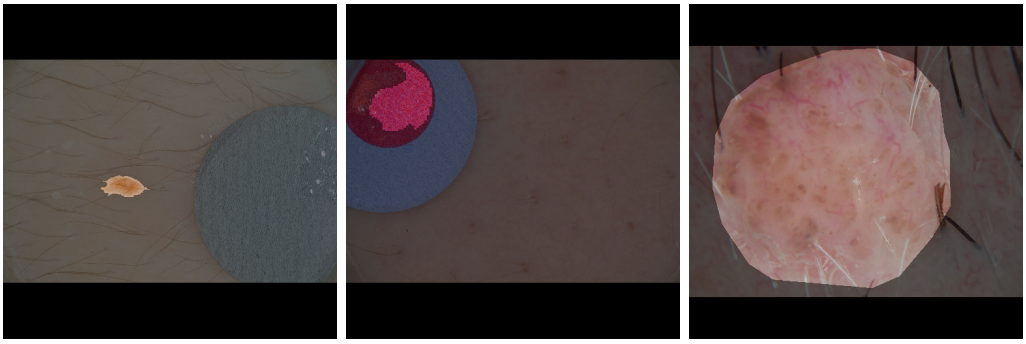


Figure 6.3.: Faulty ground truth lesion and segmentation mask pairs

two obvious examples. In the first image, almost the whole skin area is classified as lesion. This could be the result of wrongly parameterized flood-fill algorithms used to generate the ground truth mask. In the second image, instead of skin area, parts of a plaster as masked. We have no objective explanation of how this kind of ground truth originated.

As all segmentation quality metrics are averaged, incorrect ground truth might distort the resulting scores. We therefore calculate the median Jaccard similarity over all test set samples. This results in a score of 0.491 which is only slightly above the average displayed before. Our metrics can therefore be considered representative. However, faulty ground truth can still aggravate network training and distort our optimization objective, thus resulting in reduced segmentation performance.

A different matter worth evaluating is the performance of our discriminator. Our context-aware loss formulation should train the discriminator to assess the quality of any segmentation mask. Thus, we evaluate the discriminator performance on a set of ground truth segmentations. Figure 6.4 lists a selection of ground truth segmentation masks and their quality as predicted by our discriminator. Figure 6.4a shows a segmentation mask matching its ground truth very well. Our discriminator recognizes this and outputs a very low loss. Similarly, fig. 6.4b contains a ground truth segmentation mask which we consider faulty. This is also recognized by our discriminator, resulting in a rather high loss. However, as the mask is completely nonsensical, the loss for this sample should subjectively be much higher. This hints at further improvement potential in our discriminator. Figure 6.4c finally displays a very large lesion. Our discriminator predicts



(a) 0.164 (b) 0.548 (c) 0.000

Figure 6.4.: Discriminator predictions of ground truth masks

this sample as having almost zero loss. We assume this correlates with the diminished generator performance on large lesions. Our discriminator is trained with large lesions having loss zero for correct masks from the ground truth and very high loss from badly generated masks. It thus might just recognize large segmentation masks as stemming from the ground truth and classifies them as having no loss.

Finally, fig. 6.5 showcases a set of high-quality segmentations produced by our final architecture, fig. 6.6 a set of failed ones.

6.3. ISBI Challenge Comparison

The ISBI challenges are a set of four contests in the field of biomedical imaging, hosted within the scope of the annual *IEEE International Symposium on Biomedical Imaging* conference. One of those challenges is *Skin Lesion Analysis Towards Melanoma Detection*. This challenge is further divided into three parts: Lesion segmentation, lesion dermoscopic feature extraction and lesion classification. We utilize submission results of the first part to compare our results to competing models.

The challenge provides training and validation sets of skin lesions and their corresponding segmentation masks. All samples stem directly from the ISIC archive and as such already available to us. Participators create their segmentation models using the provided training set and validate them against the provided validation set. In the end, a withheld test set is used to calculate the final scores and declare a winner. Test set samples become available as soon as the challenge is finished.

We compare our previously trained final architecture to the top three results of the 2016 and 2017 challenges, using the respective challenge test sets. Our results have to be taken with a grain of salt because our training data set might partially overlap with the ISBI test set and we calculate our scores on resized 512×512 versions of the test set samples and not their original size. Furthermore, the ISBI challenges evaluate slightly different metrics than those presented in this thesis. We only display metrics for which scores on both our model and challenge submissions are available.

