INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Neural Inpainting: Repairing Artifacts in Histological Brain Sections with Deep Generative Models

Tim Kaiser

Masterarbeit

Date of issue: 07. Juli 2021
Date of submission: 15. Dezember 2021
Reviewers: Dr. Timo Dickscheid
Prof. Dr. Markus Kollmann

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, 15. Dezember 2021

(Tim Kaiser)

Diese Arbeit wurde in Zusammenarbeit mit der Forschungszentrum Jülich GmbH am Institut für Neurowissenschaften und Medizin - Strukturelle und funktionelle Organisation des Gehirns (INM-1) erstellt.

Abstract

The human brain is subject to many different branches of modern neuroscience with an increasing involvement of computational tools, such as High-Performance Computing (HPC) and Artificial Intelligence (AI). Tasks like brain mapping and brain simulations are reliant on reference brain models with high amounts of detail. While most reference models operate on a macroscopic scale, the Julich BigBrain Project [1] developed a freely available, ultrahigh resolution 3D human brain model from 7404 histological cuts, with 20 microns thickness, of a single human brain. These images, scanned at a spatial resolution of 1 micron per pixel, often contain damaged sections of varying size and origin. Physical damage may be present in the tissue sections prior to the cutting process or be a result of it. This, together with other potential causes in the data generation process, such as staining of the tissue for cell bodies, results in many kinds of histological artifacts in the tissue sections. Previously, where possible these were manually identified and fixed with digital tools as well as automated processes that blend in neighboring sections to repair missing tissue pieces or estimate simple statistics, such as cell locations, from neighboring sections [7]. This thesis documents the development of a fully automated method to repair a variety of artifacts in histological brain sections using deep generative models.

This method, called Neural Inpainting, is a pipeline consisting of three stages. The first one is the Artifact Localization stage, where a network segments corrupted from uncorrupted areas within a small crop of a scan on a pixel-wise basis. From this, the CPN network [36] computes a binary cell segmentation image that segments cell bodies from the rest. The second stage, the Binary Inpainting, generates information on the locations and shapes of new cell bodies, purely based on these two segmentation images. Lastly, the Image Inpainting stage employs a GAN model, based on the U-Net for biomedical image segmentation [30], to generate new content for the corrupted region. The output of this stage is composed together with the uncorrupted region of the original crop to produce the final image painting.

The result is a method that can process large amounts of image crops without supervision to produce repaired versions of tissue sections and deliver cell statistics, such as amount and size of cell bodies, comparable to ground truth statistics when evaluated on artificially masked data. The Image Inpainting stage provides repaired tissue sections, while cell statistics can be extracted from the repaired cell segmentation image returned by the Binary Inpainting stage. While some results display visually convincing quality, the Image Inpainting model is unable to achieve this consistently, especially on data that differs from the training data, e.g. in terms of the brain of origin, the location within the brain or unusual cutting angles relative to the brain's surface.

This thesis demonstrates the possibility of applying a deep learning method to automate the reparation of histological artifacts on a large scale, which has partly been done manually in the past. By no means does it exhaust the possibilites to do so and many aspects of the model show potential for further improvements.

The code package can be found on GitHub [15].

Contents

1 Introduction

1.1 Julich BigBrain Project

The Julich BigBrain project [1] developed an ultrahigh resolution 3D human brain model that is freely available to help with neuroimaging tasks which rely on accurate reference models. Such reference brains typically lack the level of detail that the Julich BigBrain is able to provide. A single human post mortem brain has been sectioned, stained for cell bodies, scanned at a very high resolution of 20 microns and then digitally reconstructed. This model is currently being extended by further increasing its resolution and involving methods from neuroimaging, brain modelling and AI.

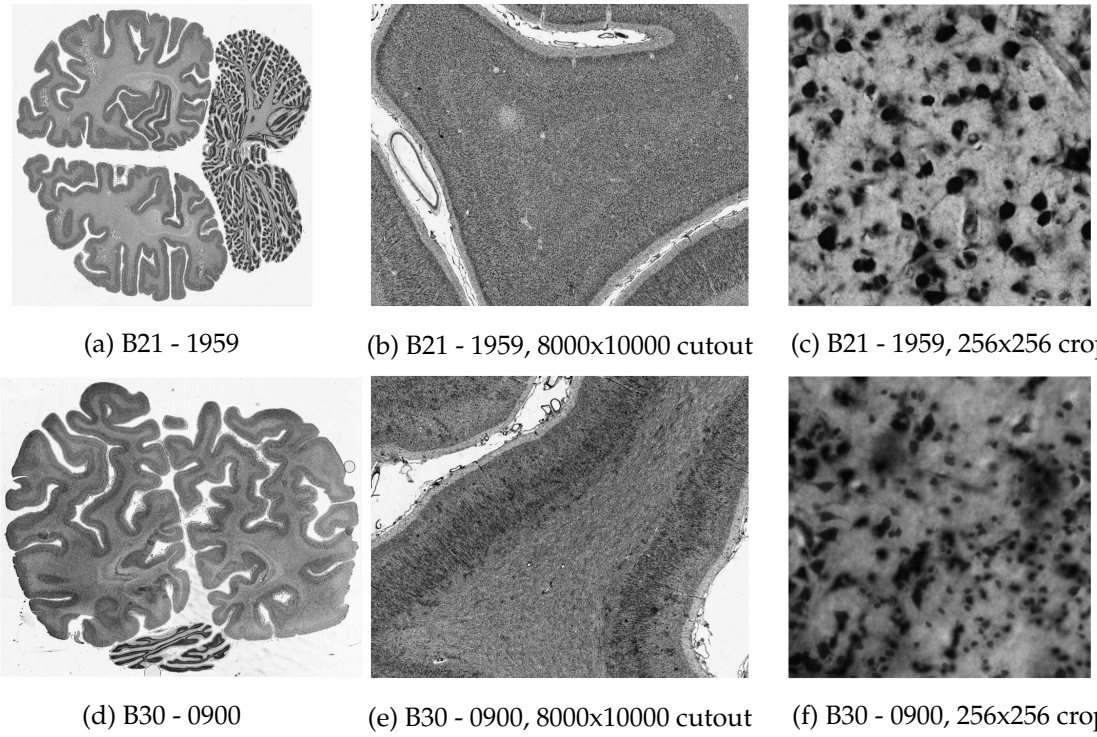
The digital images produced by the high-throughput scanners contain histological artifacts, such as tears and folds of the tissue. These can be due to already existing tissue damage, but are also inevitable during the histological processing. These artifacts vary greatly in size and impact on the data, from blurred regions to large tears. Previously, a large portion of them were manually repaired, supplemented by automated repair algorithms that used neighboring sections to estimate information for missing tissue section [7].

1.2 Histological Brain Tissue Sections

The development and usage of the BigBrain involves methods from histology. This branch of biology is the study of tissues and their structure on a microscopic level. This section describes the data generation process, from brain to digital images, and the challenges involved.

The *institute for neuroscience and medicine, structural and functional organization of the brain* (INM-1) is responsible for the data generation process in the BigBrain and other projects involving reference brain models. A single brain was chosen, coming from a 65-year-old donor at the Heinrich-Heine-University Düsseldorf. The brain was prepared for the cutting process by fixing it in formalin to preserve the proteins and vital structures within the tissue, and embedding it in paraffin wax to stabilize the cutting process. This makes it possible to cut the brain into 20 micrometer thick slices. 7404 histological sections were obtained this way. These were then placed on glass slides and stained for cell bodies using the Merker method. Neurons are stained in a darker tone and the space in between them appears lighter. The INM-1 uses a number of high-throughput scanners to scan the stained sections with a spatial resolution of 1 micron per pixel and a color depth of 16 Bit. A single section can take up to 20 GB of memory.

Each section receives an identification, denoting the brain it is from and the section number. The notation 'B01' denotes the brain number 1, 'B02' number 2 and so on. The sections are simply numbered starting from 0. The section 'B21 - 3037' for example is section 3037 from brain number 21. Data from four different brains was used for development and testing in this thesis, brains B00, B06, B20 and B21. B20 is the brain used in the BigBrain project. BigBrain dataset releases from 2014 and 2015 are freely available under [4].



(a) B21 - 1959 (b) B21 - 1959, 8000x10000 cutout (c) B21 - 1959, 256x256 crop

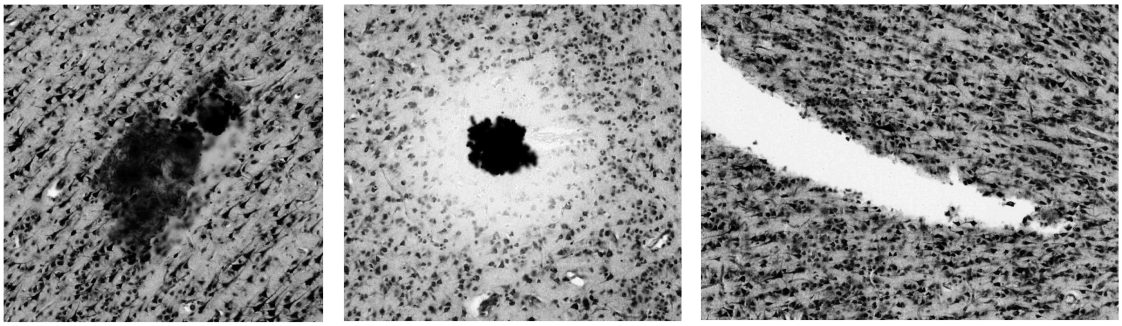
(d) B30 - 0900 (e) B30 - 0900, 8000x10000 cutout (f) B30 - 0900, 256x256 crop

Figure 1: A selection of sections from two different brains. The left column shows entire sections, the middle column shows cutouts of medium size and the right column shows small crops where individual cell bodies are clearly visible. (Data source: Gehirnsammlung INM-1 [2])

During this data generation process, multiple things can occur that result in visual damage to the final images. Physical tissue damage can occur during the preparation and cutting of the brain. The staining process can result in different levels of staining, e.g. depending on the total time of exposure to the chemicals involved. Some sections can appear significantly darker than others, hindering the use of automated methods, such as segmentation or brain mapping. The used chemicals can also stick to unwanted objects, which causes black artifacts of varying sizes. Examples of artifacts are displayed in figure 2

1.3 Research Objective

The presence of artifacts in histological tissue sections causes problems for tasks that focus on any kind of statistics related to cell bodies, because they are obstructed or missing in the scan. Other tasks, such as cell segmentation, are also hindered by the presence of artifacts. Simply ignoring the corrupted areas is often not a good solution, since they appear inconsistently, and a good estimation of the missing content is more adequate. The goal of this thesis is to develop a fully automated method that accurately detects any significant artifacts and generates a repaired image where the artifacts have been replaced



(a) Brain B06, 754x750p (b) Brain B20, 693x650p (c) Brain B21, 1041x787p

Figure 2: Three examples of different histological artifacts from different brains. The first two depict examples where the chemical used for cell staining attatched to something else than cell bodies and obstructs the tissue underneath. The last one is a large tear in the tissue section. (Data source: Gehirnsammlung INM-1 [2])

with new content. For the latter, the focus is on two criteria: Visually convincing quality of the generated image and a meaningful recovery of statistics with respect to cells in the images, e.g. the cell count and cell sizes in specific areas. The term meaningful is meant as in comparable to ground truth data when applying the method to artificially masked images, where the removed areas don't contain artifacts. The method focuses on the accurate generation of information with respect to cells, while other image features, such as fiber orientation in the tissue and histologically correct cell shapes are left for future improvements.

This task is approached within the framework of Deep Learning. The first task, the pixel-wise segmentation of artifacts within an image, is a classical segmentation task that can be learned using annotated data. This sort of image processing task is known to be very approachable with supervised learning and there are already successful architectures for image segmentation on microscopic biomedical images [30]. The second task, the generation of new content in images with accurate cell statistics based on the rest of an image, requires modeling complex, conditional distributions. In combination with the large amount of data that is available through the INM-1, this problem setup lends itself to the use of deep generative models.

Lastly, code for the method and all results in this thesis is provided as a python package on GitHub [15].

1.4 Approach Overview

The approach that is presented in this thesis is the **Neural Inpainting** pipeline, which consists of three different stages. Figure 3 provides a high level overview of this pipeline. The first stage, the **Artifact Localization**, takes in a small crop of a light microscopic scan of a human brain tissue section, that potentially contains artifacts and segments the image into corrupted and uncorrupted areas, pixel-wise. This is a supervised task, learned

on manually annotated scans of real artifacts. The second stage, the **Binary Inpainting**, operates entirely on cell segmentation images, provided by the CPN network [36]. This representation effectively discards all textural information in the crop and only keeps the semantic information with respect to cell bodies. A Normalizing Flow model is used to perform inpainting on the cell segmentation image and provide a binary painting that contains information on the locations and shapes of new generated cells. This is an unsupervised inpainting task which is learned on artificially masked data. The last stage, the **Image Inpainting**, then performs inpainting on the segmented crop from the first stage, but taking the result of the second stage into account as an additional input. This is accomplished using a heavily modified version of the U-Net [30] in a GAN setup, which can now focus primarily on the textural content of the image, because the binary painting contains the relevant semantic information. The result is a repaired crop, while the second stage is able to generate information on new cells with statistics such as cell density, size and eccentricity that are comparable to ground truth statistics when tested on artificially masked examples.

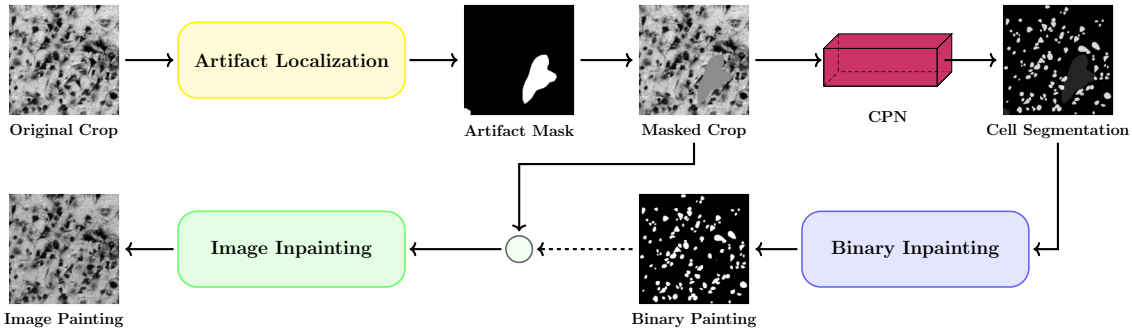


Figure 3: High level overview of the three stage Neural Inpainting pipeline. Given a masked crop of a light microscopic scan of a human brain tissue section, the CPN network [36] provides a cell segmentation image that is completed by the Binary Inpainting stage to provide an auxiliary input to the Image Inpainting stage.

The remainder of this chapter provides the necessary background knowledge on generative models and introduces the relevant model types. A good understanding of Deep Learning basics such as backpropagation, feed-forward architectures and convolutional networks is assumed.

Chapter 2 of this thesis introduces a modular architecture that is used for different models in the Neural Inpainting pipeline, followed by a detailed description of all relevant parts of the method. The setup and processing of the datasets is described, as well as an explanation and motivation of the approach. The remaining sections detail the three stages of the Neural Inpainting pipeline step by step.

Chapter 3 contains a series of experimental evaluations of all stages, including comparisons to alternative approaches and models. It is structured by experiments for easy comparison between models.

Lastly, chapter 4 discusses the results of the experiments section, states finding, limita-

tions and opportunities for future work, before concluding the thesis in section 4.2.

1.5 Background on Generative Models

The goal of this section is to introduce the concept and design of generative models for the purposes of this thesis. The core concepts behind **Generative Adversarial Networks** (GANs) and **Variational Autoencoders** (VAEs) are briefly presented. The third model class which is relevant to this thesis, **Normalizing Flow** (NF) models, is explained in more detail, since it is more complex and less established than the other two. For all three model classes, material on more complex variants of these, as well as material on more elaborate explanations is referenced in their respective segments.

Given N i.i.d. samples from an intractable probability distribution $p_{data} \in \mathbb{R}^D$, the goal of generative modelling is to generate new samples with similar features as the existing data samples. The number of samples n is usually large and the distribution p_{data} too complex or high-dimensional for classical statistical inference. An analytical expression for p_{data} is not feasible here and the evaluation of the likelihood of individual data points is desired but not always possible. Generative models typically approach this problem by learning a mapping from a tractable distribution $Z \in \mathbb{R}^d$, e.g. a Gaussian, to the complex data distribution. This mapping is called the **generator** $g_\theta : \mathbb{R}^d \to \mathbb{R}^D$ and can be implemented as a neural network with learnable parameters θ. The generation process of new samples then becomes a two-step process of sampling $z \sim Z$ and mapping it via g_θ to $g_\theta(z) \approx x \sim p_{data}$. Z is known as a **latent space**. Its properties are a design choice of the model and are sometimes learned during training, e.g. by parameterizing a distribution by a neural network. While Gaussian distributions are generally accepted as a good choice, the correct choice of the dimension d of the latent space remains an open challenge in generative modeling.

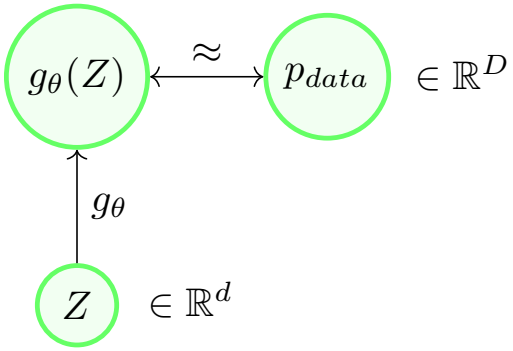


Figure 4: A base distribution Z is mapped to a more complex distribution $p_\theta(Z)$ via the generator g_θ and then compared with the data distribution p_{data}. The function parameters θ can then be updated based on this comparison.

Since this falls within the realm of unsupervised learning, classical objective functions involving labels can't be used here. Instead, a key component of the learning process

is a measure of similarity between generated samples and real samples. The choice of this component is a major difference between different types of generative models and can determine model properties such as possible evaluation of the likelihood of samples, restrictions on the dimensionality of the latent space and training characteristics of the model.

The following three types of generative models all have different approaches in this regard and offer unique advantages and disadvantages. This paper by [32] provides a comprehensive introduction to the topic of generative modelling with a focus on the three model types GAN, VAE and NF models.

1.5.1 Generative Adversarial Networks

GANs are a well established type of generative models. The concept was introduced in [11] and since has been subject to further research and countless modifications. A GAN architecture consists of two models, a generator G and a discriminator D, which are jointly optimized. G and D are differentiable functions that are parameterized by neural networks. Given a data distribution p_{data} and a source distribution p_z with efficient and exact sampling, e.g. a Gaussian distribution, the original training objective defines a minmax-game between G and D:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

The discriminator is trying to discriminate between real data samples $x \sim p_{data}$ and the generated outputs $G(z)$, while the generator is trying to produce outputs which D classifies as real data samples.