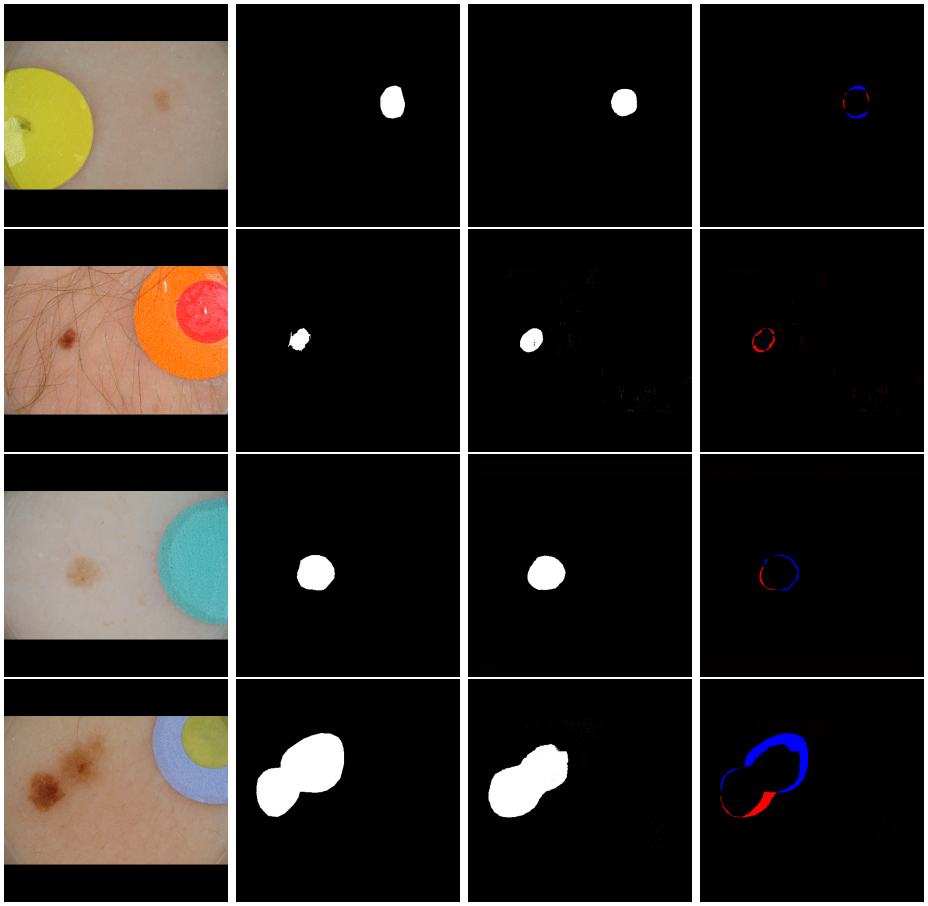


Figure 6.5.: High-quality final architecture segmentations

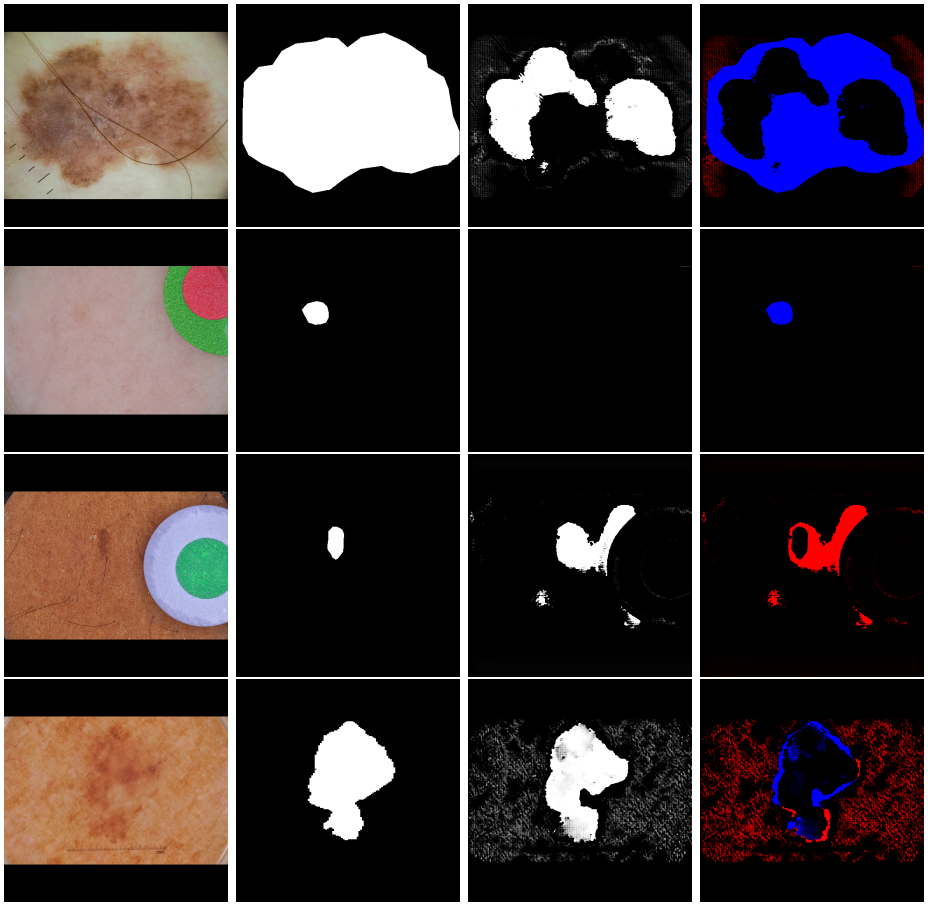


Figure 6.6.: Failed final architecture segmentations

Metric	1st place	2nd place	3rd place	Our Model
Accuracy	**0.953**	0.949	0.952	0.893
Recall	0.910	**0.911**	0.880	0.674
Specificity	0.965	0.957	0.969	**0.990**
Jaccard Similarity	**0.843**	0.829	0.822	0.601

Table 6.2.: Comparison between top three ISBI challenge 2016 entries and our results

Metric	1st place	2nd place	3rd place	Our Model
Accuracy	**0.934**	0.932	**0.934**	0.909
Recall	**0.825**	0.820	0.802	0.675
Specificity	0.975	0.978	0.985	**0.986**
Jaccard Similarity	**0.765**	0.762	0.760	0.516

Table 6.3.: Comparison between top three ISBI challenge 2017 entries and our results

Table 6.2 compares the top three ISBI 2016 submissions to our model. Table 6.3 provides an equal comparison for the top three ISBI 2017 submissions. For each metric, the highest score is marked bold. All scores were taken directly from the online challenge leaderboard. In both years, submissions are ranked according to their average Jaccard similarity.