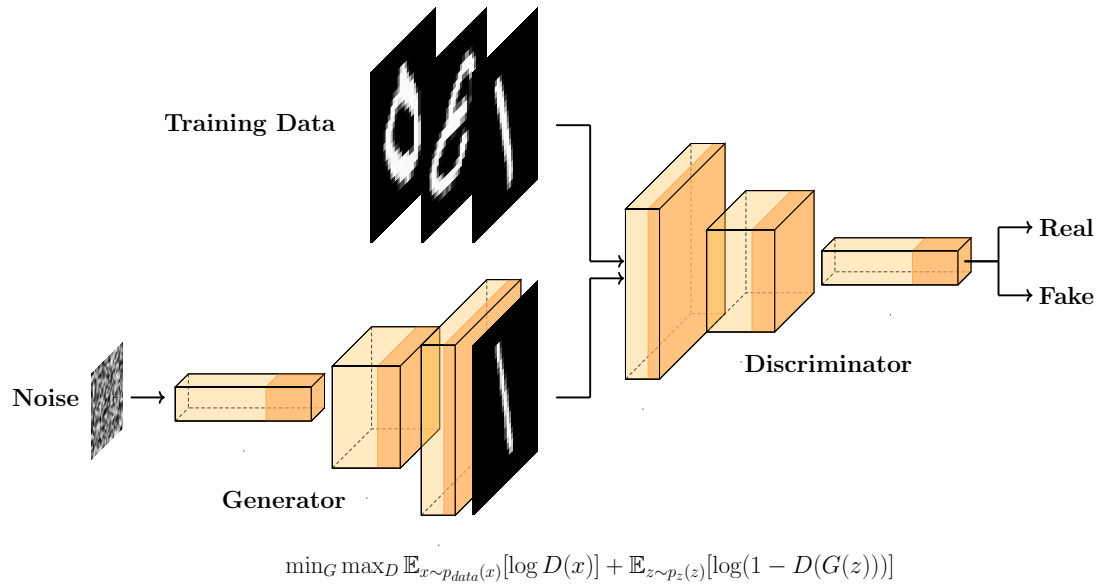


Figure 5: Vanilla GAN architecture with example images from the MNIST dataset [23].

In [11], sec. 4, it was shown that the model distribution p_G converges to the data distribution p_{data} if the discriminator is updated enough times to reach its optimum given G and if both networks have sufficient capacity. The aforementioned condition might need an arbitrarily large number of optimization steps for D in order to be satisfied. This is not practical in real world scenarios, so there is no guarantee for the convergence of a GAN model.

A large portion of GAN research is dedicated to improving the training stability and convergence behavior of GANs [3], [25], [26], [33]. Even with these improvements however, the set of hyperparameters that result in stable training and avoid problems such as mode collapse or oscillatory convergence behavior can be very small in practice. This is perhaps the biggest disadvantage compared to other generative models. Strengths of GANs are the generation of convincing data samples, especially on image data [39], fast and scalable training [6] and sampling as well as good performance on high dimensional data [37]. The tradeoff is that they have difficulty to capture the whole data distribution and tend to focus on specific parts, since this is enough to fool the discriminator.

This survey paper by [14] covers model variants, training issues and improvements as well as analysis of various GAN applications. A more in-depth explanation and illustration of the basics of generative models and GANs can be found in this article by [28].

1.5.2 Variational Autoencoders

VAEs were introduced by [20] and have since been, similar to GANs, subject to further research and modifications. Before looking at the specifics of VAEs, we quickly introduce **likelihood based models**. Given an unknown true data distribution $p_{data}(x)$, data samples $\{x^{(i)}\}_{i=0}^{N} \sim p_{data}(x)$ and a model distribution $p_\theta(x)$, parameterized by θ, the goal of these models is to maximize the likelihood $p_\theta(x^{(i)})$ for all given data samples $x^{(i)}$ during training:

$$\theta_{opt} = \arg\max_\theta \sum_{i=1}^{N} p_\theta(x^{(i)}) = \arg\min_\theta \sum_{i=1}^{N} -\log p_\theta(x^{(i)})$$

While VAEs fall into the group of likelihood based models, they don't optimize this objective directly, but use a surrogate objective that is a lower bound on the marginal likelihood $p_\theta(x)$. This can be useful when the marginal likelihood is intractable.

Now to be specific, in the problem scenario of a VAE we assume that there exists a **latent** random variable z with unknown distribution p_{θ^*} which captures the underlying structure of the data. Each data point x is the result of sampling $z \sim p_{\theta^*}(z)$ first and then sampling $x \sim p_{\theta^*}(x|z)$. All three of these distributions are unknown. In order to approximate the true data generation process, we introduce two models: $q_\phi(z|x)$ and $p_\theta(x|z)$. These are known as probabilistic **encoder** and **decoder** in the VAE architecture. Given a data sample x or latent representation z, they output a distribution over the latent space or data space. The goal is now to find a model $q_\phi(z|x)$ which approximates the true posterior $p_\theta(z|x)$ and we need a way of learning the model parameters ϕ and θ jointly.

The **KL-Divergence** (also known as relative entropy) is a common distance measure between distributions and provides an elegant way of deriving the VAE objective. Given

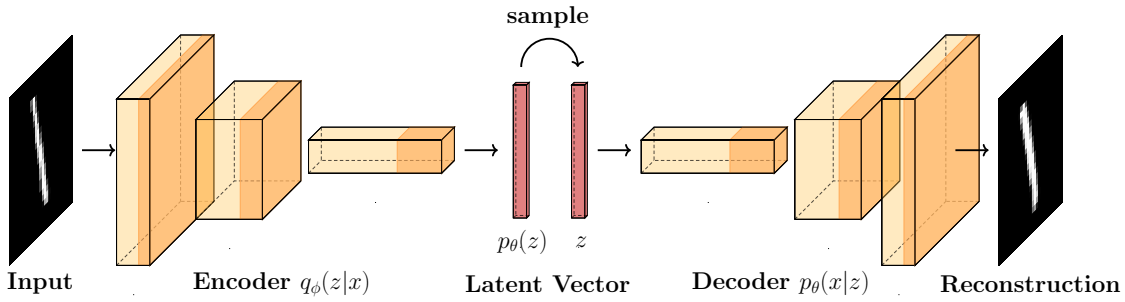


Figure 6: Vanilla VAE architecture with example images from the MNIST dataset [23].

two continuous real-valued random variables P and Q, the KL-Divergence is defined as

$$D_{KL}[P||Q] = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

We can now derive the **Evidence Lower Bound** (ELBO):

$$\begin{aligned}
D_{KL}[q_\phi(z|x)||p_\theta(z|x)] &= \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} dz \\
&= -\int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)p_\theta(x)} dz \\
&= -\int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz + \int q_\phi(z|x) \log p_\theta(x) dz \\
&= -\int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz + \log p_\theta(x) \geq 0 \\
\Leftrightarrow \log p_\theta(x) &\geq \int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz =: L(\theta, \phi, x)
\end{aligned}$$

In the last line, we used the fact that the KL-Divergence is always non-negative. This now means that $L(\theta, \phi, x)$ is a lower bound on the log-likelihood of the data and we can use it as a surrogate objective to maximize the likelihood of the data with respect to the model parameters θ and ϕ. With a few more tricks, we can optimize this via standard stochastic gradient descent methods. The training objective $L(\theta, \phi, x)$ can be written as:

$$\begin{aligned}
L(\theta, \phi, x) &= \int q_\phi(z|x) \log \frac{p_\theta(z,x)}{q_\phi(z|x)} dz \\
&= \int q_\phi(z|x) \log \frac{p_\theta(z)}{q_\phi(z|x)} dz + \int q_\phi(z|x) \log p_\theta(x|z) dz \\
&= -D_{KL}[q_\phi(z|x)||p_\theta(z)] + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]
\end{aligned}$$

The expectation here can be interpreted as a reconstruction-error term and the KL-Divergence term acts as regularization on the latent space. It is common practice to choose a simple distribution for $p_\theta(z)$, usually a standard normal Gaussian. Notably, the lower bound $L(\theta, \phi, x)$ reaches equality with the log-likelihood of the data distribution iff the approximation $q_\phi(z|x)$ is equal to the true posterior $p_\theta(z|x)$ almost everywhere.

In order to approximate the expected reconstruction error $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ via sampling the so-called **reparameterization trick** is employed. It allows us to rewrite an expectation w.r.t. $q_\phi(z|x)$ such that the Monte Carlo estimate of the expectation is differentiable w.r.t ϕ. Formally, we can express a random variable $z \sim q_\phi(z|x)$ as a deterministic variable $z = g_\phi(\epsilon, x)$, where ϵ is an auxiliary random variable with independent marginal $p(\epsilon)$ and $g_\phi(.)$ is a deterministic function parameterized by ϕ. In the gaussian case for example, let $\epsilon \sim N(0,1)$ and $g_\phi(\epsilon) = \mu + \sigma\epsilon$, $\phi = (\mu, \sigma)$. Then $z = g_\phi(\epsilon) \sim N(\mu, \sigma^2)$. Therefore, we can rewrite an expectation w.r.t $N(z; \mu, \sigma^2)$ as

$$\mathbb{E}_{N(z;\mu,\sigma^2)}[f(z)] = \mathbb{E}_{N(\epsilon;0,1)}[f(\mu + \sigma\epsilon)] \simeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma\epsilon^{(l)}) \quad \text{with } \epsilon^{(l)} \sim N(0,1)$$

This was a short version of the full derivation of the reparameterization trick from [20], chapter 2.4.

In the above, a few technical details were left out, mostly assumptions on the distributions like differentiability w.r.t θ and ϕ and details in the reparameterization trick. The original paper [20] contains a complete derivation of the VAE method. A more intuitive explanation with good illustrations for this method can be found in this article by [29]. The authors of the original paper also released a greatly expanded version of their earlier work, which explains the topic in finer detail and gives more elaborate context: [19].

A strength of VAEs is the generality of the method. There is a lot of freedom in the choice of the models and the involved distributions, which potentially allows for the inclusion of prior knowledge of the data distribution or underlying data structure. VAEs are relatively easy to optimize and it is easy to compare the performance of two VAEs by comparing their estimated log-likelihood, while it is unclear how to quantitatively compare the performance of two GANs. When looking at image data however, VAEs tend to be outperformed by GANs in terms of sample quality [5]. A crucial difference is that a GAN can assign a training sample effectively 0 probability under the model, because it is not a maximum likelihood model. For a VAE, this would result in an infinitely high loss.

1.5.3 Normalizing Flow Models

This introduction to Normalizing Flow models follows the structure of the comprehensive survey paper by [21], with added details and explanation on the decomposition of the maximum likelihood objective. More information on background, applications and model types which are not relevant to this thesis, can be found in chapters 1-3 of the survey paper.

Concept

Normalizing Flow (NF) models are, together with the popular GAN and VAE models, members of the family of generative models. They are also **likelihood based models**, which means that given an unkown true data distribution $p_{data}(x)$, data samples $x^{(i)} \sim p_{data}(x)$ and a model distribution $p_\theta(x)$, parameterized by θ, the model is trained

by maximizing the likelihood $p_\theta(x^{(i)})$ for all given data samples $x^{(i)}$:

$$\theta_{opt} = \arg\max_\theta \sum_{i=1}^{N} p_\theta(x^{(i)})$$

The unique property of NF models is that they optimize this likelihood directly, alleviating the need for surrogate objectives, such as the ELBO used in VAEs. The standard objective for NF models is

$$\theta_{opt} = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} -\log p_\theta(x^{(i)})$$

To be able to optimize this objective directly, we need to be able to evaluate $p_\theta(x^{(i)})$. The core idea here is to use the **Change of Variables Formula** to map a simple distribution with efficient sampling to our more complex data distribution.

Theorem 1. *(Change of Variables Formula). Let X and Z be random variables and $f : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible map such that $X = f(Z)$. Then*

$$p_X(x) = p_Z(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

By parameterizing the function f, e.g. as a neural network with parameters θ, we can learn it from data samples $x \sim p_{data}(x)$. This imposes several restrictions on the neural network architecture:

1. f needs to be an invertible function with inverse $g := f^{-1}$
2. f needs to be sufficiently expressive to accurately model the target distribution
3. f and/or g need to be efficient to compute, f for learning and g for sampling
4. The determinant of the Jacobian of f^{-1} needs to be efficient to compute

In practice, we use a simple base distribution Z, e.g. a Gaussian, and design an invertible network architecture, whose forward pass and Jacobian is efficient to invert, while being expressive enough to learn the mapping from our base distribution to the target distribution.

The transformation from base distribution to target distribution is called **generative direction** and the inverse is called **normalizing direction**, hence the name 'Normalizing' Flow.

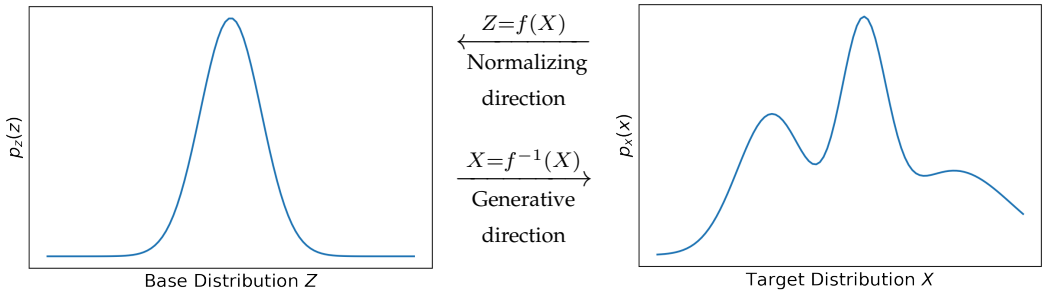


Figure 7: Change of variables between the density functions of the base distribution $p_z(z)$ and target distribution $p_x(x)$.

To achieve the desired expressivity, we can concatenate multiple invertible networks $f_1, ..., f_K$ with simple Jacobians to obtain a more expressive network:

$$f = f_1 \circ ... \circ f_K$$

With $g_k = f_k^{-1}$:

$$f^{-1} = g_K \circ ... \circ g_1 = g$$

and determinant of the Jacobian

$$\det(Df(x;\theta)) = \det\left(\prod_{k=1}^{K} Df_k(x_k;\theta)\right) = \prod_{k=1}^{K} \det(Df_k(x_k;\theta))$$

The invertibility is preserved here, because a chain of invertible functions is again invertible, and the final Jacobian is simply the product of all Jacobians of the smaller networks. Because the determinant is a multiplicative map on square matrices, the determinant of the Jacobian of g is the product of the determinants of all g_k . This concatenation of transformations is responsible for the second part of the name, Normalizing 'Flow'.

We can now start to write out the maximum likelihood objective into terms we can compute. First, consider the i-th inverse transformation in the flow and denote $z_{k-1} := f_k(z_k)$ for $k = 1, ..., K-1$. $z_K := x$, $z_0 := z$ and $z_k \sim p_k(z_k)$. For readability, we omit the parameters of f: $f_k(z_k) := f_k(z_k;\theta)$. Then

$$p_k(z_k) = p_{k-1}(f_k(z_k)) \left|\det \frac{\partial f_k(z_k)}{\partial z_k}\right|$$
$$\Rightarrow \log p_k(z_k) = \log p_{k-1}(z_{k-1}) + \log \left|\det \frac{\partial f_k(z_k)}{\partial z_k}\right|$$

This allows us to decompose the log likelihood of a single sample as follows.

$$\begin{aligned}\log p(x) = \log p_K(z_K) &= \log p_{K-1}(z_{K-1}) + \log \left|\det \frac{\partial f_K(z_K)}{\partial z_K}\right| \\ &= \log p_{K-2}(z_{K-2}) + \log \left|\det \frac{\partial f_{K-1}(z_{K-1})}{\partial z_{K-1}}\right| + \log \left|\det \frac{\partial f_K(z_K)}{\partial z_K}\right| \\ &= ... \\ &= \log p_0(z_0) + \sum_{k=1}^{K} \log \left|\det \frac{\partial f_k(z_k)}{\partial z_k}\right|\end{aligned}$$

The whole maximum likelihood objective becomes

$$\arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)}) = \arg\max_\theta \sum_{i=1}^{N} \log p_z(f(x^{(i)};\theta)) + \sum_{k=1}^{K} \log \left|\det \frac{\partial f_k(z_k^{(i)};\theta)}{\partial z_k^{(i)}}\right|$$

All terms on the left side can be computed after a foward pass through the model. $p_z(f(x^{(i)};\theta))$ is the likelihood of the input sample $x^{(i)}$ after the complete model transformation f and $\det \frac{\partial f_k(z_k^{(i)};\theta)}{\partial z_k^{(i)}}$ are the determinants of the Jacobians of each individual layer f_k. In practice, this sum is accumulated during the foward pass.

[21] contains a comprehensive list of different types of NF models. For now, only the types of NF models which appear in this thesis are introduced, namely **Coupling and Autoregressive Flows**.

Coupling and Autoregressive Flows

Coupling and Autoregressive Flows are two types of NF designs which employ similar strategies to increase the expressivity of a network while maintaining an upper triangular Jacobian matrix. They are the most common type of NF with well known NF models, such as **RealNVP** [9] and Glow [17], being a Coupling Flow. This section closely follows the sections 3.4.1 and 3.4.2 from [21].

Coupling Flows

Introduced by [8], Coupling Flows are characterized by their use of **coupling layers**. Consider an input $x \in \mathbb{R}^D$ and an invertible function $h_\theta : \mathbb{R}^d \to \mathbb{R}^d$, which is parameterized by θ. If we now partition x into two subsets $(x^A, x^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$, we can define the invertible function $g : \mathbb{R}^D \to \mathbb{R}^D$:

$$\begin{aligned}y^A &= h(x^A, \Theta(x^B)) \\ y^B &= x^B\end{aligned}$$

with inverse $g^{-1} : \mathbb{R}^D \to \mathbb{R}^D$:

$$\begin{aligned}x^A &= h^{-1}(y^A, \Theta(x^B)) \\ x^B &= y^B\end{aligned}$$

The **coupling function** h is parameterized by an arbitrary function $\Theta(x^B)$. $\Theta(\cdot)$ is called a **conditioner**. The funtion g corresponds to a single layer g_k from the concept segment above. For simplicity, we omit the index here. We can now adjust the expressivity of this layer by using different conditioners, e.g. neural networks. The Jacobian matrix of g has the form

$$Dg(x) = \frac{\partial g(x)}{\partial x} = \begin{bmatrix} \frac{\partial y^A}{\partial x^A} & \frac{\partial y^A}{\partial x^B} \\ \frac{\partial y^B}{\partial x^A} & \frac{\partial y^B}{\partial x^B} \end{bmatrix} = \begin{bmatrix} \frac{\partial h(x^A)}{\partial x^A} & \frac{\partial h(x^A)}{\partial \Theta(x^B)} \\ 0 & I_d \end{bmatrix}$$

This implies that the determinant of $\frac{\partial g(x)}{\partial x}$ is simply the determinant of $\frac{\partial h(x^A)}{\partial x^A} = Dh(x)$.

The partitioning of x is an open design choice here. Also, if we were to use multiple coupling layers after each other, the second subset would always be passed through without any transformation. This is why Coupling Flows employ some form of permutation of x between coupling layers. For example, **RealNVP** [9] uses an alternating checkerboard pattern, such that the parts of x which were passed through in one layer are updated in the next one. The **Glow** model [17] uses learnable 1x1 convolutions as a generalization of a permutation operation.

Autoregressive Flows

The idea of an Autoregressive Flow first appeared in [18]. Similarly to Coupling Flows, we define an invertible function $h_\theta : \mathbb{R} \to \mathbb{R}$, parameterized by θ, and call it a coupling function. Define $g : \mathbb{R}^D \to \mathbb{R}^D$ with $y = g(x)$. For $t = 1, ..., D$, the output of g is defined via the coupling function as

$$y_t = h(x_t, \Theta_t(x_{1:t-1}))$$

Again, the functions $\Theta_1, ..., \Theta_D$ are called conditioners and can be chosen as arbitrary functions, except for Θ_1 which is constant. Note, that we have a similar structure to coupling flows, namely the outputs are computed via a coupling function which uses part of x as an input and is parameterized by other parts of x. In Coupling Flows, we build a partition of x, while here we move sequentially through the entries of x.

Due to the autoregressive structure, the Jacobian matrix Dg is triangular with determinant:

$$\det(Dg) = \prod_{t=1}^{D} \frac{\partial y_t}{\partial x_t}$$

The forward pass of g can be implemented with an autoregressive network and does not need to be computed sequentially, because each output y_t solely depends on the previous inputs $x_{1:t}$. The **Masked Autoregressive Flow (MAF)** model [27] uses this idea. For the inverse

$$x_t = h^{-1}(y_t, \Theta(x_{1:t-1}))$$

however, each output x_t depends on all previous outputs $x_{1:t-1}$. This needs to be computed sequentially and thus cannot be parallelized. [18] used this observation to formulate the **Inverse Autoregressive Flow (IAF)** model, which is effectively the inverse transformation to MAF. It defines the forward pass as

$$y_t = h(x_t, \Theta(y_{1:t-1}))$$

which implies the inverse to be

$$x_t = h^{-1}(y_t, \Theta(y_{1:t-1}))$$

Now, the forward pass needs to be computed sequentially and the inverse can be parallelized. Under a small restriction, MAF and IAF are mathematically equivalent ([27] Appendix A). Their tradeoff is in computational efficiency: MAF has a fast normalizing direction (likelihood computation and learning) and a slow generative direction (sampling), vice versa for IAF.

2 Methods and Materials

2.1 Modular U-Net for Inpainting

The U-Net [30] is a fully convolutional encoder-decoder model which was originally proposed for biomedical image segmentation . Both, in the final version and in the development of the Neural Inpainting model, variations of the U-Net model were used. The result is a modular architecture that is called the modular U-Net for the rest of this thesis. This section gives an overview over the model architecture that was used in this project. It is modular in the sense that different parts of the architecture can be omitted or added depending on the task. Later in this chapter, several variations of the full architecture are specified by excluding features of the following full architecture.

In the following, the term convolution always refers to a convolutional layer as it is common in deep learning, not the mathematical convolution operation.

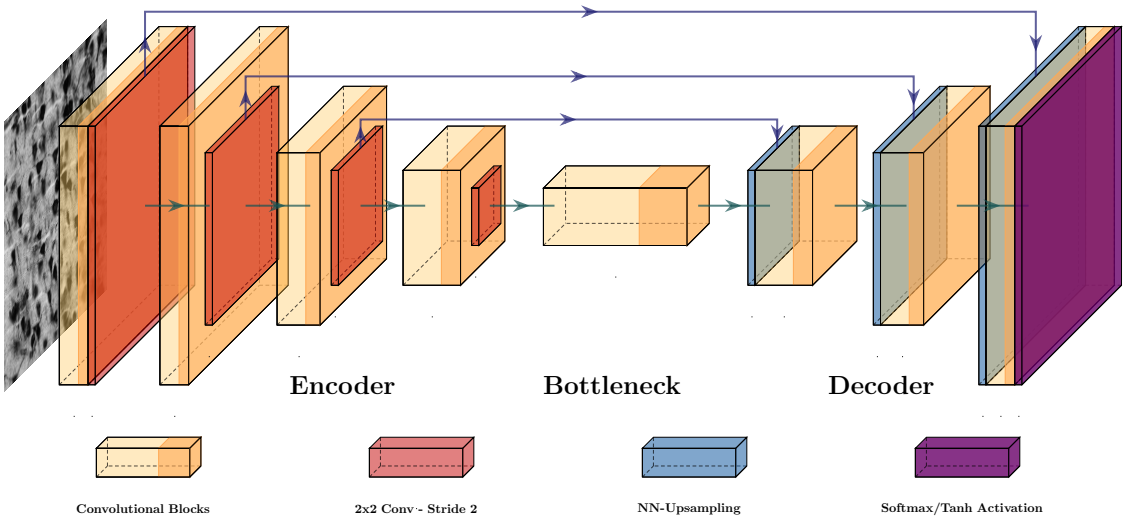


Figure 8: Overview of the Modular U-Net Architecture. The very first and last block are the input and output block which achieve the correct number of channels for the encoder and model output respectively.

Overview The model consists an encoder section, a bottleneck and a decoder section. The encoder contains three **U-Net blocks** with each of them decreasing the feature dimension and increasing the number of channels. The bottleneck is a series of dilated convolutions, normalization and activation layers. Feature and channel dimensions remain constant here. The decoder contains two U-Net blocks with each of them increasing the feature dimension and decreasing the number of channels. There is one less U-Net block in the decoder than there is in the encoder because the end of the bottleneck includes the first upsampling layer. This is an implementational restriction because of the skip-connections between encoder and decoder. At the very beginning of the model, there is an input block. The only difference to a U-Net block is that the channel dimension is

not doubled, but scaled from the channel dimension of the input to a channel dimension which can be chosen as a width parameter of the model. Similarly, at the end of the model there is an output block. The first skip connection connects these two blocks.

The following paragraphs describe different parts of the architecture and training setup. Skip connections and U-Net blocks are from the regular U-Net, only with a modification to the upsampling layers. All other parts are new or significantly modified compared to the regular U-Net.

Skip Connections These are the characteristic feature of the U-Net: After each block in the encoder the output is not only passed onto the next layer in the encoder but also copied over to the block in the decoder with matching feature and channel dimension. There, it is concatenated with the output of the previous block in the decoder to form the input for the next block in the decoder, see figure 8.

U-Net Block A U-Net block consists of three parts. The first is a convolutional layer which increases or decreases the number of channels, followed by a batch normalization layer [12] and an activation layer. The second part is a convolutional layer with a residual connection and again followed by batch normalization and activation layers. The last part is a layer to upscale or downscale the feature dimension. For downscaling the model employs standard convolutional layers with kernel size 2 and stride 2. For upscaling nearest-neighbor upsampling is used, instead of transposed convolutions which are used in the regular U-Net, as these can cause checkerboard artifacts. The upsampling also comes with a reduction in the number of channels by a factor of 2. This is achieved with a standard convolutional layer with kernel size 1. For all convolutional layers in the above for which the kernel size was not specified it is a design choice of the model, kernel size 3 being the most common choice.

Variable Model Width The model has a hyperparameter that determines the number of channels of each convolutional layer. This parameter sets the number of output channels of the input block. All subsequent convolutional layers have their number of channels determined by this, because each U-Net block scales this number up or down by a factor of 2.

Normalization Layers For normalization there are two options: Batch normalization [12] and spectral normalization [26]. The former is a popular form of input normalization where the inputs to a layer are normalized to approximately have mean 0 and standard deviation 1. The latter is a form of weight normalization which normalizes the weights of a layer using an estimate of their eigenvalues. Its intended use is in the discriminator of a GAN to stabilize the training, although it can also be used in the generator.

Activation Layers This model employs two possible activation layers: ReLU and LeakyReLU. ReLU is perhaps the most popular activation function in neural networks. It was used long before the recent success of deep learning. A very early occurrence is

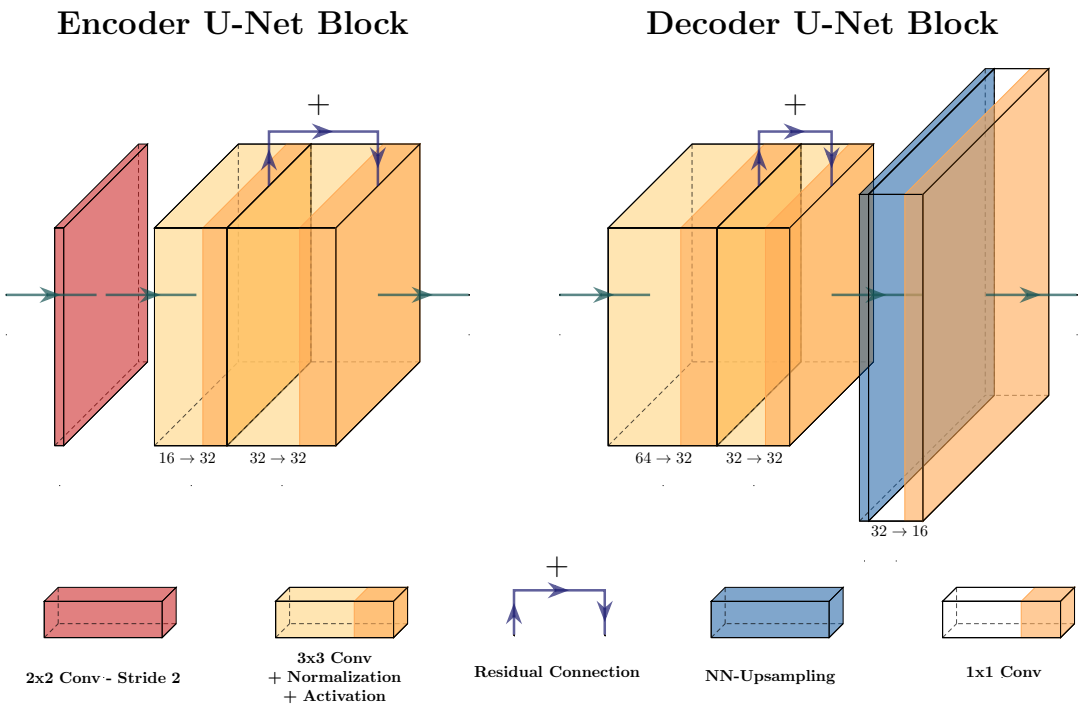


Figure 9: More details on the U-Net block. The numbers below the convolutional blocks are an example of the transformation of the number of channels of the input.

in [10]. LeakyReLU is a modification of ReLU, first proposed in [**redmon2016**], which provides a non-zero gradient for every input as opposed to the ReLU function. It is a popular choice in GAN networks.

Gated Convolutions Gated convolutional layers are a variation of the standard convolutions [39]. They were specifically designed for inpainting tasks on image data where a binary mask, used to mask a number of pixels in the image, is passed as an additional input to the network. They provide a learnable, element-wise weighting of the output of a convolutional layer, where each output is multiplied with a weight between 0 and 1.

The output formula for a single layer with gated convolutions is:

$$\begin{aligned} Gating &= \sum\sum W_g \cdot I \\ Feature &= \sum\sum W_f \cdot I \\ Output &= \phi(Feature) \odot \sigma(Gating) \end{aligned}$$

for input I, convolutional filters W_g and W_f and arbitrary activation function ϕ. *Gating* and *Feature* are computed via regular convolutions and the results are multiplied element-wise and with different activation functions.

If used, these replace all standard convolutional layers in the network, with the exception of the 1x1 convolutions used for scaling the number of channels. This roughly doubles

the number of learnable parameters, since an additional convolution of the same size needs to be performed for each standard convolution.

Bottleneck with Dilated Convolutions The bottleneck is a small series of convolutional blocks, including normalization and activation layers with constant feature and channels dimensions. It is a common practice in GAN based inpainting networks to use dilated convolutions here [38], [39], [40] to achieve full coverage of the internal representation by the receptive field.

GAN Loss In a regular GAN setup, the U-Net architecture would be used as a generator together with an encoder-like discriminator that outputs a scalar prediction on whether the input was real or fake. However, it can also be utilized as a discriminator, as described in [35]. The GAN objective needs to modified to be applied element wise to every pixel of the output. Effectively, the discriminator judges each pixel individually instead of the whole image. This is a perfect fit for inpainting tasks, since we no longer need to pass fake and true images separately to the discriminator during training. Instead, the discriminator's objective is to learn to distinguish the real and fake parts of a painting.

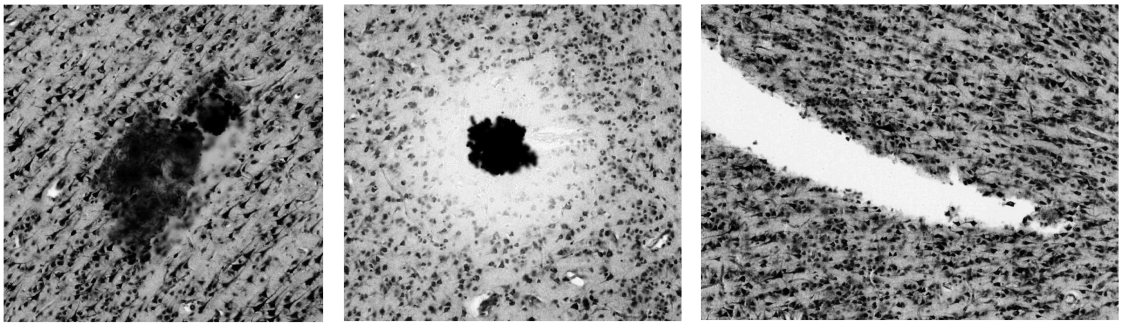
2.2 Training Data

Two datasets were used for training, one that purposefully contains at least one artifact per scan and one that contains as much intact tissue as possible, although smaller artifacts are also present here. The two training datasets are composed of light microscope scans of human brain tissue sections, as described in section 1.2. They contain single-channel, gray scale images with a spatial resolution of 1 micron per pixel. Because the cortical regions with their clearly visible, dense population of cell bodies are of interest here, smaller cutouts of cortical patches are saved instead of scans of entire slices.

The artifact dataset contains manually created cutouts of artifacts with image dimensions in the 4-digit pixel range and below. This is still too large for a network to process quickly, but rather small compared to the sizes of an entire slice. The dataset consists 45 of these cutouts, which amounts to more than 1 GB of artifact data. Each cutout was annotated manually to label the corrupted area pixel wise. The annotation covers slightly more than the artifacts to ensure that they are always completely removed. Most artifacts tend to have a blurry transition to the intact part of the image. These need to be removed fully since the part which is labelled as uncorrupted by the network are copied over into the final painting. The data here comes from different regions of 4 different brains, B00, B06, B20 and B21.

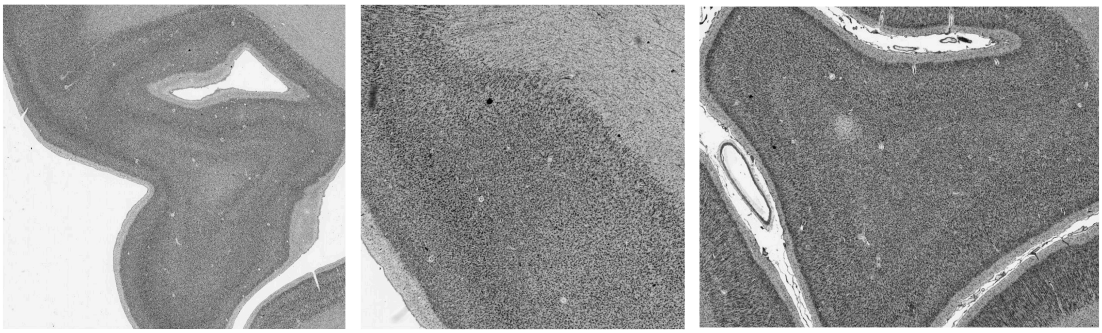
The dataset of intact tissue contains larger cutouts of cortical patches with image dimensions from 18000x17000 pixels down to 6000x6000. It consists of 7 cutouts, all coming from brain B21, totaling almost 1GB of intact tissue data. While these cutouts also contain artifacts, most of the tissue is intact and they serve as ground truth data for feedback on what intact tissue should look like.

Considering the internal representations in feed-forward neural networks, these image sizes come with high demands for memory on the GPUs that process them. Also, while



(a) Brain B06, 754x750p (b) Brain B20, 693x650p (c) Brain B21, 1041x787p

Figure 10: Three examples of different histological artifacts from different brains. The first two depict examples where the chemical used for cell staining attatched to something else than cell bodies and obstructs the tissue underneath. The last one is a large tear in the tissue section. (Data source: Gehirnsammlung INM-1 [2])



(a) Brain B21-3037 (b) Brain B21-3037 (c) Brain B21-1959

Figure 11: Three examples of cutouts that make up the intact dataset. (Data source: Gehrinsammlung INM-1 [2])

fully convolutional architectures are technically able to process different image sizes with the same network, this does not apply to all networks in the Neural Inpainting pipeline. Hence, for all training and testing purposes, the cutouts from the datasets are cropped to a smaller, constant and quadratic image size. This size is a hyperparameter choice that needs to be consistent across the Neural Inpainting pipeline. Since the Artifact Localization network is fully convolutional, it technically works on every crop size, even without retraining. The encoder-decoder structure of the U-Net imposes the small restraint of divisibility by 8, because the feature dimension is halved three times along both axes. Ultimately, the crop size is mostly determined by the Normalizing Flow network in the Binary Inpainting stage, since it has the least flexibility here. Common crop sizes during the development ranged from 256x256 to 384x384. These do not necessarily have to be square.

For the remainder of this thesis, the word 'crop' refers to these quadratic image crops

of constant size that serve as input data to the networks. During training, these crops are generated as follows: A cutout from the dataset is chosen, then a random pixel in that cutout is selected as the top left corner for the crop. A crop with the given crop dimension is cropped out. The total brightness of the crop checked to see whether it contains enough tissue area. Since the cortical area is close to the surface of the brain, most cutouts also contain white background areas that are not part of the brain. If the crop contains too much white, it is discarded and a new corner pixel is chosen. This process is repeated until the desired amount of crops for a batch have been selected. The batch is normalized to the range [-1,1] and simple data augmentation, including horizontal and vertical flipping, as well as 90 degree rotations is applied.

The original data format is UINT8 with a single color channel. All training crops are normalized to the range [-1,1] by dividing by 127.5 and subtracting 1. Note that the mean pixel value is not necessarily 0 after this.

2.3 Problem Analysis and Approach

In this section, the task of finding and repairing artifacts in histological brain sections is analyzed and the approach motivated. While sections 2.4 to 2.6 give a detailed description of the final approach, this section aims to illustrate the reasoning behind why certain parts were or were not included and how the three stage structure emerged during development of this method. Firstly, section 2.3.1 provides a concise overview of the final approach.

2.3.1 Neural Inpainting Overview

The complete Neural Inpainting Pipeline contains three individual stages. The first stage is the **Artifact Localization** whose goal it is to detect histological artifacts within a crop and to return a binary segmentation mask which assigns a label to each pixel based on whether it belongs to the corrupted region of the crop or not. This is a supervised learning task trained on manually annotated data.

The second stage, the **Binary Inpainting** consists of two steps, the **Density Inpainting** and **Shape Inpainting**, where an inpainting task is completed on a cell segmentation image of the crop. This cell segmentation image is a binary image, segmenting pixels which belong to cells from pixels which belong to the background. This effectively removes all textural information from the crop and only the semantic content remains. To achieve this, a U-Net based encoder-decoder architecture and a Normalizing Flow model are used for the two steps respectively. The aim here is to generate new cells in a way that is as coherent as possible with the cell statistics within a crop, mainly focusing on the density and shape of cells.