The evaluation metrics over the 2017 test set are lower compared to 2016 as noticed in the top three results. This effect is also noticeable in our model. In both challenges, our result leads to around 0.25 points less Jaccard similarity. This concludes that, in order to be competitive, our model has to be further improved. However, we also observed our model outperforming low-ranked submissions in both challenges. Despite not performing like state-of-the-art models, our results still exhibit practical relevancy. Furthermore, we reach the highest specificity compared to both year's top-ranked submissions.

7. Conclusion

We have empirically shown that GANs can be used in a fully supervised setting by creating a neural network architecture termed molanet. The molanet architecture was successfully applied to the problem of skin lesion segmentation.

We have further proposed an improved, context-aware discriminator loss that can be used in a fully supervised setting. We experimentally confirmed that the improved loss formulation leads to better results and much more stable training. We encourage the use of context aware discriminator loss in any supervised GAN application.

Furthermore, we have shown that the discriminator with context-aware loss learns a meaningful representation of what constitutes a good segmentation mask that corresponds to human understanding.

In conclusion, while our results could not outperform state-of-the-art models, they still show that GANs relying solely on adversarial loss can be used in the context of biomedical image segmentation and other supervised scenarios.

It would be interesting to research the theoretical implications of the context-aware discriminator loss on the GAN objective. Specifically, it should be investigated whether the improved loss already imposes a Lipschitz constraint on the optimization process, which would render gradient-penalty superfluous. We suspect this to be the case since a similar formulation was used in [43] to impose a Lipschitz constraint.

Another area of exploration for future work could be the exploitation of our fully convolutional architecture to generate masks of arbitrary size. We could not investigate this due to time constraints.

Another interesting venture would be to evaluate the molanet architecture with context-aware objective in conjuction with an additional supervised loss term.

Lastly, it might be worth to pursue longer training of our final architecture as its scores were steadily improving until we halted training. We did not explore this due to time and budget constraints.

Because of the demanding memory and processing power requirements we used *Amazon Elastic Compute Cloud (EC2)* to parallelize training. We observed the EC2 GPU accelerated instances were slower than a physical computer with a current generation GPU. Nevertheless, cloud instances are a viable solution if computation power and parallelization is needed.

We deem generative adversarial networks a promising novel approach to biomedical image segmentation. Especially in settings where noisy ground truth is common, GANs could proliferate. Thus, we expect to see more research in the area of supervised adversarial models. We hope that our contribution to the field of skin lesion segmentation in particular inspires a deeper exploration of GANs and ultimately leads to better automated systems for the timely diagnosis and treatment of skin cancers.

Bibliography

[1] T. Kubota. (2017). Deep learning algorithm does as well as dermatologists in identifying skin cancer, [Online]. Available: http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/ (visited on 08/17/2017).

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, «Generative adversarial networks,» *CoRR*, vol. abs/1406.26618, 2014. [Online]. Available: https://arxiv.org/abs/1406.2661.

[3] A. Jung. (2017). Generate cat images with neural networks, [Online]. Available: https://github.com/aleju/cat-generator/ (visited on 08/17/2017).

[4] A. Radford, L. Metz, and S. Chintala, «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,» *ArXiv e-prints*, Nov. 2015. arXiv: 1511.06434 [cs.LG].

[5] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,» *ArXiv e-prints*, Sep. 2016. arXiv: 1609.04802 [cs.CV].

[6] L. Chen, S. Srivastava, Z. Duan, and C. Xu, «Deep Cross-Modal Audio-Visual Generation,» *ArXiv e-prints*, Apr. 2017. arXiv: 1704.08292 [cs.CV].

[7] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, «Semantic Segmentation using Adversarial Networks,» *ArXiv e-prints*, Nov. 2016. arXiv: 1611.08408 [cs.CV].

[8] Schweizerische Gesellschaft für Dermatologie und Venerologie. (2017). Melanoma.ch - starke zunahme von hautkrebs, [Online]. Available: http://derma.ch/spec/melanoma/Patienten/2.html (visited on 08/15/2017).

[9] International Society for Digital Imaging of the Skin. (2017). Isic project | isdis, [Online]. Available: http://isdis.net/isic-project/ (visited on 08/15/2017).

[10] International Skin Imaging Collaboration: Melanoma Project. (2017). Isic archive. English.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, «Improved Techniques for Training GANs,» *ArXiv e-prints*, Jun. 2016. arXiv: 1606.03498 [cs.LG].

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] J. Li, A. Madry, J. Peebles, and L. Schmidt, «Towards Understanding the Dynamics of Generative Adversarial Networks,» *ArXiv e-prints*, Jun. 2017. arXiv: 1706.09884 [cs.LG].

[14] M. Mirza and S. Osindero, «Conditional Generative Adversarial Nets,» *ArXiv e-prints*, Nov. 2014. arXiv: 1411.1784 [cs.LG].

[15] M. Arjovsky and L. Bottou, «Towards Principled Methods for Training Generative Adversarial Networks,» *ArXiv e-prints*, Jan. 2017. arXiv: 1701.04862 [stat.ML].

[16] M. Arjovsky, S. Chintala, and L. Bottou, «Wasserstein GAN,» *ArXiv e-prints*, Jan. 2017. arXiv: 1701.07875 [stat.ML].

[17] International Skin Imaging Collaboration: Melanoma Project. (2017). Isic archive, [Online]. Available: https://isic-archive.com/ (visited on 08/01/2017).