The last stage, the **Image Inpainting**, utilizes a U-Net based GAN to generate visually convincing paintings, given a masked crop, the mask itself and the binary painting from the previous stage. See figure 12 for an overview of the complete Neural Inpainting pipeline.

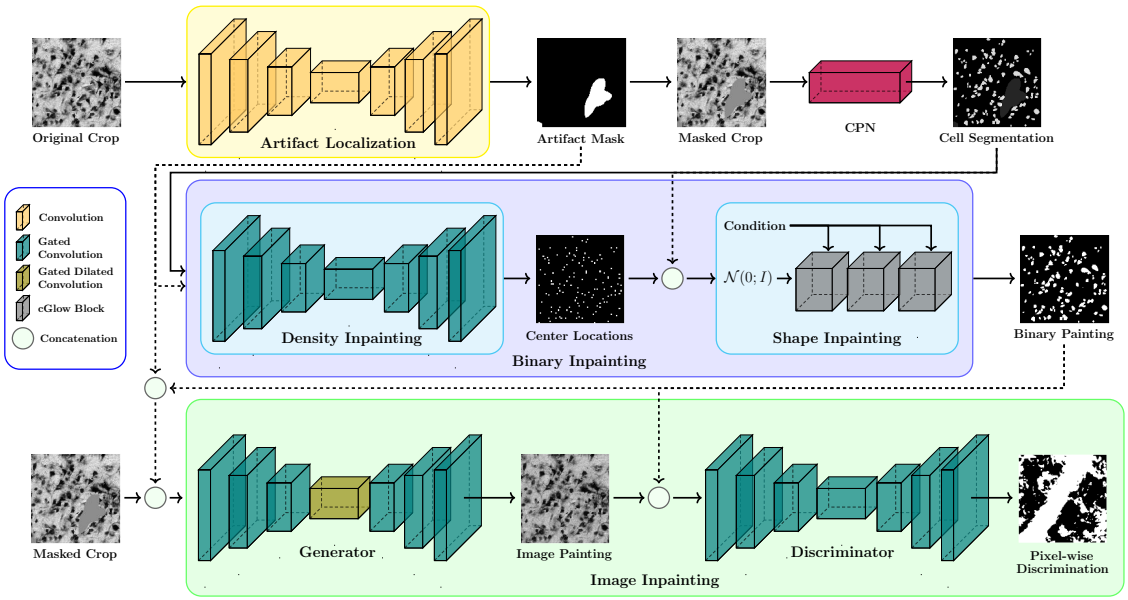


Figure 12: Overview of the Neural Inpainting pipeline. The three levels in the figure display the three stages of the Neural Inpainting pipeline, with the second stage, the Binary Inpainting, being split into two steps. All networks except for the cGlow model are based on the U-Net model [30]. For details on these modular U-Net variants, see section 2.1. All of them employ skip-connections, which are not depicted in this figure.

2.3.2 Artifact Localization and Reparation as a Two Step Process

The task of repairing histological artifacts is fundamentally split into two steps: The first step is to identify the corrupted areas. The second step is to modify them in a way that conforms to a set of criteria, in this case convincing visual quality and a meaningful recovery of select statistics, such as cell density and shapes within a crop of a scan. Using the framework of Deep Learning, these two tasks are approached as two completely separate tasks, involving different networks, training setups and even different data. Both parts are separately solved and then put together to form a complete model pipeline.

The first task, the **Artifact Localization**, is approached as a problem of supervised learning and with a single network. The input to the network is a raw crop of a brain scan and the network is supposed to return a binary mask of the same size, assigning a label to every pixel. Intuitively, the pixels are labelled as uncorrupted or corrupted tissue, in a technical sense however a positive label means that the pixel should be removed for following steps of the method. This is not necessarily the same since it makes sense to be generous with the labelling of the corrupted area, mostly because the exact border around it is not always clear, e.g. when an artifact includes blur. The data here is a set of manually annotated images containing various forms of histological artifacts. This task is explained in detail in section 2.4.

The second task, the **inpainting**, is a classic image inpainting task where the goal is to replace a missing part of an image with repaired content. The output of the Artifact

Localization network is a mask which is used to remove a portion of the image. This masked crop, together with the mask itself, serve as the inputs to the inpainting network. The network then has to generate new content for the masked pixels and this is composited together with the original pixels from the unmasked region of the image to a completed **painting**. During training however, real examples of corrupted brain tissue are not needed, because the corrupted regions would be cut out anyway. Instead, parts of scans of fully intact brain tissue can be cut out artificially. This has the crucial advantage that the real content of the masked section is known and can be used for feedback during the training process. Because of this, the dataset is not the manually annotated data used for training the Artifact Localization network. Instead, large scans of intact brain tissue are used. Since this is a generative task, the appropriate class of networks here are generative networks. The modular U-Net from section 2.1 was primarily developed for this task and is used, including all its features. In the remainder of section 2.3 the concept behind the approach to this task is presented. In sections 2.5 and 2.6 the execution of the approach is examined in detail.

2.3.3 Artifact Reparation in Cell Segmentation Images - Binary Inpainting

The naive approach to the inpainting task would be to take a generative model, e.g. a GAN, and set up the training with a masked image together with the mask as inputs to the network. The fundamental problem with this approach is to achieve the balance between the two target criteria, good visual quality and the faithful reproduction of cell statistics within a crop. The second criterion demands very specific properties of the outputs while focusing on only a small part of the image, the cells. The network needs to be able to precisely capture quantities like cell density and sizes from the masked input image and then produce new content accordingly. In other words, the network needs to be able to filter out the semantic content of the input image with respect to cells. While high visual quality can be achieved using state of the art GAN based inpainting models, it is difficult to optimize such a model with respect to statistical properties of a crop, such as segmented cell density or sizes. Experimental evidence for this is provided in the experiments sections 3.2-3.4 and 3.7.3 and discussed further in the second paragraph of section 4.1.

The high resolution scans of brain tissue show a very clear distinction between semantic content and textural content. The placement and shapes of cell bodies within a scan is the semantic content and contains very little textural information, while the background contains almost exclusively textural information. The generation of correct cell statistics for a painting requires a good semantic representation within the network, which can be described using far less information than is needed to describe the whole image. The generation of the background can then be performed based on the information on new cell bodies.

Given this observation, the idea is to separate the generation of semantic and textural content, i.e. cell bodies and background. Because natural images also tend to have the relationship of semantic content with foreground-background separation, this idea is not a new one. [38] proposed an inpainting model for natural images with three steps. First, a binary foreground map is created which separates background and foreground in the

image. Given this map, an incomplete contour mask is created, which effectively marks the border between background and foreground. It is incomplete because the missing part of the image usually overlaps at some point with this contour and thus part of the contour is missing. In this second step, the inpainting process is now applied to this incomplete contour map to obtain a completed contour. Using the completed contour as an additional input to the original incomplete image, a final inpainting network now generates the completed image.

Inspired by this method, the inpainting task in the Neural Inpainting pipeline is be split into two stages: **Binary Inpainting** and **Image Inpainting**. Instead of a foreground-background segmentation, a cell segmentation is used, which creates a binary mask of the same size as the input image based on which pixels belong to cells. Given this cell segmentation image, the Binary Inpainting stage is a completely separate inpainting task whose goal is to generate information on the position and shape of new cells. Effectively, all textural information was removed from the image and this stage is responsible for the satisfaction of the cell statistics criteria of the final image painting. The Image Inpainting stage then receives a completed cell segmentation image, the **binary painting**, as an additional input together with the masked image and mask itself, see figure 12. It can then focus on the generation of correct textural content while the semantic content is already determined from the binary painting. The output of this stage, composited with the pixels from outside the corrupted area from the original crop, is the final **image painting**. This is the end result of the Neural Inpainting pipeline, however if cell statistics without artifacts are the only interest, these should be directly extracted from the binary painting.

The primary goal of this idea is to give the model more control over the generation of information for new cell bodies, because such a specific feature is hard to single out when optimizing a GAN. Ideally, this also results in better cell statistics in the final image painting, however this requires the Image Inpainting model to include the binary painting faithfully into the image painting.

2.3.4 Model Choice for Binary Inpainting

Similarly to the approach without the Binary Inpainting step, the naive approach here would be to take a state of the art, GAN based, inpainting method. The sections 3.6.4 and 4.1.2 evaluate and discuss a GAN based approach to the Binary Inpainting task, which includes concepts from the U-Net based discriminator [35], foreground aware image inpainting [38] and gated convolutions [39]. Section A.1 contains the relevant model and training details. Experiments with this approach show severe problems with mode collapse, a common issue with GANs. This manifests in two ways: One issue is an oscillatory behavior of the generator where it cycles between outputs containing too few or too many generated cells, as well as between generated cells being too small or too large. For example at some point during training, the generator is in a state where it tends to output too few but too large generated cells, regardless of the input. Given some more training epochs and the outputs now contain too many generated cells in relation to the ground truth. Ultimately, the GAN failed to show a convergence behavior w.r.t the two simple statistics of cell amount and sizes. The second form of mode collapse happened at seemingly random points during training. Here, the generator wouldn't generate any

cells at all and simply output black (negative label) areas. Since black areas of varying sizes are part of the true data as well, the discriminator seemed to be unable to learn at which point the size of these areas becomes unreasonable.

This observation is related to a core reason why a GAN is inherently a bad approach here. Recall that the goal of the Binary Inpainting task is to generate new cells in such a way that specific statistics of a crop are maintained to a reasonable degree. This includes cell density and shapes. While a GAN does tend to pick out specific features from the input data, it doesn't stick to these and constantly changes its focus instead. The discriminator looks for features in the data which are incorrectly portrayed in the outputs of the generator and then make the final decision of real or fake based on this feature. Because the generator is updated based on the discriminator, it shifts its attention to this specific feature, causing the discriminator to start focusing on a different feature. This results in a constant back and forth between image features which is not desirable for this task. More details on this result and different approaches to remedy this problem is discussed in section 4.1.

Now that we have established that a GAN based approach doesn't perform as well as it does on natural images, the question is what kind of model to use and whether a different representation of the data is helpful.

A more specialized kind of networks, Graph Neural Networks (GNNs), have gained significantly more attention in the last years. Given the resemblance of cells in a crop to nodes in a graph, a method based on a GNN might offer some unique advantages. The main problem here is that each input to the network corresponds to a completely different graph. While there are models for dynamic graphs with good performance, like the TGN model [31], they are restricted to small changes in the graph from iteration to iteration, like the addition or deletion of a single node. This makes the class of GNNs, at least for now, unsuitable for this task.

Since inpainting is a generative task, the method has to be able to accurately model a very high dimensional, conditional distribution and be able to sample from it. The underlying true distribution of cell locations and shapes should be of a much lower dimension than the one in pixel space. Additionally, if we compare our cell segmentation images to natural images, they contain far less information in total, since we have already omitted all textural information. This suggests that the representation as a 2d-image is inefficient and can be replaced with a smaller data format without losing any relevant information. Such an alternative is the **Fourier format** for cell segmentation images from [36]. Given the contour of a segmented cell as a sequence of x- and y-coordinates, the concept of the Fourier series can be applied to describe these sequences as smooth functions. These functions are parameterized by N Fourier coefficients where N is a hyperparameter choice. This means that the shape and location of a segmented cell can be described by $4N+2$ coefficients, where appropriate values for N are between 2 and 6. A crop of size 256x256 usually contains less than 192 segmented cells, which means it can be described via the Fourier format with no more than 192x26 coefficients, for $N = 6$. This drastic reduction in dimensionality only comes with a minor loss of information. The contour representation via a Fourier series is not exact, with higher values for N resulting in better approximations. The Fourier format is used in all steps of the Binary Inpainting task. A detailed formulation and explanation of the Fourier format can be found in section

2.5.1.

Given this Fourier format, the first two coefficients for each segmented cell are the x- and y-coordinate of its center (center in a loosely defined way). This means that comparatively, the locations of all cells in an image requires far less information to be described than the shapes of all cells. Together with the fact that the cell density is an important statistic in a crop, the idea presents itself to solve the problem of generating new locations for cells as a separate, easier problem first. One crucial advantage here is that the extremely low dimensionality of 2 allows for very easy sampling. The core idea is to learn to complete a density estimate of a masked crop and then sample from the completed density. This can be done in a straight forward way using Kernel Density Estimation (KDE). This step of the Binary Inpainting stage, called **Density Inpainting** is examined in detail in section 2.5.2.

What remains is the generation of coefficients that describe the shape of a generated cell. Here, the information on where all generated cells are located can be used as an additional input to the network. This step, the **Shape Inpainting**, completes the Binary Inpainting and is described in section 2.5.3. Since the goal here is to fit a complex, high dimensional distribution as accurately as possible, Variational Autoencoders (VAEs) and Normalizing Flow (NF) models should be a better fit for the task than GANs are, with the NF models offering more extreme advantages and disadvantages. The representation of the data in the Fourier format helps to remedy some shortcomings of NF models. This ultimately makes them the better choice here, especially regarding good statistics in the painted sections, as the experiments in sections 3.3-3.5 show.

2.4 Artifact Localization

In this section, the Artifact Localization stage is presented in detail. Model and training details related to hyperparameter choices can be found in section A.1.

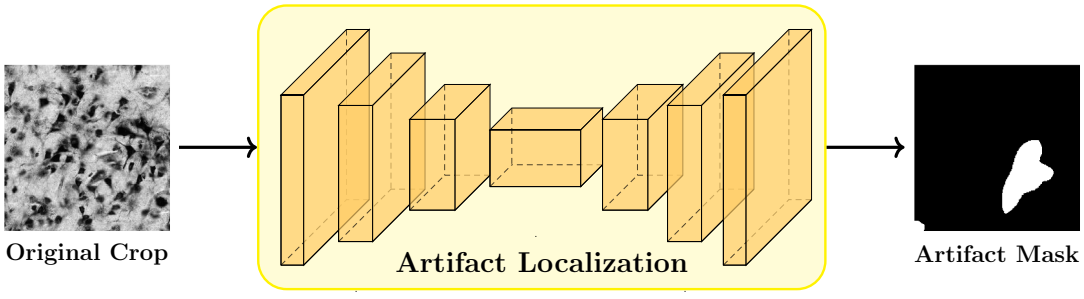


Figure 13: Overview of the Artifact Localization stage. This network is based on the U-Net model [30], hence it employs skip-connections between the encoder and decoder sections.

The goal of this stage is to localize histological artifacts in raw crops of brain scans. This is accomplished with a single network that takes a crop as input and returns a binary

mask of the same size as the crop, which segments pixels based on whether they belong to the corrupted area or not. The network is trained on manually annotated data, which makes this a supervised learning problem. As previously mentioned, the annotation of artifacts is deliberately done in a generous manner, to make sure the entire artifact is always covered, without exceptions. This is because the area of the crop that is labeled as uncorrupted, is copied over without changes to the final painting. It is better to cut out too much and have the network repair a larger section than to end up with small sections of corrupted tissue. By annotating in a generous manner consistently, the network learns to cut out the artifacts generously as well. Especially when the artifacts contain some form of blur, the exact border between corrupted and uncorrupted areas is not always clear.

The network here is a variation of the modular U-Net from section 2.1. It omits inpainting specific parts such as the gated convolutions and the dilated convolutions in the bottleneck, as well as features intended for GANs such as spectral normalization and LeakyReLU activation layers. Everything else is present as described in section 2.1. The width parameter of the model, which determines the number of output channels of the input block, is set to 32. For all non-1x1 convolutions, the kernel size is set to 3x3 and padding mode is 'reflect'. This results in 471,809 learnable parameters.

For training, the loss function is the Binary Cross Entropy with the model output as the input and the annotation as the target. This effectively treats each pixel as the probability of it belonging to the corrupted region. This objective showed better training behavior compared to L1 or L2 distance between input and target as the loss function.

2.5 Artifact Reparation in Cell Segmentation Images - Binary Inpainting

This section provides a detailed description of the Binary Inpainting stage. Model and training details as well as hyperparameter choices can be found in section A.1.

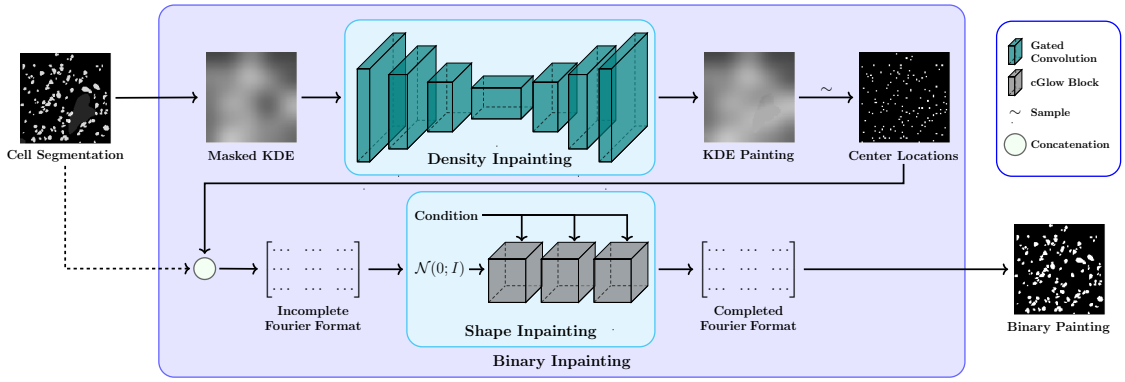


Figure 14: Overview of the Binary Inpainting stage. The Density Inpainting network is based on the U-Net model [30] with skip connections between the encoder and decoder sections. The Shape Inpainting network is the cGlow model [24], applied to a cell segmentation image in the Fourier format.

This stage acts as a supporting step to the Image Inpainting stage. It is responsible the generation of locations and shapes for the new cells ahead of the full Image Inpainting and provides this as an additional input. This implies that this stage is responsible for the achievement of good cell statistics in the final painting, while the Image Inpainting stage is responsible for the visual quality. To achieve this, a separate inpainting task is performed on a cell segmentation image. This is a binary image where each pixel is assigned a label based on whether it was part of a cell body in the input image. The segmentation is performed by the CPN network [36]. This effectively removes all textural information from the image and leaves only the semantic content with respect to cell bodies, see figure 15.

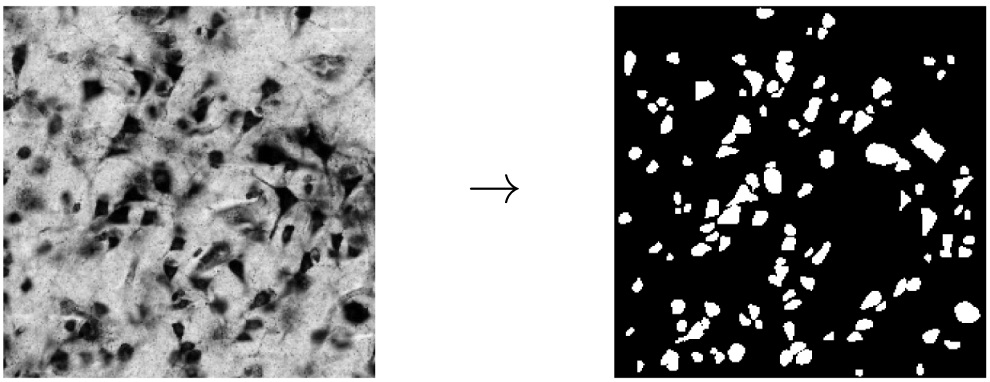


Figure 15: **Left:** A 384x384 crop of a brain scan with a spatial resolution of 1 micron per pixel. **Right:** The corresponding cell segmentation image from the CPN network [36].