[18] Edinburgh Research and Innovation. (2017). Dermofit image library 3.00. English.

[19] Z. Zhang, W. V. Stoecker, and R. H. Moss, «Border detection on digitized skin tumor images,» *IEEE Transactions on Medical Imaging*, vol. 19, no. 11, pp. 1128–1143, Nov. 2000, ISSN: 0278-0062. DOI: 10.1109/42.896789.

[20] D. Chai and K. N. Ngan, «Face segmentation using skin-color map in videophone applications,» *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 4, pp. 551–564, Jun. 1999, ISSN: 1051-8215. DOI: 10.1109/76.767122.

[21] J. Canny, «A computational approach to edge detection,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[22] opencv dev team. (2017). Hough line transform - opencv 2.4.13.3 documentation, [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html (visited on 08/01/2017).

[23] P. Refaeilzadeh, L. Tang, and H. Liu, «Cross-validation,» *Encyclopedia of Database Systems*, pp. 532–538, 2009.

[24] S.-H. Lim, S. R. Young, and R. M. Patton, «An analysis of image storage systems for scalable training of deep neural networks,» in. Jan. 2016.

[25] Google. (2017). Performance guide | tensorflow, [Online]. Available: https://www.tensorflow.org/performance/performance_guide#use_large_files (visited on 07/29/2017).

[26] S. Chacon and B. Straub, *Pro Git*, 2nd. Apress, 2014. [Online]. Available: https://git-scm.com/book/en/v2/ (visited on 07/29/2017).

[27] Google. (2017). Threading and queues | tensorflow, [Online]. Available: https://www.tensorflow.org/programmers_guide/threading_and_queues (visited on 07/29/2017).

[28] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow, Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[29] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, «Image-to-Image Translation with Conditional Adversarial Networks,» *ArXiv e-prints*, Nov. 2016. arXiv: 1611.07004 [cs.CV].

[30] O. Ronneberger, P. Fischer, and T. Brox, «U-Net: Convolutional Networks for Biomedical Image Segmentation,» *ArXiv e-prints*, May 2015. arXiv: 1505.04597 [cs.CV].

[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng, «Rectifier nonlinearities improve neural network acoustic models,» 2013.

[32] V. Nair and G. E. Hinton, «Rectified linear units improve restricted boltzmann machines,» in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Fürnkranz and T. Joachims, Eds., Omnipress, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf.

[33] B. Xu, R. Huang, and M. Li, «Revise Saturated Activation Functions,» *ArXiv e-prints*, Feb. 2016. arXiv: 1602.05980 [cs.LG].

[34] S. Ioffe and C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» *ArXiv e-prints*, Feb. 2015. arXiv: 1502.03167 [cs.LG].

[35] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, «The Importance of Skip Connections in Biomedical Image Segmentation,» *ArXiv e-prints*, Aug. 2016. arXiv: 1608.04117 [cs.CV].

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, «Dropout: A simple way to prevent neural networks from overfitting,» *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[37] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, «Improved Training of Wasserstein GANs,» *ArXiv e-prints*, Mar. 2017. arXiv: 1704.00028 [cs.LG].

[38] S. Ruder, «An overview of gradient descent optimization algorithms,» *ArXiv e-prints*, Sep. 2016. arXiv: 1609.04747 [cs.LG].

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, «Imagenet classification with deep convolutional neural networks,» in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[40] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, «Flexible, high performance convolutional neural networks for image classification,» in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11, Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 1237–1242, ISBN: 978-1-57735-514-4. DOI: 10.5591/978-1-57735-

516-8/IJCAI11-210. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210.

[41] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, «Striving for simplicity: The all convolutional net,» *CoRR*, vol. abs/1412.6806, 2014. [Online]. Available: http://arxiv.org/abs/1412.6806.

[42] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollàr, «Learning to Refine Object Segments,» *ArXiv e-prints*, Mar. 2016. arXiv: 1603.08695 [cs.CV].

[43] G.-J. Qi, «Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities,» *ArXiv e-prints*, Jan. 2017. arXiv: 1701.06264 [cs.CV].

[44] D. Sussillo and L. F. Abbott, «Random Walk Initialization for Training Very Deep Feedforward Networks,» *ArXiv e-prints*, Dec. 2014. arXiv: 1412.6558.

[45] T. Schlusser and K. Reitz, *The Hitchhiker's Guide to Python, Best Practices for Development*. O'Reilly Media, 2016. [Online]. Available: http://docs.python-guide.org/en/latest/ (visited on 08/14/2017).

Appendices

A. Architecture Details

A.1. Base Architecture

Table A.1 and Table A.2 list all layers and their properties for the base architecture generator and discriminator respectively. All normal convolutions perform a leaky ReLu activation with slope $\alpha = 0.2$, transposed convolutions a normal ReLu activation. As an exception, the generator output convolution uses the tanh activation function. Every input is zero padded in order to keep the spatial dimensions stable. The first generator and last generator convolutions, as well as the first discriminator convolution use 5×5 filters. All other convolutions utilize 4×4 filters as a memory tradeoff. Batch normalization is applied prior to the activation function on all generator layers except the last one. The discriminator does not utilize batch normalization. Dropout with a probability of 50% to keep a neuron connection is performed after the first two decoder levels in the generator.

All bias weights are initialized as 0, all transposed convolution weights uniform without scaling variance [44]. Normal convolutions are initialized from a normal distribution with mean 0.0 and standard deviation 0.02. Values further than two standard deviations from the mean are discarded and re-drawn.