A major deviation from the common image inpainting framework here is the use of a more specialized data format. Section 2.5.1 introduces the **Fourier format** from [36]. Using this format, the cell segmentation of a brain scan of arbitrary size can be described with a matrix of Fourier coefficients containing one vector of length 4·N+2 for each segmented cell. Appropriate values for N range from 2 to 6 which results in a drastic reduction in dimensionality compared to the regular 2d-image data format, see figure 16.

Because the location of a cell, i.e. the x- and y-coordinate of its center, is a highly relevant feature and is described with an extremely small amount of information, the generation of locations for new cells is split from the rest of the Binary Inpainting stage. Given the locations of all segmented cells in the segmentation image, a 2-dimensional Kernel Density Estimate (KDE) is computed. The KDE of the masked segmentation image serves as input and the KDE of the ground truth segmentation image serves as target to a regular image inpainting task. Note that this step already determines the number of generated cells for final painting. This **Density Inpainting** step is described in detail in section 2.5.2.

Given the locations generated by the Density Inpainting step, together with the locations of segmented cells from the uncorrupted area, the second step is to generate shape information for the new cells. This is called **Shape Inpainting**. Here, a Normalizing Flow model tries to model the conditional, high-dimensional distribution of segmented cell shapes as accurately as possible. The segmentation images for the input and target for the network are presented in the Fourier format as a matrix of Fourier coefficients. Details

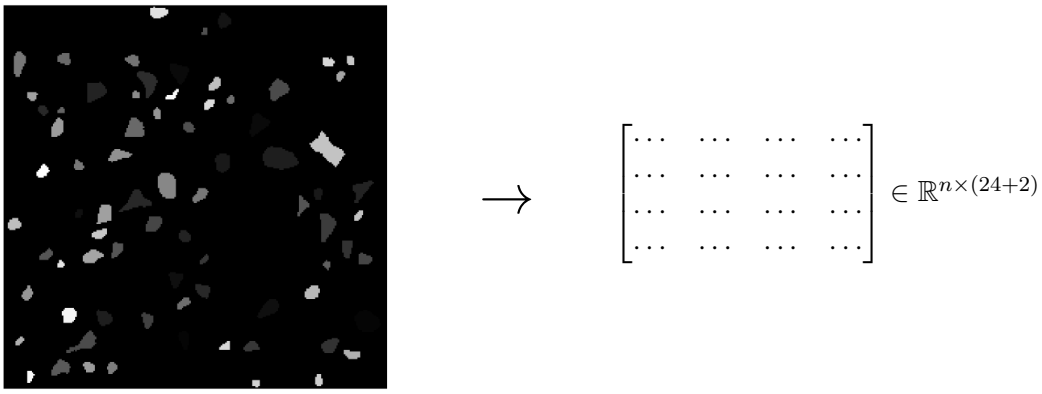
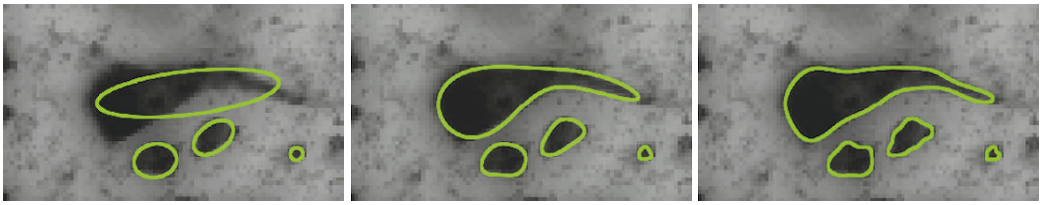


Figure 16: **Left:** A labeled cell segmentation image from the CPN network [36]. **Right:** The representation of this image via the Fourier format. The matrix contains n rows of Fourier coefficients, here 24+2 for each cell, where 24 is a hyperparameter. 2 coefficients are the x- and y-coordinates of the cell center, the other 24 describe the shape.

on this step can be found in section 2.5.3.

2.5.1 Cell Contour Description via Fourier Series

A different way of describing segmented cell shapes in a parameterized way for neural networks was proposed in [36]. Originally, [22] described a way of using the Fourier sine and cosine transformations to approximate closed contours. The resulting method can be used to elegantly describe the contour of segmented cells in a segmentation image. The resulting representation in matrix form describes all cells of such a segmentation image with far fewer parameters. The following is a formal definition of the Fourier format for closed contours and an explanation of the computation of the Fourier coefficients for this use case.



(a) $N = 1$, 6d vector (b) $N = 3$, 14d vector (c) $N = 8$, 34d vector

Figure 17: Figure 2 and caption from [36]: Contour representation with different settings of the order hyperparameter N. It defines the vector size of the descriptor that is given by $4N + 2$. The higher the order, the more detail is preserved. The 2d contour coordinates are sampled from the descriptor space with Eq. 1. Even small settings of N yield good approximations of odd and non-convex shapes, in this case human neuronal cells, including a curved apical dendrite.

A 2-dimensional Fourier contour of degree N is described by the two functions:

$$x_N(t) = \frac{a_0}{2} + \sum_{n=1}^{N} \left(a_n \cos(n2\pi t) + b_n \sin(n2\pi t)\right)$$
$$y_N(t) = \frac{c_0}{2} + \sum_{n=1}^{N} \left(c_n \cos(n2\pi t) + d_n \sin(n2\pi t)\right)$$

with $t \in [0, 1]$. At a fixed value $t = \hat{t}$, the pair $(x_N(\hat{t}), y_N(\hat{t}))$ is a point on the contour. We have $x_N(0) = x_N(1)$ and $y_N(0) = y_N(1)$ and the functions are continuous, hence the resulting Fourier contour is always closed.

The functions $x_N(t)$ and $y_N(t)$ are parameterized by the Fourier coefficients $(a_0, ..., a_N), (b_1, ..., b_N), (c_0, ..., c_N), (d_1, ..., d_N)$. Given an initial contour $(f_x(t), f_y(t))_{t\in[0,1]}$, we can compute these coefficients with the formulas:

$$a_n = 2\int_0^1 f_x(t) \cos(n2\pi t)dt$$
$$b_n = 2\int_0^1 f_x(t) \sin(n2\pi t)dt$$
$$c_n = 2\int_0^1 f_y(t) \cos(n2\pi t)dt$$
$$d_n = 2\int_0^1 f_y(t) \sin(n2\pi t)dt$$

The hyperparameter N, called **order** in [36], controls how good the approximation of $x_N(t)$ to $f_x(t)$ and $y_N(t)$ to $f_y(t)$ is. Higher orders introduce higher frequencies in the exponential (or trigonometric) terms of the Fourier series and allow for modeling of more complex shapes. With large enough finite N, any smooth function $f(t)$ can be exactly written as a Fourier series. If we allow infinite N, we can express non-differentiable and even non-continuous functions as a Fourier series, as long as the Dirichlet conditions are satisfied.

Some additional explanation and intuition on why the above formulas hold true and why this is a very elegant way to parameterize an almost[1] arbitrary closed contour can be found in section A.2 in the appendix.

In practice, the coefficients a_n, b_n, c_n and d_n for $n = 1, ..., N$ can be computed via an approximation of the above integrals from [22]: For $p = 1, ..., K$, let x_p be a point on a

[1]The existence of a Fourier series is characterized by the Dirichlet conditions. Since we are dealing with continuous and closed contours, these are always satisfied and investigating these conditions is not useful here.

contour f with $f(t_p) = x_p$ and $t_p \in [0, 1]$. Then

$$a_n = \frac{1}{2N^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} (\cos(2N\pi t_p) - \cos(2N\pi t_{p-1}))$$

$$b_n = \frac{1}{2N^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} (\sin(2N\pi t_p) - \sin(2N\pi t_{p-1}))$$

$$c_n = \frac{1}{2N^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p} (\cos(2N\pi t_p) - \cos(2N\pi t_{p-1}))$$

$$d_n = \frac{1}{2N^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p} (\sin(2N\pi t_p) - \sin(2N\pi t_{p-1}))$$

The offset-coefficients a_0 and c_0 are computed via

$$a_0 = \sum_{p=1}^{K} \frac{\Delta x_p}{2\Delta t_p} (t_p^2 - t_{p-1}^2) + \xi_p (t_p - t_{p-1})$$

$$c_0 = \sum_{p=1}^{K} \frac{\Delta y_p}{2\Delta t_p} (t_p^2 - t_{p-1}^2) + \delta_p (t_p - t_{p-1})$$

with

$$\xi_p = \sum_{j=1}^{p-1} \Delta x_j - \frac{\Delta x_p}{\Delta t_p} \sum_{j=1}^{p-1} \Delta t_j$$

$$\delta_p = \sum_{j=1}^{p-1} \Delta y_j - \frac{\Delta y_p}{\Delta t_p} \sum_{j=1}^{p-1} \Delta t_j$$

and

$$\xi_1 = \delta_1 = 0$$

The CPN network [36] that is used in this thesis for cell segmentation uses these formulas for the computation of Fourier coefficients.

2.5.2 Density Inpainting

The location of segmenteds cell in a segmentation image is a highly relevant feature that can be described perfectly with very little information: The x- and y-coordinate of its center. Various definitions of the cell center work here. Conveniently, the Fourier coefficients a_0 and b_0 describe exactly this. The **Density Inpainting** step is performed purely on the locations of the centers of the segmented cells. The goal is to sample new cell locations in a way that conforms to the density of segmented cells from the ground truth segmentation image.

In its naive setup, this problem exhibits an extreme case of the superposition problem. The ground truth provides feedback on where the segmented cell center locations are, but there is no way of inferring these correctly from the masked segmentation. Because of the 2-dimensional from of the data here, we can modify it to reduce the impact of the superposition problem. A 2d binary image that only contains center locations of segmented cells can be interpreted as a sample from a cell density for this crop. We can approximate this density by computing a 2-dimensional Kernel Density Estimate (KDE) on this crop. Doing this on such binary center images of the ground truth segmentation and masked segmentation provides target and input for a regular image inpainting problem. Unlike other image inpainting tasks however, this one does not require a generative model since the stochasticity has been removed already. The network learns directly on the density. By not requiring the network to generate exact locations of new cells, the superposition problem is alleviated and the network is be able to infer the target image from the input image with a high degree of accuracy.

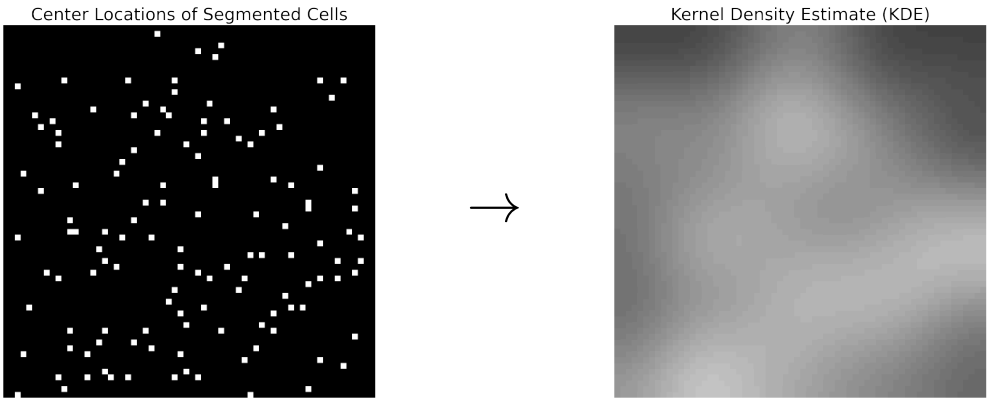


Figure 18: **Left:** A binary image of center locations of segmented cells. **Right:** The Kernel Density Estimate of the left image using Gaussian filters with bandwidth 30/4. Both images have been reduced in size by a factor of 4 along each dimension compared to the original segmentation image.

The KDE is performed by applying a multidimensional Gaussian filter to the binary image of center locations of the segmented cells. Ideally, in the center locations image and in the KDE image, each pixel value can be interpreted as the probability of this pixel being the center of a segmented cell. Applying a raw Gaussian filter means we lose some of this probability mass around the image borders though. Performing the KDE on a much larger image crop would remove this issue, because neighboring segmented cells from slightly outside the border of our smaller crop add some probability mass to the border region of the smaller crop. To approximate this effect, a reflect mode is used in the computation of the KDE. Probability mass that would land outside the crop is reflected back into it. The result is an equal amount of probability mass in the center locations image and the KDE image. Crucially, a pixel-wise sampling that sets a pixel to 1 with the probability of its previous value, results in the correct amount of generated cells in expectation. The standard deviation of the Gaussian kernel is a hyperparameter choice.

Before the KDE is applied to an image of center locations, the image is downscaled along

both axes by a factor of 4. This greatly reduces the computational cost of the training process for this step with minimal drawbacks. Technically, this introduces two limitations: It limits the number of total centers an image can contain. Because the number of pixels in a regular crop far exceeds the common number of cells in it by a much larger factor than 16, this is not an issue. The second limitation is that centers can't be placed right next to each other, but have at minimum 3 pixels in between them. Since this is physically not very feasible in the first place, this is also rather negligible.

The employed model here is another variation of the modular U-Net architecture from section 2.1. Because the task is effectively an inpainting task, the network features gated convolutions. The dilated convolutions in the bottleneck have been omitted because of the image size reduction of every KDE image. For example, a crop of size 256x256 results in an 8x8 feature dimension in the bottleneck. The GAN specific features, spectral normalization and LeakyReLU activation, are omitted as well. The width parameter is set to 8 and all non-1x1 convolutions have kernel size 3x3 with standard zero-padding. This results in a modest 229,113 learnable parameters.

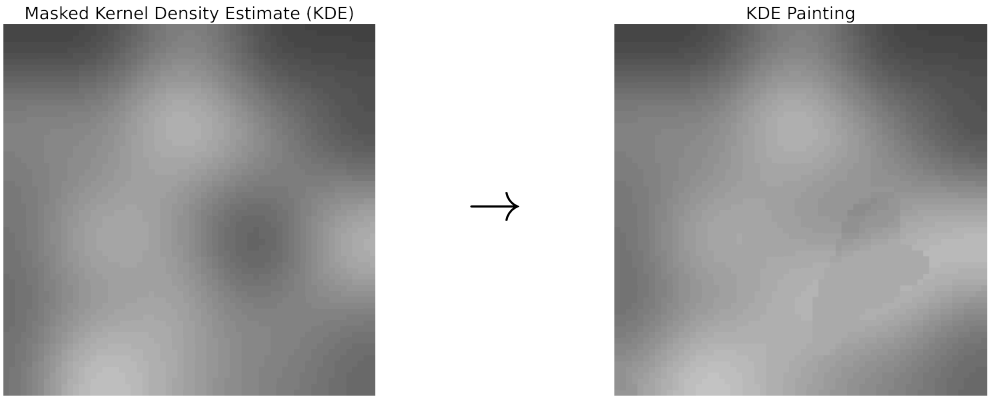


Figure 19: **Left:** The Kernel Density Estimate of the masked image of center locations of segmented cells from figure 18. Note the darker region to the right of the image center. **Right:** The KDE painting, which is a composition of the original pixels from outside the masked area and the network output for the pixels inside the masked area.

After the network has completed the masked KDE image to feasible KDE painting, the last step is to convert this back into an image of center locations. Since we already know the center locations from all segmented cells from outside the masked area, we only need to focus on the KDE inside the masked area. Because of the setup here, we can interpret a pixel value in the KDE as the probability of this pixel being the center of a segmented cell. We can iterate a simple sampling process through each pixel in the masked area, where it is set to 1 with the probability of its previous value. In expectation, the resulting number of generated cells here is equal to the sum over all pixels from the masked area. Assuming the network did a perfect job, this is even equal to the number of segmented cells from the ground truth, still in expectation. The simplicity of this process is what enabled the split into Density Inpainting and Shape Inpainting in the first place.

2.5.3 Shape Inpainting

Now that we have locations for the generated cells, we need to generate shapes for each one. The resulting completed Fourier matrix can then be converted into a binary painting, the result of the Binary Inpainting stage. This task exhibits the superposition problem similarly to the Density Inpainting step. The difference is that the distribution of shapes of the segmented cells is more complex and higher dimensional than the distribution of locations of segmented cells. In the Fourier format representation, all coefficients but two, a_0 and c_0, describe the shape of a segmented cell. Recall that the aim of the Binary Inpainting stage is to model the cell statistics of a crop as accurately as possible. Conceptually, this makes **Normalizing Flow** (NF) models an attractive approach here. Because of their tractability of the exact log-likelihood and subsequent optimization by directly maximizing the log-likelihood of data samples, they are the model class of choice for this step. Their main drawback is strong limitations in architectural design, because every layer needs to be invertible in an efficient way. This poses a challenge for very high dimensional data, such as natural images, where large and specialized architectures achieve the best results. The use of the Fourier format, as an already very elegant representation, fundamentally enables the use of NF models here.

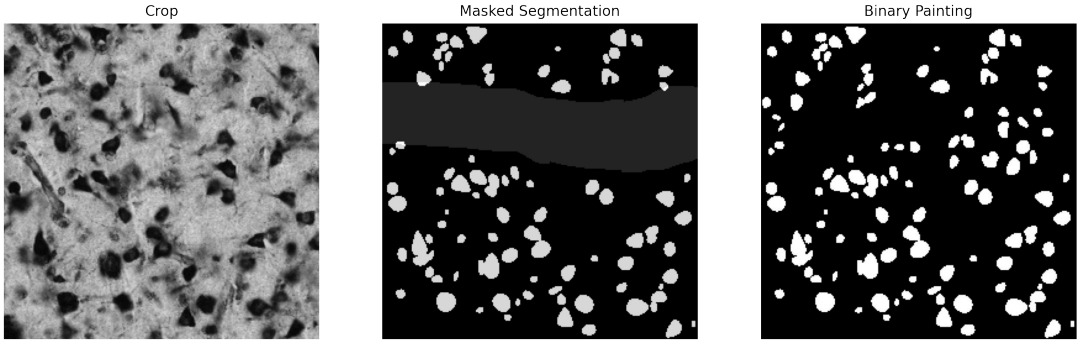


Figure 20: An inference examples of the Binary Inpainting stage, with crop dimensions are 256x256. The masked region is highlighted for legibility purposes.

A small challenge here is the application of NF models to an inpainting task. The difference to classical applications of NF models is that inpainting is a conditional task, where the uncorrupted region of a crop serves as a condition. Luckily, maximum likelihood methods naturally extend to conditional maximum likelihood methods by conditioning all considered distributions. This applies to VAEs as well as NF models. [24] proposed a conditional version of the popular Glow method [17] that can also be used for inpainting. This method, called **cGlow** was employed for the Shape Inpainting step with no architectural modifications. It was only slightly modified to work with the Fourier format and new hyperparameter tuning was performed (see section A.1).

The cGlow method uses the three basic layers from the Glow method: Actnorm, 1x1 convolutions and affine coupling layers, but in a conditional version respectively. To achieve this, a conditioning network is used, which is a basic convolutional network that does not need to be invertible. For the conditional actnorm and conditional 1x1 convolutions,

the conditioning network is used to generate the parameters for these layers. Hence, the learnable parameters are the ones of the conditioning network and not the ones of the main layers. For the conditional affine coupling, the conditioning network extracts features from the condition x and these are used as an additional input to the regular network of the coupling layer.