A.2. Architectural Extensions

A.2.1. Enhanced U-Net Generator

Table A.3 lists all layers and their properties for the enhanced generator architecture. All normal convolutions perform a leaky ReLu activation with slope $\alpha = 0.2$, transposed convolutions a normal ReLu activation. As an exception, the output convolution uses the tanh activation function. Every input is padded by mirroring it for undefined areas. The two-dimensional filter sizes are 3×3 except for the output layer which performs a 1×1 convolution. Batch normalization is applied prior to the activation function on all layers except the last one.

All bias weights are initialized as 0, all convolution weights uniform without scaling variance [44]. The constant factors are chosen according to the layer's activation function.

A.2.2. Enhanced Discriminator

Table A.4 lists all layers and their properties for the enhanced discriminator architecture. All normal convolutions, except the very last 1×1, perform a leaky ReLu activation with slope $\alpha = 0.2$. Every input is padded by mirroring it for undefined areas. After

Name	Type	Stride	Concatenation	Output Features
e1	conv	2	-	64
e2	conv	2	-	128
e3	conv	2	-	256
e4	conv	2	-	512
e5	conv	2	-	1024
e6	conv	2	-	1024
e7	conv	2	-	1024
e8	conv	2	-	1024
e9	conv	2	-	1024
d9	transposed conv	2	e8	1024
d8	transposed conv	2	e7	1024
d7	transposed conv	2	e6	1024
d6	transposed conv	2	e5	1024
d5	transposed conv	2	e4	512
d4	transposed conv	2	e3	256
d3	transposed conv	2	e2	128
d2	transposed conv	2	e1	64
d1	transposed conv	2	-	1

Table A.1.: Base architecture generator layers

Name	Type	Stride	Output Features
c1	conv	2	64
c2	conv	2	128
c3	conv	2	256
c4	conv	2	512
c5	conv	2	1024
c6	conv	2	1024
c7	conv	2	1024
c8	conv	2	1024
out	conv	2	1

Table A.2.: Base architecture discriminator layers

Name	Type	Stride	Concatenation	Output Features
e1_1	conv	1	-	64
e1_2	conv	1	-	64
e1_down	conv	2	-	64
e2_1	conv	1	-	128
e2_2	conv	1	-	128
e2_down	conv	2	-	128
e3_1	conv	1	-	256
e3_2	conv	1	-	256
e3_down	conv	2	-	256
e4_1	conv	1	-	512
e4_2	conv	1	-	512
e4_down	conv	2	-	512
bottleneck_in	conv	1	-	1024
bottleneck_out	conv	1	-	1024
d4_up	transposed conv	2	e4_2	512
d4_1	conv	1	-	512
d4_2	conv	1	-	512
d3_up	transposed conv	2	e3_2	256
d3_1	conv	1	-	256
d3_2	conv	1	-	256
d2_up	transposed conv	2	e2_2	128
d2_1	conv	1	-	128
d2_2	conv	1	-	128
d1_up	transposed conv	2	e1_2	64
d1_1	conv	1	-	64
d1_2	conv	1	-	64
out	conv	1	-	1

Table A.3.: Enhanced U-Net generator layers

Name	Filter Size	Stride	Output Features
x1	5×5	1	16
x2	5×5	1	32
y1	5×5	1	32
concatenate x2 and y1			
c1	3×3	1	64
c1_down	3×3	2	64
c2	3×3	1	128
c2_down	3×3	2	128
c3	3×3	1	256
c3_down	3×3	2	256
c4	3×3	1	512
c4_down	3×3	2	512
c5	3×3	1	512
c5_down	3×3	2	512
c6	3×3	1	512
c6_down	3×3	2	512
c7	3×3	1	512
c7_down	3×3	2	512
c8	3×3	1	512
c8_down	3×3	2	512
c9	3×3	1	1024
c9_down	3×3	2	1024
out	1×1	1	1

Table A.4.: Enhanced discriminator layers

the concatenation of the x and y inputs, which are the image and mask respectively, the layer has a size of 64. The 64 feature are then compacted into 32 feature layers in the next convolution.

A.2.3. Final Architecture

Table A.5 lists all generator layers of the final architecture, table A.6 all discriminator layers. All normal convolutions, except the very last 1×1 ones, perform a leaky ReLu activation with slope $\alpha = 0.2$. Every input is padded by mirroring it for undefined areas. Batch normalization is used in every generator layer and nowhere in the discriminator. All transposed convolutions perform a normal ReLu activation.

Name	Type	Stride	Concatenation	Output Features
e1	conv	1	-	64
e1_down	conv	2	-	64
e2	conv	1	-	128
e2_down	conv	2	-	128
e3	conv	1	-	256
e3_down	conv	2	-	256
e4	conv	1	-	512
e4_down	conv	2	-	512
bottleneck_in	conv	1	-	1024
bottleneck_out	conv	1	-	1024
d4_up	transposed conv	2	e4_2	512
d4	conv	1	-	512
d3_up	transposed conv	2	e3_2	256
d3	conv	1	-	256
d2_up	transposed conv	2	e2_2	128
d2	conv	1	-	128
d1_up	transposed conv	2	e1_2	64
d1	conv	1	-	64
out	conv	1	-	1

Table A.5.: Final architecture generator layers

Name	Filter Size	Stride	Output Features
x1	5×5	1	32
y1	5×5	1	32
concatenate x1 and y1			
c1	3×3	1	32
c1_down	3×3	2	32
c2	3×3	1	64
c2_down	3×3	2	64
c3	3×3	1	128
c3_down	3×3	2	128
c4	3×3	1	256
c4_down	3×3	2	256
c5	3×3	1	512
c5_down	3×3	2	512
c6	3×3	1	512
c6_down	3×3	2	512
c7	3×3	1	512
c7_down	3×3	2	512
c8	3×3	1	512
c8_down	3×3	2	512
c9	3×3	1	1024
c9_down	3×3	2	1024
out	1×1	1	1