Another small challenge here is the variable size of the Fourier matrix. The amount of segmented cells in a crop determines the amount of rows in the Fourier matrix, resulting in a potentially different input size for every example. Only very specific architectures, like Transformers, can deal with variable input sizes, which severely limits the architectural choice here, especially for NF models. The solution is to keep the architecture and instead modify the Fourier format to have a constant size. This is accomplished by determining a feasible maximum number of segmented cells for a crop of a given size, and then pad the Fourier matrix with zero-rows up to this maximum size. The choice of this maximum length is a hyperparameter that presents a tradeoff between training sample efficiency (throwing away too many samples) and model performance (introducing too much padding). In the event that a crop exceeds the maximum number of segmented cells, it is removed from the current batch.

2.6 Artifact Reparation in Histological Sections - Image Inpainting

This is the third and last stage of the Neural Inpainting pipeline. While the Artifact Localization provides a binary mask, segmenting corrupted and uncorrupted pixels, and the Binary Inpainting provides a binary painting containing information on the locations and shapes of newly generated cells, this stage performs the actual inpainting on a masked version of the original crop. The binary mask and binary painting from the previous two stages are used as additional inputs to the network here. Because the semantic (w.r.t. cell bodies) part of the inpainting is already done in the Binary Inpainting stage, this stage needs to mostly generate textural information and aims to achieve a visually convincing result. Wile this stage leads to a deterioration of the accuracy of the cell statistics, compared to the binary painting, this is not a big concern because these cell statistics can be directly extracted from the binary painting when they are of interest. See figure 21 for an overview of this stage.

This task is very similar to common inpainting tasks on natural images and is approached accordingly. The modular U-Net (section 2.1) was primarily designed for this task and all its features are employed here. Exact details on the network architecture, including hyperparameter settings, can be found in section A.1. A large number of random examples are displayed in section A.3.

Pixel-wise Weighted Reconstruction Loss One difference to a usual GAN setup for natural images is the implementation of the reconstruction loss. Usually, some form of distance measure in pixel space, e.g. L1-distance, is used as a loss term in addition to the adversarial loss. This doesn't work here because of the severity of the superposition problem with this data. With a vanilla L1 loss the inner regions of the generated content becomes too blurry or doesn't form texture at all. When applying a weight smaller than 1 to improve the balance between adversarial and reconstruction loss, the regularization

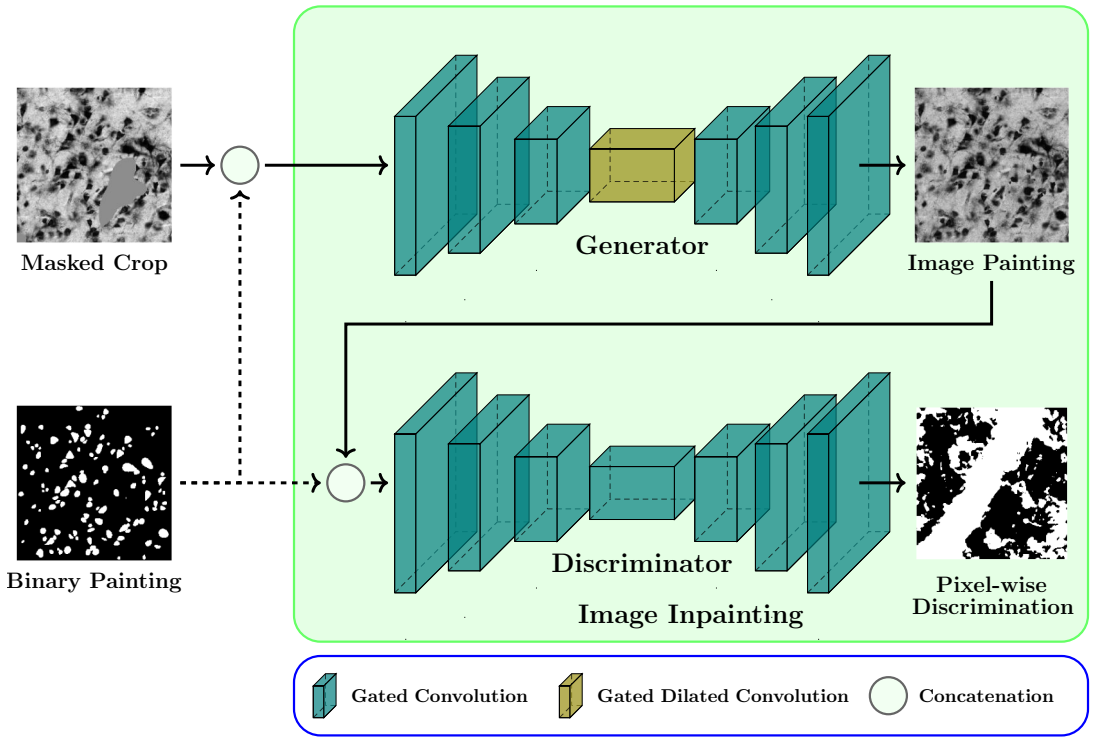


Figure 21: Overview of the Image Inpainting stage. Both networks are based on the U-Net model [30]. They employ skip-connections, which are not depicted in this figure. During inference, only the generator is used.

effect of the reconstruction loss disappears together with its problems. Instead, a pixel-wise weighted reconstruction loss is used. The importance of each pixel is weighted based on its distance to the nearest uncorrupted pixel. The effect is that the border regions of the corrupted area receive feedback from the reconstruction loss while the inner regions don't. Note that this is applicable to arbitrary mask shapes, even when they are disjoint. For this method, only 0 and a fixed $w \in [0, 1]$ are viable weights, with all pixels within a threshold distance from the nearest uncorrupted pixel having the weight w. This threshhold and w are hyperparameters that are fixed during training runs and provides control of the amount of regularization via the reconstruction loss. This formulation of reconstruction loss provides meaningful feedback to pixels where this is feasible but avoids the pitfalls of the superposition problem with L1-loss. The goal is a smooth and convincing transition between original and generated pixels in the final painting.

Conditioning on the Binary Painting To ensure that the generator correctly includes the information from the binary painting into the image painting, it is given to both networks as a conditional input. This way, the discriminator has an easy way to find out where the generated area is by checking the existence of cells in the correct locations. The discriminator learns to do so early on in training and the generator subsequently learns

to avoid this form of discrimination.

Noise Injection In the original GAN setup on a non-conditional task, the input to the generator is a noise vector. Since inpainting is a conditional task where the generator receives the incomplete image as input, it is unclear how to include noise in this setup, if at all. There are various approaches to this, like adding noise to the data, providing an additional input channel that contains random noise or even a layer-wise injection of noise [16]. In this method, the generator always received an additional channel of noise drawn from a standard-normal distribution with the same dimension as the input image. The Style-GAN layer-wise noise injection didn't show any meaningful improvements in the generator outputs. This might be due to the different kinds of features present in histological images compared to natural images.

3 Experiments

This section provides a range of experiments to test and compare the performance of the models from the Neural Inpainting pipeline with respect to the research objective. It is structured by experiments rather than models and is sectioned into four parts: Section 3.1 contains experiments and evaluations for the Artifact Localization stage. Sections 3.2 to 3.5 provide quantitative experiments, while section 3.6 shows qualitative evaluations for the Binary Inpainting stage. Lastly, section 3.7 provides a qualitative evaluation of the Image Inpainting stage. The discussion in section 4.1 afterwards interprets these results by focussing on individual models and their pair-wise comparisons.

Besides the Artifact Localization (AL), Density Inpainting (DI), Shape Inpainting (SI) and Image Inpainting (II) models from section 2, there are three additional models featured in this section. They provide comparison and experimental motivation for the use of the four models in the Neural Inpainting pipeline. Model and training details can be found in the appendix in section A.1.

Binary Inpainting GAN As described in section 2.3.4, the naive approach to the Binary Inpainting stage would be a GAN based Inpainting model. Conceptually, this is very similar to the Image Inpainting model, but the segmentation images are a very different type of data than the full brain scans. This GAN model is based on the modular U-Net from section 2.1, the architecture and hyperparameters are detailed in the appendix in section A.1.

Shape Inpainting with VAEAC Alternative to the cGlow model introduced in section 2.5.3, a conditional VAE model for Inpainting was implemented and modified to work on the Fourier Format. It is based on the VAEAC model [13] with minor modifications and was employed in the exact same setup as the cGlow model. This includes the masking process, the Fourier format, the crop dimension, the amount of padding in the Fourier format and the evaluation process. Note that this is a fully convolutional architecture which is a suboptimal inductive bias for the Fourier format, since information in this data representation is not distributed as locally as in 2d-images. The use of the Fourier format still proved to be an improvement though, compared to using the regular cell segmentation images.

The VAEAC model is a conditional variant of the regular VAE model [20]. The VAE problem setting naturally extends to a conditional one, because it is a maximum likelihood approach. The variational lower bound (ELBO) in the conditional setting can be derived in the same way, by conditioning all involved distributions on a condition y:

$$L(\theta, \phi, \psi, x, y) = -D_{KL}[q_\phi(z|x,y)||p_\psi(z|y)) + \mathbb{E}_{q_\phi(z|x,y)} \log p_\theta(x|z,y) \leq \log p_{\theta,\psi}(x|y)$$

The prior distribution $p_\psi(z|y)$ is conditioned on y and now modeled by a neural network with parameters ψ, the prior network. Hence, we now have a prior network parameterizing $p_\psi(z|y)$, a proposal network parameterizing $q_\phi(z|x,y)$ and a generative network parameterizing $p_\theta(x|z,y)$. The two encoders, the prior and proposal networks, always receive the condition as input, the proposal network additionally receives the target. The

KL-Divergence term in the objective is between the two encoders, next to the reconstruction loss of the output of the decoder. The encoders' distributions are Gaussian, which enables the use of the reparameterization trick to compute the KL divergence analytically.

There are two figures for each evaluation statistic of the VAEAC model, because there is an important choice in the final sampling process of the model. The decoder parameterizes a multivariate Gaussian distribution with diagonal covariance matrix and the default setting is to return the mean of this distribution as the final output of the model. Alternatively, one can sample from the distribution and thus increase the variety in model outputs. Because this has a very strong effect on the evaluation statistics and the sample quality, both settings are examined.

Image Inpainting without Binary Inpainting This model is very similar in architecture to the Image Inpainting model from section 2.6. It was tuned and retrained without additional inputs from the Binary Inpainting stage. Section A.1 contains information on the model and training details. As this one and the regular Image Inpainting model don't appear in the same experiments, both are referred to as 'II model' at various points in this chapter.

For this model, we can evaluate the same statistical properties as for the Density and Shape Inpainting steps by computing cell segmentation images from the paintings. Given such a cell segmentation image, we can remove all segmented cells outside the corrupted area and compute statistics such as segmented cell count, size and eccentricity. Importantly, the fact that we compare two unprocessed cell segmentation images from the CPN network here might favor this model. The cGlow and VAEAC models process the segmentation images, instead of real crops, and thus have a harder job of matching the ground truth segmentation images. These models can produce shapes that the CPN network wouldn't, because it has never seen image paintings like those from the Image Inpainting model, only real crops.

Experiments with this model motivate the choice to develop the Binary Inpainting stage, as described in section 2.3.3.

3.1 Artifact Localization

The Artifact Localization stage is a supervised learning task on manually annotated data. It is the least complex and easiest to optimize network of the Neural Inpainting pipeline. Because the goal here is to generate 2-dimensional binary masks that segment the corrupted from uncorrupted pixels, there are no quantitative evaluation metrics apart from the label coverage or accuracy. The validation and evaluation of the quality of the networks was mostly performed using the test loss and visual quality as indicators for performance.

Quantitative Evaluation The final model achieved an average segmentation accuracy of 97.8%. Note that an error of 1% corresponds to 1475 wrong pixels for the crop dimension 384. See figures 23 and 24 for some example images together with their segmentation accuracy. Everything below 90% can be seen as a bad inference example just by judging

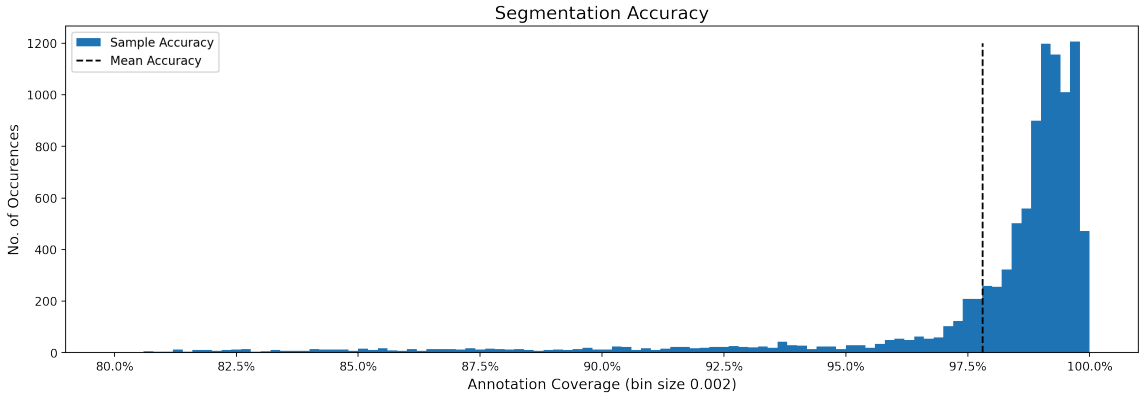


Figure 22: Artifact Localization segmentation accuracy evaluation on a run of 10000 examples with crop dimension 384. The segmentation accuracy is the percentage of equal pixels in model output and annotation. 100% means the model's output was identical to the annotation, not just that the masked area was fully covered.

visually. Around 5.4% of the 10000 examples are below 90% accuracy and around 10.7% are below 95% accuracy.

Qualitative Evaluation Judging from visual quality, the Artifact Localization stage works as expected, although the results heavily depend on the type of artifact. There is a large variety of possible artifacts and some are segmented better than others. This strongly suggests that the size and diversity of the dataset is the performance bottleneck here, more on that in section 4.1. Figure 23 shows an example of an artifact that is segmented almost perfectly and figure 24 shows an example of an artifact that the network has problems segmenting.

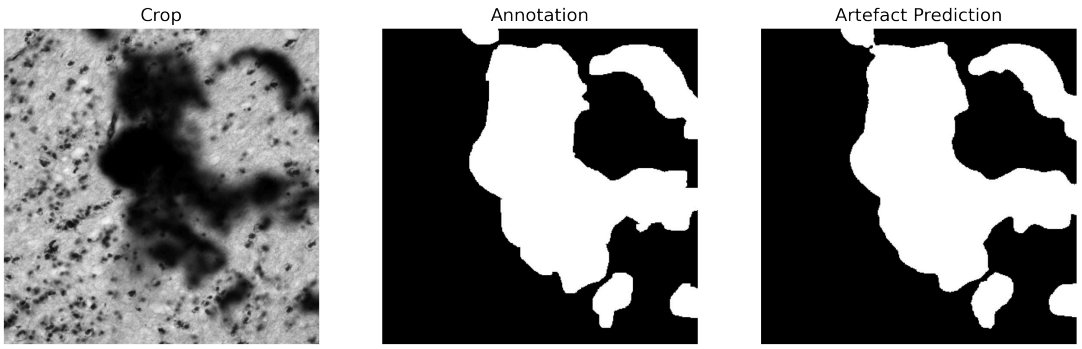


Figure 23: Artifact Localization sample from the artifact dataset. The sample is a 384x384 crop from a test scan that was not part of the training set. The large artifact is segmented with a high accuracy of 98.0%.

Typically, artifacts that cause a blur of their surrounding region tend to be problematic,

because the blurred area still contains texturally correct content which might be segmented as uncorrupted. A larger collection of examples can be found in section A.3.

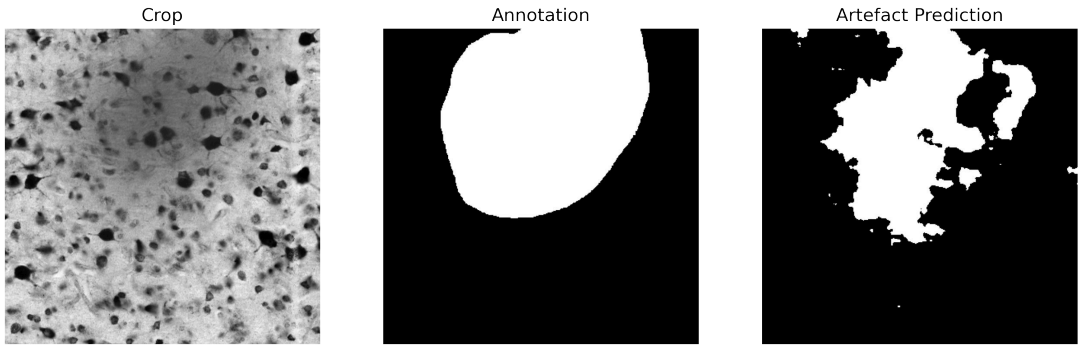


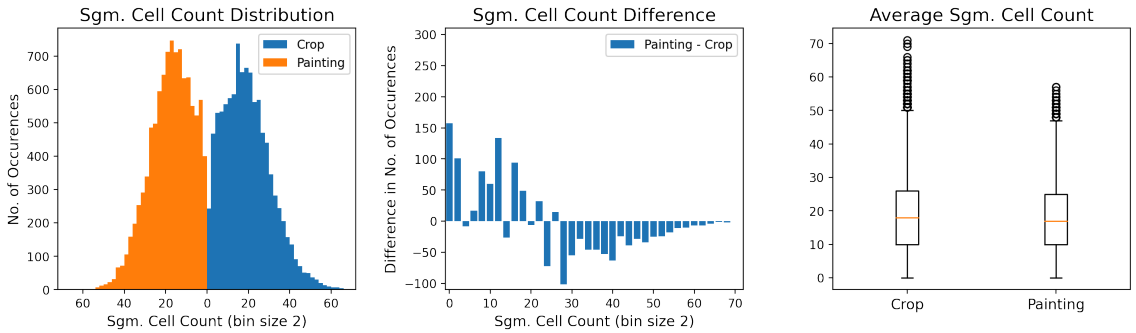
Figure 24: Artifact Localization sample from the artifact dataset. The sample is a 384x384 from a test scan that was not part of the training set. The blurry nature of the artifact causes problems for the network and results in a segmentation accuracy of only 86.9%.

3.2 Cell Count in Repaired Cell Segmentation Images

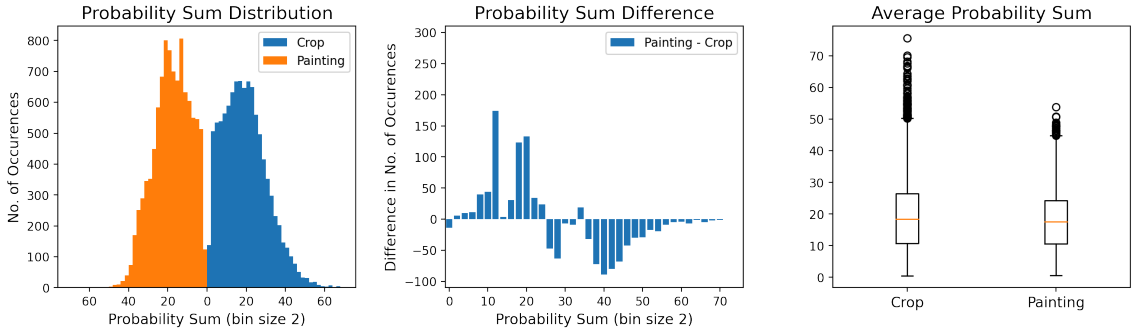
The first evaluation statistic is a comparison between the segmented cell count of ground truth crops and paintings. Each evaluation run includes 10000 examples with crop dimension 256 from scans from the intact dataset that were not seen during training. All plots and values below consider the corrupted areas only, unless specified otherwise. Two models are tested here, the Density Inpainting model from section 2.5.2 and the Image Inpainting model without binary paintings as additional input. There is an alternative evaluation scenario for the DI model, where the sum of probability mass from the KDE inpainting is used, instead of the cell count after sampling. Because of the 1-to-1 correspondence of probability mass in the KDE images and cells in the segmentation images, this is gives insight into the exact amount of predicted cells, without the variance of the sampling process.

Each of the three scenarios has a separate evaluation run, but figures 25 and 26 display results from the same run for (a), (b) and (c) respectively.

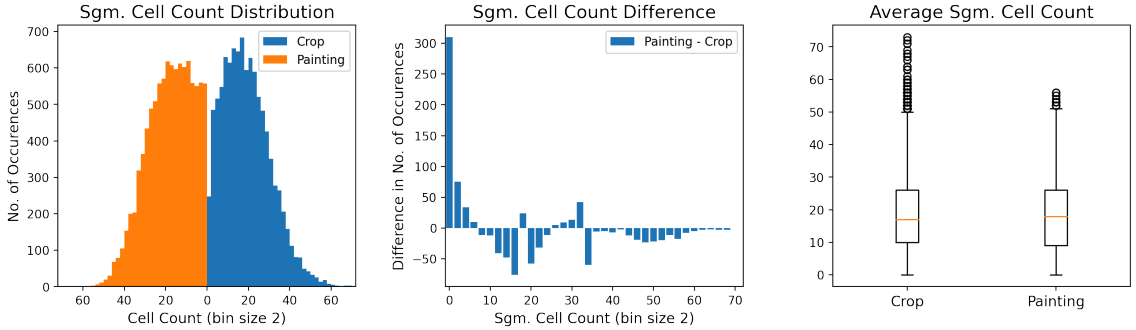
There are two experiments here, one looking at the overall distribution of segmented cell counts without comparing ground truth and paintings directly to each other. For the first experiment, the results from all examples of the run are accumulated separately and plotted as histograms, box plots and plots that show the differences between corresponding bins in the histograms. These results are shown in figure 25.



(a) Full Density Inpainting model



(b) KDE image probability sum



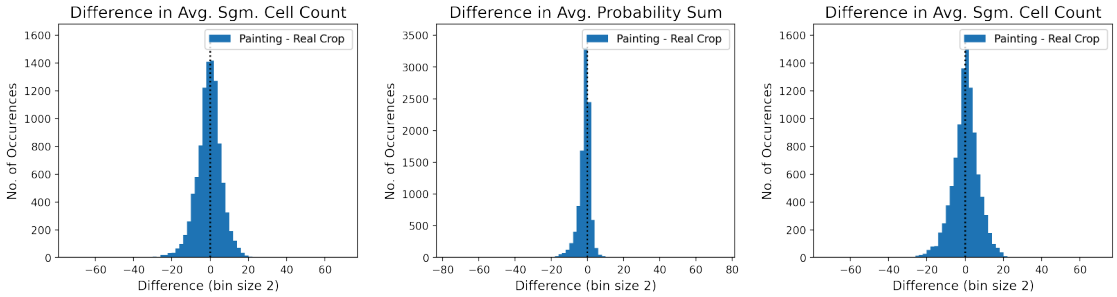
(c) Image Inpainting without Binary Inpainting

Figure 25: Results for the first segmented cell count experiment. The segmented cell counts were accumulated for ground truth crops and paintings separately over the 10000 examples. **(a)** Results for the full DI step as described in 2.5.2. **(b)** Results for the DI step when omitting the sampling at the end and replacing the segmented cell count with the sum over the KDE image. **(c)** Results for the II model that was trained without auxiliary inputs from the Binary Inpainting stage.

Both models are able to learn the segmented cell count distribution reasonably well and show similar results here. They are not able to capture the full variance of the distribution from the ground truth data, which is clearly visible in the box plots. The mean however is very accurate. Note that the segmented cell count for the DI model also contains noise from the sampling process at the end of the Density Inpainting step. The network's out-

put is a KDE matrix with a certain amount of probability mass. Figures 25 and 26 (b) shows the error when comparing the probability mass between the KDE of the original crop and the network's output.

For the second experiment, the segmented cell count error (CCE) is computed. For each pair of ground truth crop and painting, the absolute value of the difference in segmented cell count is computed. The CCE is the average of these. The histograms in figure 26 are computed on the values before averaging. This way, we can see how much of the CCE can be attributed to the sampling noise of the DI step and how accurate the network's predictions are. The added variance from the sampling process leads to less accurate results and an overall higher CCE. The II model performs comparably to the full Denisty Inpainting model here. A low mean in combination with a low standard deviation is desirable here.



(a) DI (b) DI KDE (c) II w/o BI

Figure 26: Results for the second segmented cell count experiment. The absolute value of the difference in segmented cell count between ground truth crop and painting is taken for every example. **(a)** Results for the full DI step as described in 2.5.2. **(b)** Results for the DI step when omitting the sampling at the end and replacing the segmented cell count with the sum over the KDE image. **(c)** Results for the II model that was trained without auxiliary inputs from the Binary Inpainting stage.

Table 1 contains the results from both experiments for easy comparisons. The slight bias towards smaller amounts of segmented cells for the DI model is a consistent result and not a training artifact.

	CCE Mean	CCE Std	Avg. Count Painting	Avg. Count Real
DI	4.92 (3.93%)	4.35	18.0	19.0
DI KDE	**2.57 (2.06%)**	2.94	17.9	19.5
II w/o BI	5.15 (4.12%)	4.55	**18.5**	18.7

Table 1: Results for the segmented cell count error (CCE) experiment. All computations consider the corrupted area only. The CCE is the average over all examples of the absolute value of the difference in segmented cell count between corresponding ground truth crops and paintings. The percentage for the CCE mean is with respect to the total amount of segmented cells in the ground truth crop, including the uncorrupted area.

While the DI model has a lower CCE than the II model, it also is slightly more biased when looking at the average segmented cell count over all examples.

3.3 Cell Size in Repaired Cell Segmentation Images

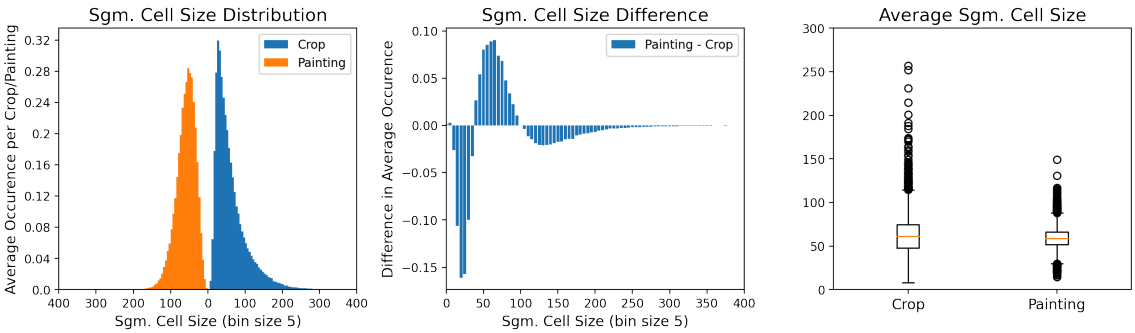
The second evaluation statistic is the size of generated cells compared to the ground truth segmentation images. This gives a good indication for the accuracy of generated cell shape information. Each evaluation run includes 10000 examples again, with crop dimension 256 from scans from the intact dataset that were not seen during training. All plots and values in this section consider the corrupted areas only, unless specified otherwise. Besides the cGlow model for Shape Inpainting on Fourier matrices, from section 2.5.3, the VAEAC model and the Image Inpainting without Binary Inpainting model were included in the experiments, the latter is referred to as the II model.

As mentioned at the beginning of the experiments section, there are two different settings for the generation of samples for the VAEAC model. The decoder parameterizes a multivariate Gaussian distribution with diagonal covariance matrix. One can either take the mean of this distribution as a result or sample from it. These are referred to as mean-return and sample-return settings.

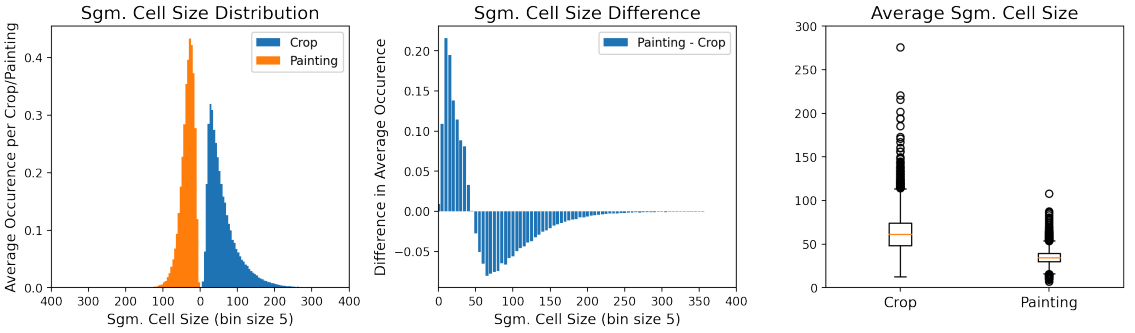
The segmented cell sizes are not averaged per crop or painting, because that would hide the variety of cell sizes within a segmentation image or painting. As a result, the histograms contain a different amount of data points, as not all models generate the same amount of cells on average. The important characteristic here is the shape of the distribution and the balance between small, medium and large cells, not the height of the bars.

Similar to the segmented cell count section, there are two ways to evaluate the runs here. For the first one, the segmented cell sizes are accumulated separately for ground truth crops and paintings and then plotted as histograms, box plots and plots that show the differences between corresponding bins in the histograms. These results are shown in figure 27.

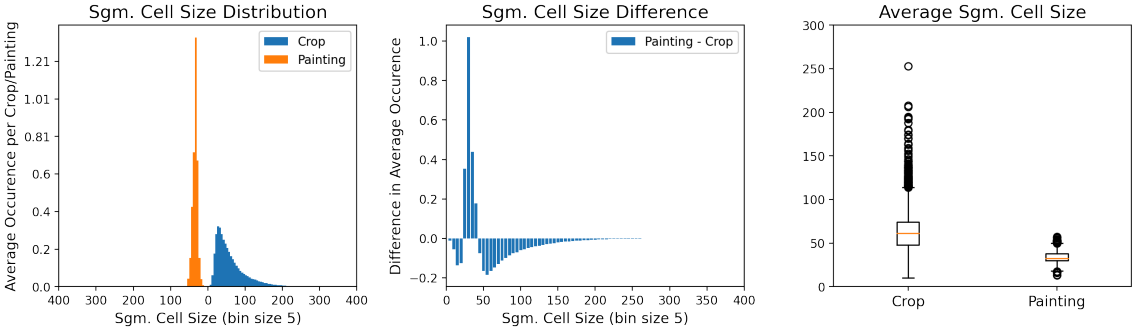
The learned distribution for the cGlow model shows very similar characteristics to the segmented cell count from the Density Inpainting step, in that it is unable to capture the full variance of the data distribution and has a slight bias towards smaller generated cells. The 'Segmented Cell Size Difference' plot in figure 27 (a) illustrates this well. Segmented cells with sizes around 60 pixels appear more frequently in paintings, while generated cell sizes under 50 and above 100 pixels appear more frequently in the ground truth images. The bias here is a consistent result and not a training artifact. Section 4.1 discusses possible reasons for this.



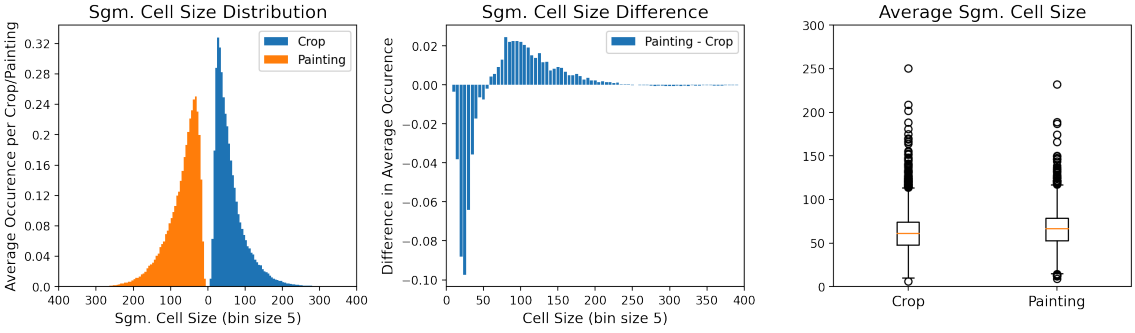
(a) cGlow model for Shape Inpainting on Fourier matrices



(b) VAEAC model for Shape Inpainting on Fourier matrices with sample-return setting



(c) VAEAC model for Shape Inpainting on Fourier matrices with mean-return setting



(d) Image Inpainting without Binary Inpainting

Figure 27: Results for the segmented cell size experiment. **(a)** Results for the cGlow model for Shape Inpainting, as described in 2.5.3. **(b)** Results for the VAEAC model with the sample-return setting. **(c)** Results for the same VAEAC model, but with the mean-return setting. **(d)** Results for the Image Inpainting model without auxiliary inputs from the Binary Inpainting stage.

The VAEAC model shows severe model collapse in the evaluation plots for the mean-return setting. The model focuses on a very narrow range of cell sizes and the resulting distribution does not have the same mean or mode as the ground truth one. The sample-return settings fares better here, but still displays a strong lack of variance and a strong bias towards smaller cell sizes. Both settings perform significantly worse than the cGlow model.

The Image Inpainting model without Binary Inpainting displays the best variance out of all models and is the only model that generates a reasonable amount of larger cells. It does have a bias towards larger cell sizes though, not as strong as the VAEAC, but significantly stronger than the cGlow model. See table 2 for a comparison.

For the second experiment, the segmented cell sizes are averaged and then compared between corresponding ground truth crops and paintings, still considering the corrupted areas only. Taking the absolute value of the difference in average segmented cell size for an example results in the segmented cell size error (CSE). A low mean in combination with a low standard deviation is desirable here.

	CSE Mean	CSE Std	Avg. Size Painting	Avg. Size Real
cGlow	18.3	15.4	**59.7**	61.8
VAEAC mean	28.7	19.0	34.2	61.6
VAEAC sample	28.1	18.7	35.3	61.7
II w/o BI	**16.9**	14.6	69.9	61.5

Table 2: Results for the segmented cell size error (CSE) experiment. All computations consider the corrupted area only. The CSE is the average over all examples of the absolute value of the difference in average segmented cell size between corresponding ground truth crops and paintings.

The cGlow and II models clearly outperform the VAEAC when it comes to segmented cell sizes. They display lower CSE values, with the II model narrowly beating the cGlow model, and have less bias when looking at the averaged segmented cell size over all examples (without averaging over the crops and paintings individually). The cGlow model is the only one without a significant bias here. It loses out on the CSE however, because it's not properly adapting to examples with large size average.

3.4 Cell Eccentricity in Repaired Cell Segmentation Images

The third evaluation statistic is the eccentricity of cell contours in the segmentation images. The eccentricity of an ellipse is the ratio of the focal distance (distance between focal points) over the major axis length. The focal points of an ellipse are the pair of points whose sum of distances to any point of the ellipse is constant. The eccentricity is in the interval $[0, 1)$ with the value 0 corresponding to a circle, because a circle's focal points are both in its center. It effectively measures how 'stretched out' the ellipse is. The eccentricity of a segmented cell can be defined as the eccentricity of the ellipse with the same second-moments as the closed cell contour.

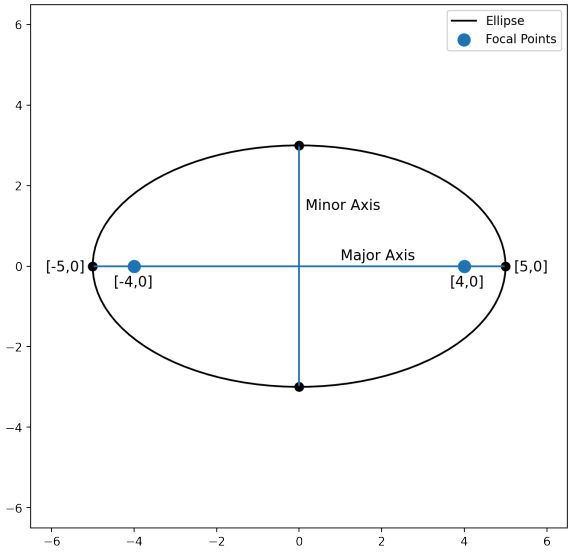
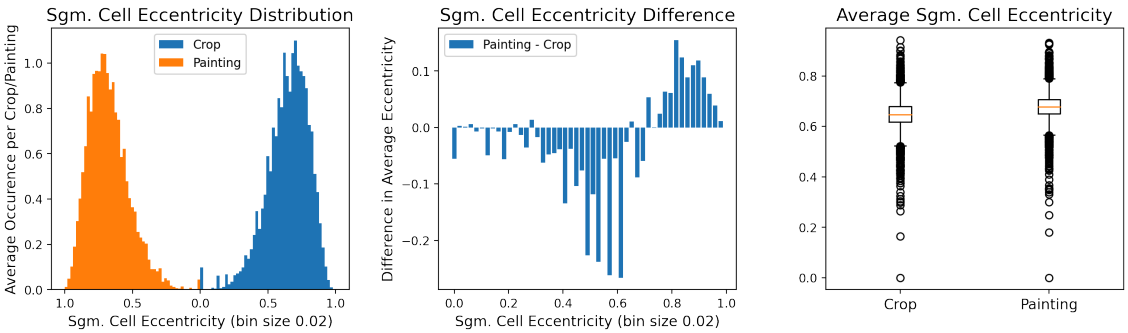


Figure 28: An ellipse in standard position with the major axis along the x-axis. The eccentricity is the ratio of the focal distance over the major axis length, here $8/10 = 0.8$.