Table A.6.: Final architecture discriminator layers

B. Implementation Details

B.1. Frameworks and Technologies

We developed our implemetation in python 3.6. We heavily utilized the TensorFlow 1.2 library for numerical computations on the CPU and GPU. We further utilize Numpy 1.13 for numerical computations outside of the scope of neural networks, OpenCV 3.2 and Pillow 4.1 for various imaging-related tasks and Matplotlib 2.0 to create our plots. Jupyter was utilized in the data analysis and PostgreSQL for storage. A complete list of all requirements can be found in the `requirements.txt` file provided with our code.

B.2. Code Architecture

We developed a lightweight framework which allows simple and quick evaluation of different models. We structure our project into different modules according to python best practices [45]. Table B.1 provides an overview over the module structure.

We further developed various executable modules. Most of them are *run scripts*. A run script captures the complete architecture and hyperparameter specification of a single experiment. Each experiment in this thesis can be recreated by executing the related run script. Table B.2 displays all runscripts and their corresponding experiment. All run scripts are children of the `molanet.run` module. Table B.3 lists the important other executable modules and their purpose. All scripts are commandline-based.

Figure B.1 provides an overview of the primary framework classes and interfaces. In order to provide a high level overview, method parameters and module-level methods are omitted. All exported classes and methods are fully documented in the code.

The `InputPipeline` class provides basic pipeline functionality. We provide two different implementations, one for training and one for evaluation. Every `InputPipeline` optionally receives a `ColorConverter` which converts from the RGB color space to some other one and back.

`NetworkFactory` implementations create specific generator and discriminator architectures. As the same generator and discriminators have to be created multiple times during a single training run, this functionality is abstracted into factories. The `create_generator` and `create_discriminator` each receive all contextual information required to create a specific model through their parameters. For example, to prepare training, the `create_discriminator` method is called twice, once given a ground truth segemtation mask placeholder as y input, once the generator output.

`ObjectiveFactory` implemenations serve a similar purpose. They each create a specific generator or discriminator loss through the `create_generator_loss` and `create_discriminator_loss`

Module	Description
`molanet`	Base module which acts as a root for all other modules.
`molanet.base`	Contains the architecture-related interfaces and two classes which perform training and evaluation respectively.
`molanet.input`	Contains the input-related interfaces, concrete input pipeline implementations and data augmentation methods.
`molanet.operations`	Cotains high level operations utilized to build neural network models.
`molanet.data`	Base module which acts as a root for all data-related modules.
`molanet.data.database`	Contains all database-related functionality.
`molanet.data.entities`	Contains the data model.
`molanet.data.feature_extraction`	Base module for various modules used to extract features from our data set and performing the data set split.
`molanet.models`	Base module for all concrete architectures implementing the interfaces in `molanet.base`.
`molanet.run`	Base module for *run scripts*, each representing a single experiment performed during this thesis.

Table B.1.: Modules and their purpose

Module	Experiment
`base_architecture`	Base architecture
`enhanced_generator`	Enhanced U-Net generator
`enhanced_discriminator`	Enhanced segmentation discriminator
`better_loss`	Context-aware discriminator loss
`combined_generator_discriminator`	Combination of enhanced generator and enhanced discriminator
`all_enhancements`	Combination of all three architectural extensions

Table B.2.: Run scripts and corresponding experiments

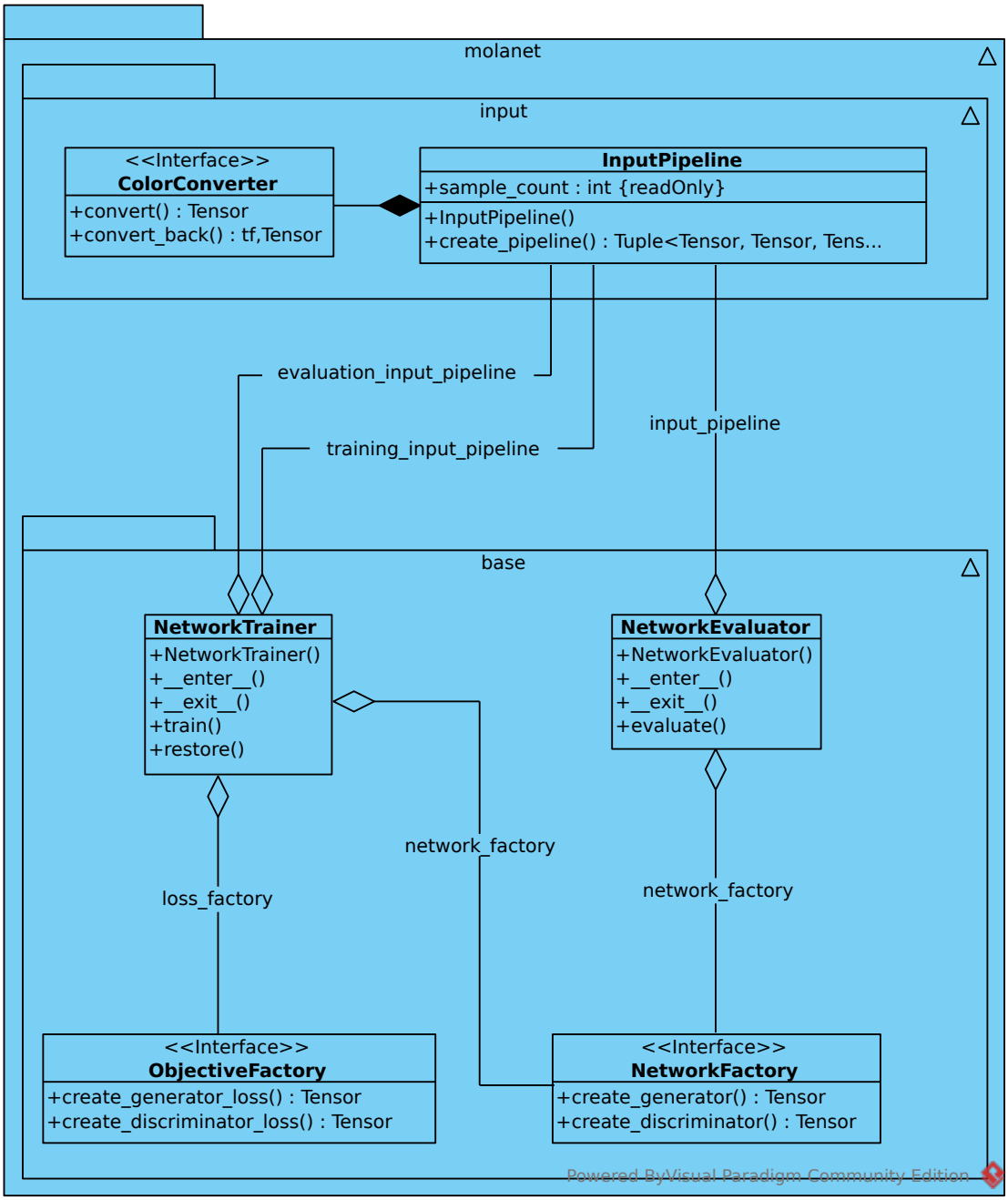


Figure B.1.: UML class diagram providing an architectural overview

Module	Description
`molanet.evaluation`	Evaluates a trained model on a data set and outputs individual and global sample metrics as well as segmentations.
`molanet.application`	End-to-end application which is used for practical application.
`molanet.data.save_tfrecords`	Exports one or more data sets from the database, either meta data only or including serialized record protocol buffers.

Table B.3.: Additional executable modules and their purpose

respectively. As in `NetworkFactory`, all contextual parameters are supplied as method parameters.

Finally, `NetworkTrainer` and `NetworkEvaluator` contain the actual training and evaluation functionality. A `NetworkTrainer` receives a `NetworkFactory` and `ObjectiveFactory` which create the model to train. It further requires two `InputPipeline`s, one serving training samples, the other serving cross-validation samples. The `NetworkTrainer` can then train a model and optionally restore weights from a training checkpoint. Similarly, a `NetworkEvaluator` receives a `NetworkFactory` which re-creates the network architecture to train, together with an `InputPipeline` which provides the evaluation samples. Both `NetworkTrainer` and `NetworkEvaluator` manage Tensorflow sessions, logging and output saving.

B.3. Application

We provide an application script which allows to use our model for segmentation and mask qualification. Running this script requires our `molanet` module to be in the pythonpath, for example by setting an environment variable or running the script from the root directory of our code.

The application script can be run like any other python module. If at the project-root, it is executed by issuing `python -m molanet.application PARAMS` from the command line wher `PARAMS` are parameters as specified in table B.4. An additional help text describing those parameters is displayed if the script is executed with a `-h` parameter.

Our script provides two modes:

segment creates a segmentation mask for each input.

evaluate evaluates each input together with its corresponding mask and prints the quality score as produced by the discriminator.

`segment` mode interprets all input files as lesion images. The generated mask file name is equal to the input file name plus an additional `_mask` suffix prior to the file extension. If the resulting file already exists, segmentation of that mask is skipped. In `evaluate` mode,

Parameter	Type	Description
`INPUT`	list of paths	One or more path to files and directories serving as inputs.
`--output`	directory	Output directory into which segmentation results should be written. If not specified, generated segmentation images will be stored in the same location as their inputs.
`--checkpoint`	file	Path to the file containing model parameters to use. If not specified, this defaults to `model.ckpt`.
`--gpu`	-	If specified, the GPU is used to run the networks. If not specified, the CPU is used.
`--verbose`	-	If specified, the script prints verbose output which can be used for debugging purposes.

Table B.4.: Application script parameters

all input files are assumed to be lesion images. The corresponding mask is expected to be a file with the same file name but a `_mask` suffix prior to the file extension.

All inputs can be either file names or directories. If a directory is specified, all contained files are assumed to be lesion images.

Our application script assumes the file containing model parameters to reside in the directory the script is invoked from and to be named named `model.ckpt`.

B.4. Project Setup

As already mentioned, our project is structured according to python best practices [45]. We provide a `setup.py` script which is used to install our code as a module. There is also a `requirements.txt` file listing all dependencies and their versions. This file can be fed directly into the python package manager by issuing `pip install -r requirements.txt`. We strongly recommend to use a python virtual environment in order to avoid dependency conflicts. Furthermore, we recommend to build TensorFlow from source to achieve the highest performance possible. Our project structure is compatible with most modern python IDEs and should be easily importable.

B.5. Data Organization

Tables in this section describe our data scheme on a contextual and physical level. Table B.5, table B.6 and table B.7 each describe the attributes of an entity. Table B.8 lists the data type mapping from entity attribute types to PostgreSQL types, table B.9 the primary key constraints of our pyhsical data model.

Attribute	Data Type	Description
`uuid`	UUID	128 bit UUID generated on first import. This is the unique sample identifier in our system, regardless of the original data source. The same UUID always references the same sample, even if that sample is re-imported.
`data_source`	string	Data source from which the sample originates. This is `isic` for ISIC Archive samples, `dermofit` for Dermofit Library samples.
`data_set`	string	Data set name or id from which the sample originates.
`source_id`	string	Original id of the sample in the source data set. This allows referencing samples on re-import and keeping the generated `uuid` attribute pointing to the same sample.
`name`	string	Name of the sample in the original data source. This attribute is just for the ease of debugging and does not serve a linking purpose.
`height`	integer	Height of the sample image, in pixels.
`width`	integer	Width of the sample image, in pixels.
`diagnosis`	string	Rough clinical diagnosis of this sample. Either `UNKNOWN` if the diagnosis is not known or the sample diagnosis, normalized over all source data sets. This attribute merely exists for reasoning purposes.
`image`	binary array	Serialized numpy image matrix of shape $height \times width \times 3$. It is always assumed that the image contains 3 channels. The matrix is stored row-major with interleaved R, G and B channels. The values are 8 bit unsigned integers in the range $[0, 255]$.

Table B.5.: mole_samples Attributes

Field	Data Type	Description
`mole_sample_uuid`	UUID	Reference to the sample UUID this segmentation belongs to.
`source_id`	string	Id of this segmentation in its original data source. This attribute is necessary to update a segmentation and to distinguish in case a single sample image has multiple segmentations.
`height`	integer	Height of the segmentation, in pixels.
`width`	integer	Width of the segmentation, in pixels.
`mask`	binary array	Serialized numpy segmentation matrix of shape $height \times width$. The matrix is stored row-major. The values are 8 bit unsigned integers in the range $[0, 1]$ where 0 indicates a skin pixel, 1 a lesion pixel.

Table B.6.: segmentations Attributes

Field	Data Type	Description
`mole_sample_uuid`	UUID	Reference to the sample UUID this segmentation belongs to.
`segmentation_source_id`	string	Id of the referenced segmentation in its original data source.
`set_name`	string	Name of the data set. This attribute groups samples into individual data sets.

Table B.7.: set_entries Attributes

Conceptual Type	PostgreSQL Type
`UUID`	`text`
`string`	`text`
`integer`	`numeric`
`binary array`	`bytea`

Table B.8.: Conceptual to PostgreSQL Data Type Mapping

Table	Primary Key Attributes
`mole_samples`	`uuid`
`segmentations`	`mole_sample_uuid, source_id`
`set_entries`	`mole_sample_uuid, segmentation_source_id, set_name`

Table B.9.: Primary Keys

C. Code Listings

C.1. Data Model SQL

Listing C.1: Data Model

```
-----------------------------------------------------------
--                    MOLANET DATA MODEL                 --
--                                                       --
-- This assumes a user called molanet_load exists        --
-- which will be the owner of the created tables.        --
-- All tables are created in the public schema.          --
-----------------------------------------------------------


-- Table: public.mole_samples

-- DROP TABLE public.mole_samples;

CREATE TABLE public.mole_samples
(
  uuid text NOT NULL,
  data_source text NOT NULL,
  data_set text NOT NULL,
  source_id text NOT NULL,
  name text NOT NULL,
  height numeric NOT NULL,
  width numeric NOT NULL,
  diagnosis text NOT NULL,
  image bytea NOT NULL,
  CONSTRAINT samples_pkey PRIMARY KEY (uuid),
  CONSTRAINT samples_uuid_unique UNIQUE (uuid)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.mole_samples
  OWNER TO molanet_load;
```

```
-- Table: public.segmentations

-- DROP TABLE public.segmentations;

CREATE TABLE public.segmentations
(
  mole_sample_uuid text NOT NULL,
  source_id text NOT NULL,
  height numeric NOT NULL,
  width numeric NOT NULL,
  mask bytea NOT NULL,
  CONSTRAINT segmentations_pkey PRIMARY KEY (mole_sample_uuid,
      source_id),
  CONSTRAINT segmentations_mole_sample_uuid_fkey FOREIGN KEY (
      mole_sample_uuid)
       REFERENCES public.mole_samples (uuid) MATCH SIMPLE
       ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT segmentations_mole_sample_uuid_source_id_unique
      UNIQUE (mole_sample_uuid, source_id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.segmentations
  OWNER TO molanet_load;


-- Table: public.set_entries

-- DROP TABLE public.set_entries;

CREATE TABLE public.set_entries
(
  mole_sample_uuid text NOT NULL,
  segmentation_source_id text NOT NULL,
  set_name text NOT NULL,
  CONSTRAINT set_entries_pkey PRIMARY KEY (mole_sample_uuid,
      segmentation_source_id, set_name),
  CONSTRAINT set_entries_segmentation_fkey FOREIGN KEY (
      mole_sample_uuid, segmentation_source_id)
       REFERENCES public.segmentations (mole_sample_uuid,
           source_id) MATCH SIMPLE
       ON UPDATE CASCADE ON DELETE CASCADE,
```

```
  CONSTRAINT set_entries_unique UNIQUE (mole_sample_uuid,
      segmentation_source_id, set_name)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.set_entries
  OWNER TO molanet_load;
```

Statement of Authorship

We hereby declare that we completed this thesis on our own, without the help of third parties and that information which has been directly or indirectly taken from other sources has been noted as such.

Windisch, August 18, 2017

....