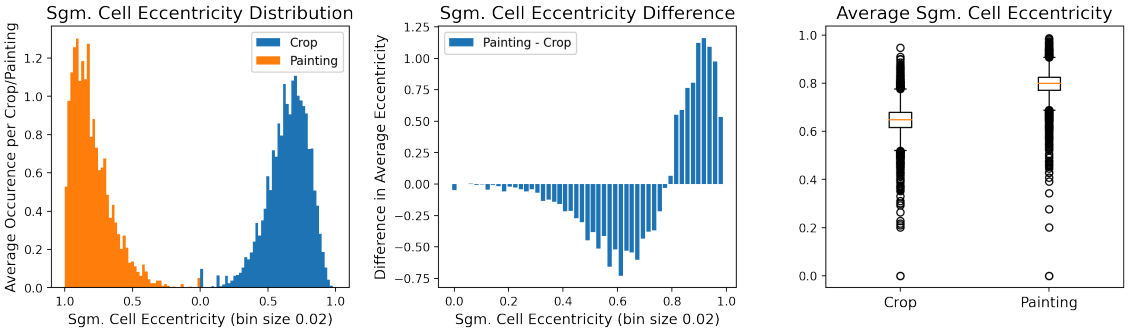
This can now be evaluated analogously to the segmented cell size. In the same way that every segmented cell has a size, it has an eccentricity and we can compute the same evaluation statistics from this. There are again two experiments, with the same procedure as in the segmented cell size section. Figure 29 shows results for the first experiment, where the segmented cell eccentricities are accumulated for ground truth crops and paintings separately and then plotted as histograms, box plots and plots that show the difference between corresponding bins in the histograms. All runs include 10000 examples with crop dimension 256 and consider the corrupted areas only. They were performed on data from the intact dataset that was not seen during training.

The small peaks at 0 come from very small segmented cells that are displayed as a square due to a lack of resolution. For the cGlow model, the overall distribution here is learned well and shows no obvious flaws like a strong bias or less variance compared to the ground truth distribution. This evaluation is especially useful to detect training collapse in the model, because a collapsed model tends to only output similar shapes with very little variation.

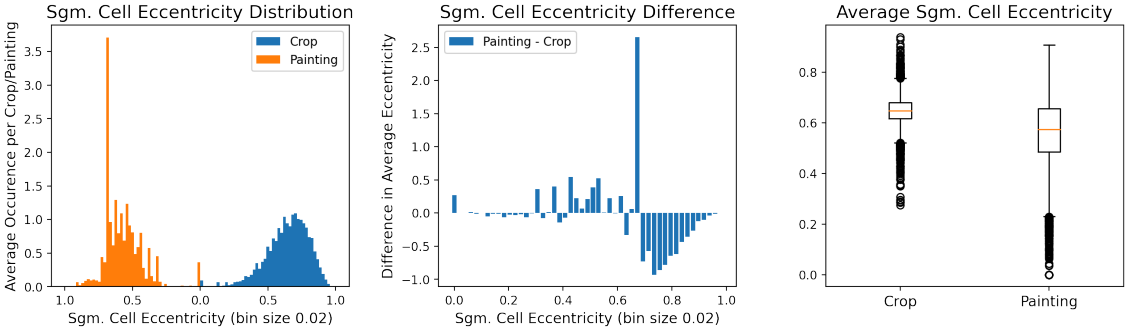
The VAEAC model performs overall very similar to the cell size statistic. The mean-return settings results in a severely collapsed model that generated cells with one eccentricity value most of the time and the sample-return setting displays a strong bias compared to the ground truth data. The plot for the sample-return setting hints at the bad visual sample quality in the qualitative evaluation. The model tends to produce sstetched out, thin cell contours, which results in an eccentricity distribution skewed towards higher values. The segmented cell eccentricity is learned accurately.



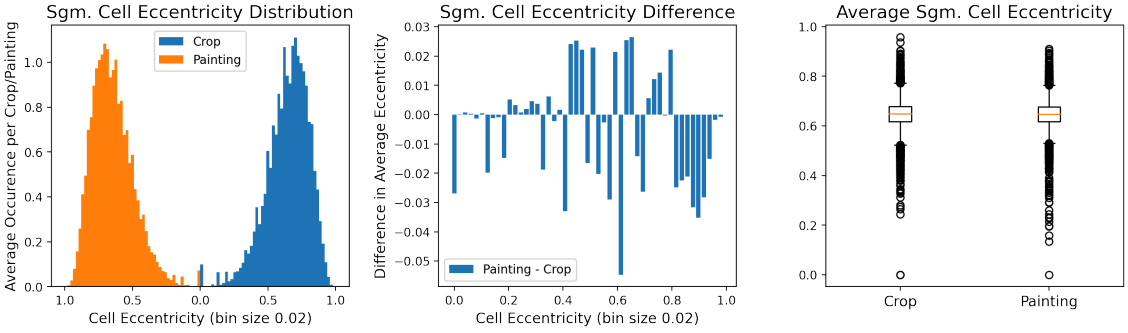
(a) cGlow model for Shape Inpainting on Fourier matrices



(b) VAEAC model for Shape Inpainting on Fourier matrices with sample-return setting



(c) VAEAC model for Shape Inpainting on Fourier matrices with mean-return setting



(d) Image Inpainting without Binary Inpainting

Figure 29: Results for the segmented cell eccentricity experiment. **(a)** Results for the cGlow model for Shape Inpainting, as described in 2.5.3. **(b)** Results for the VAEAC model with the sample-return setting. **(c)** Results for the same VAEAC model, but with the mean-return setting. **(d)** Results for the Image Inpainting model without auxiliary inputs from the Binary Inpainting stage.

The learned distribution for the II model here looks very similar to the ground truth one, with no deficiencies when it comes to bias or variance. The average segmented cell eccentricity of the paintings was very close to the ground truth average. While this and the cGlow model both perform very well, the II model actually edges out the cGlow model in this experiment.

The second experiment computes the same data as the first one, but averages the eccentricities over a crop or painting and compares these averages by taking the absolute value of the difference between them. Averaging the results of this over all examples yields the segmented cell eccentricity error (CEE). A low mean in combination with a low standard deviation is desirable here.

	CEE Mean	CEE Std	Avg. Ecc. Painting	Avg. Ecc. Real
cGlow	0.06	0.05	0.68	0.65
VAEAC mean	0.12	0.1	0.57	0.65
VAEAC sample	0.15	0.07	0.80	0.65
II w/o BI	**0.05**	0.05	**0.64**	0.65

Table 3: Results for the segmented cell eccentricity error (CEE) experiment. All computations consider the corrupted area only. The CEE is the average for all examples of the absolute value of the difference in average eccentricity cell size between corresponding ground truth crops and paintings.

Again, the VAEAC model is clearly outperformed by both, the cGlow and II model. While the cGlow and II models are close in performance, the II model shows the best results in the second experiment as well, with the least bias in average eccentricity and the lowest CEE.

3.5 PCA on Fourier Matrices of Cell Contours

Completing the quantitative evaluation for segmentation images is a comparison of the Fourier matrices via PCA. PCA is applied to the ground truth Fourier matrix and the network's output and the first two principal dimensions are plotted. Similar to the above, only cells from the corrupted area of a crop are being considered here, for both ground truth and network output. While this doesn't have as clear of an interpretation as segmented cell sizes or eccentricities, it is a good way of comparing the variance in the parameters themselves and see if the network is able to model dependencies within the data.

The 10000 examples were stacked into two large matrices with shape (n_examples, $4 \cdot$order). The PCA was computed on the ground truth matrix only and then the dimensionality reduction was applied to both matrices. Still, only segmented cells from the corrupted areas are taken into account here.

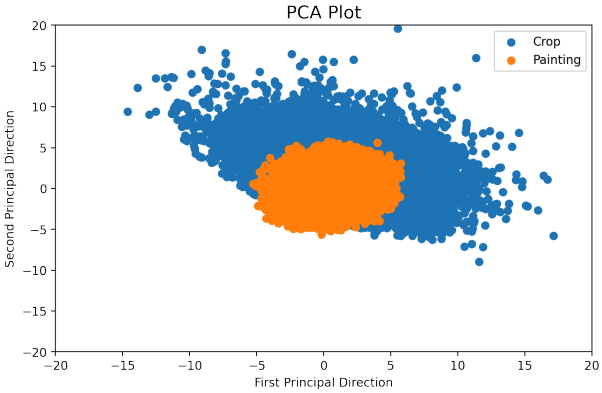
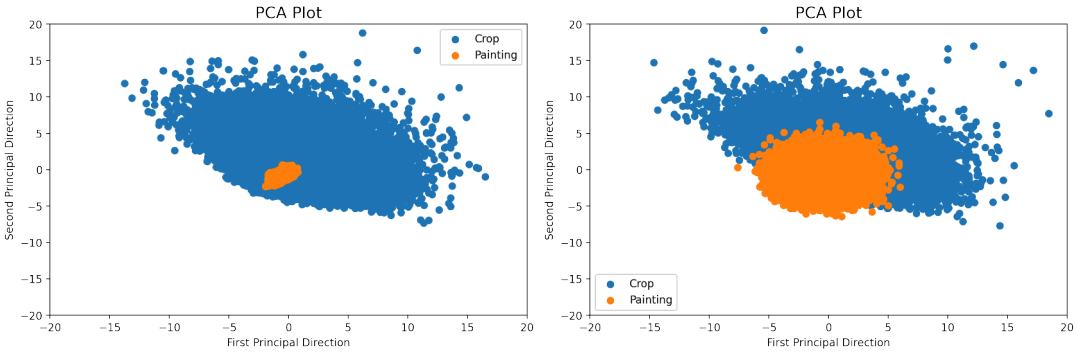


Figure 30: A plot of the first two principal components of the Fourier matrices of ground truth crops and paintings from the cGlow model. The PCA was computed on the ground truth matrices and the dimensionality reduction applied to both matrices.

This plot clearly shows what was partly seen in the previous quantitative evaluations, namely that the model is unable to capture the full variance in the data. Also, while the data points from the ground truth data are asymmetrically distributed, they are of course perfectly 0-centered. The component-wise mean of the model's data points is slightly off-center with [0.32,0.01].

Figure 31 shows the PCA results for the VAEAC model. Once again, the mean-return settings displays a severe lack of variance, while the sample-return settings fares much better in this regard. From the PCA plot alone, the VAEAC model performs comparably to the cGlow model in this experiment.



(a) VAEAC mean-return setting (b) VAEAC sample-return setting

Figure 31: 2 Plots of the first two principal components of the Fourier matrices of ground truth crops and paintings from the VAEAC model. The PCA was computed on the ground truth matrices and the dimensionality reduction applied to both matrices.

3.6 Qualitative Evaluation of Binary Inpainting Models

3.6.1 Density Inpainting

The Density Inpainting step is responsible for the generation of information on the amount and the placement of new cells, given the cell segmentation image of a masked crop. The quality of the generated content can be judged by two criteria: The overall amount of new cells, which should be coherent with the uncorrupted part of the image, and the placement of these cells relative to the uncorrupted part of the image, i.e. conforming to high and low segmented cell density areas within the crop. The former was evaluated in the segmented cell count section and the latter is addressed in the qualitative evaluation.

Regardless of the performance of the Density Inpainting network, it is important to mention that without this step, the Shape Inpainting via Normalizing Flow models wouldn't be possible in its current form. This is because the Shape Inpainting model is unable to sample the number of generated cells. It can only assign Fourier parameters, including a location, to a given number of new cells, making the split into a two-step process necessary. Of course, there are ways to approach this problem differently, but that would result in a complete redesign of the Binary Inpainting stage.

The second desired criterion for the Density Inpainting model, the accordance with high and low density regions within a crop, proved difficult to evaluate in a quantitative manner. This criterion is also less important than the overall segmented cell count, because very few crops have area with considerably different segmented cell densities due to the rather small crop dimension. When training the Neural Inpainting pipeline on larger crop dimensions, e.g. 512 or higher, this aspect of the Density Inpainting results becomes more significant.

That being said, the results here a clearly suboptimal, as the masked area in the paintings (composition of network output and ground truth KDE) are always very easy to spot. One reason for this is that the unmasked areas in the ground truth KDE and the masked KDE that the network receives as input are different. To simulate the inference process, the input images for training have to be masked first, before the KDE is computed. Hence, the global influence of the segmented cells from the masked area is present in the ground truth KDE, but not in the masked KDE. The training of the Density Inpainting model is thus not a classical inpainting task, where a missing part of the input image needs to reconstructed. Instead, the network needs to infer what the masked area from the ground truth image looked like, given the masked KDE which is different in the entirety of the image, not just to masked area.

Another shortcoming here is that the superposition problem is still present, although much less severe than without the KDE format. The network does tend towards more uniform solutions and is sometimes averse to continuing visible structures from the uncorrupted part of the masked KDE, see figure 33 for an example.

Figure 32: A training example from the Density Inpainting model. The masked KDE is the result of computing a KDE on an image of segmented cell locations, where the cells within the masked area have already been removed. The KDE painting is a composition of pixels from the model's output and the ground truth image, not the masked KDE. This painting is for evaluation purposes only, since only the masked area has to be sampled from during inference.

Figure 33: A training example from the Density Inpainting model displaying a rather uniform section in the corrupted area of the painting.

3.6.2 Shape Inpainting

The second step of the Binary Inpainting stage, the Shape Inpainting, generates information on the shape of the new cells, given their amount and center locations. It does so by filling in missing values in the Fourier format of the cell segmentation image. The results here can be evaluated in many different ways, since a distribution of shapes is difficult to quantify. The quantitative evaluation segment provided three different statistics to evaluate the performance of the Shape Inpainting step: A ground truth to painting comparison with segmented cell sizes, segmented cell eccentricities and a PCA of the Fourier format. A qualitative evaluation by visually inspecting examples still provides additional insight here. It can expose the occurrence of unwanted patterns and check the coherence of cell shapes within a crop. Crops from different regions of the brain contain different kinds of cells. The network needs to detect this and generate new shape information accordingly.

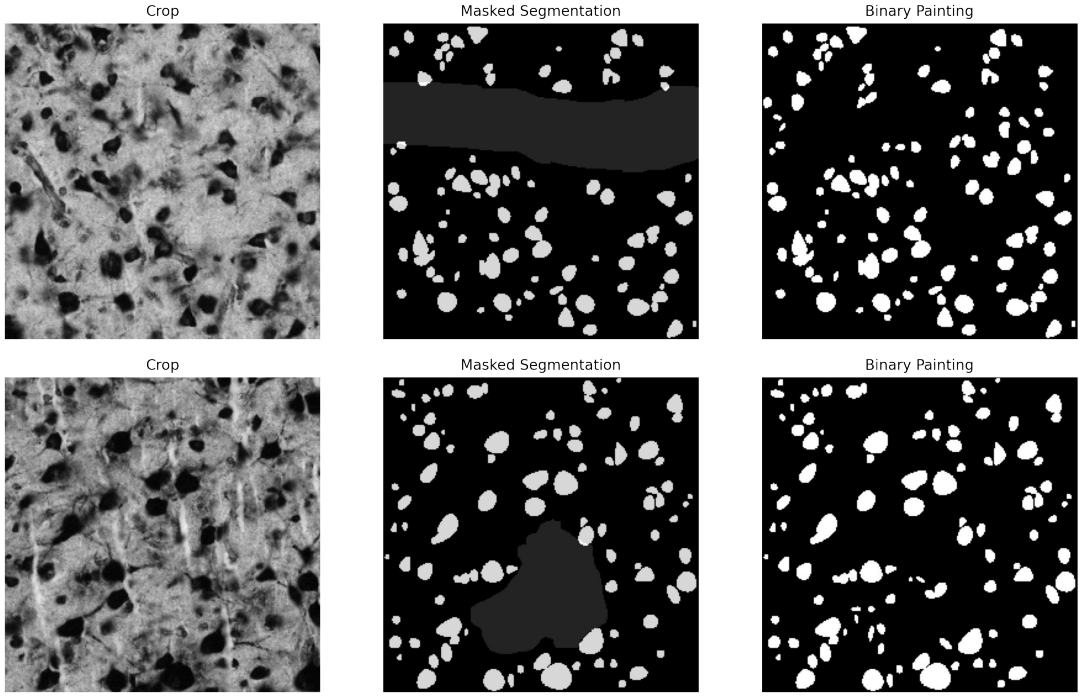


Figure 34: Two inference examples of the cGlow Shape Inpainting model. The crop dimensions are 256x256 and they were not seen during training. The masked region is highlighted for legibility purposes.

Because the formation of feasible shapes for generated cells is a complex task, the visual evaluation of results during and after training can provide a lot of information on the network's performance and learning process. This includes undesired features like the occurrence of unwanted patterns, a lack of variety in cell shapes and sizes, as well as cell contours 'folding' in on themselves. This folding artifact is always present in the initial phases of training and should disappear quickly. Higher settings of the order hyperparameter displayed problems with this folding behavior. Additionally, the size and eccentricity statistics don't full describe the shape of a segmented cell. The network could perform well with respect to these statistics, but still generate unrealistic cell shapes.

Figure 34 shows some inference examples from the final cGlow model. Cell shapes overall look comparable in quality to the ground truth data. There are no obvious issues with the individual cells. The second example shows the main flaw of the model, the inability to condition its output properly to the cell shapes in the uncorrupted area. Here, the masked segmentation image contains a lot of large, circular cell shapes, but the network only generated smaller shapes. Usually, this is not a problem, it only becomes apparent for select examples. Similarly, for crops with unique cell shapes, such as pyramid cells.

Early on in the learning process and for higher settings of the order parameter (≥ 4), the samples show a folding artifact, where a single contour crosses over itself at least once. This is visible in figure 35 and the main reason why the final model employs order 2.

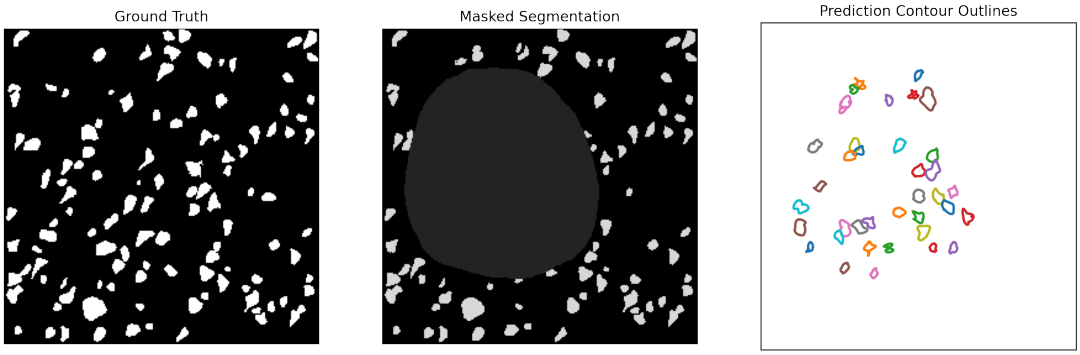


Figure 35: cGlow Shape Inpainting model inference example with folding artifact. The right shows the contours of generated cells for order parameter 6. The small red contour in the top right and the small green contour towards the bottom are folded in onto themselves, something that can't happen in the cell segmentation image of a real crop.

3.6.3 Shape Inpainting with VAEAC

The visual sample quality was the main problem for the VAEAC model. As seen in the quantitative evaluation segment, the mean-return setting resulted in a collapse of the model, while the sample-return settings displayed better performance. The model collapse of the mean-return setting is clearly visible when looking at inference examples, as shown in figure 36. Notably, almost all generated cells in the corrupted area of the Binary Painting are circular and of similar size.

The sample-return setting generates a wider variety of generated cells, but regularly produces odd generated cell shapes that are not comparable to anything in the ground truth data. Especially long, stretched out shapes appear far too often. As a result, the cGlow model clearly outperforms the VAEAC model regarding visual sample quality.

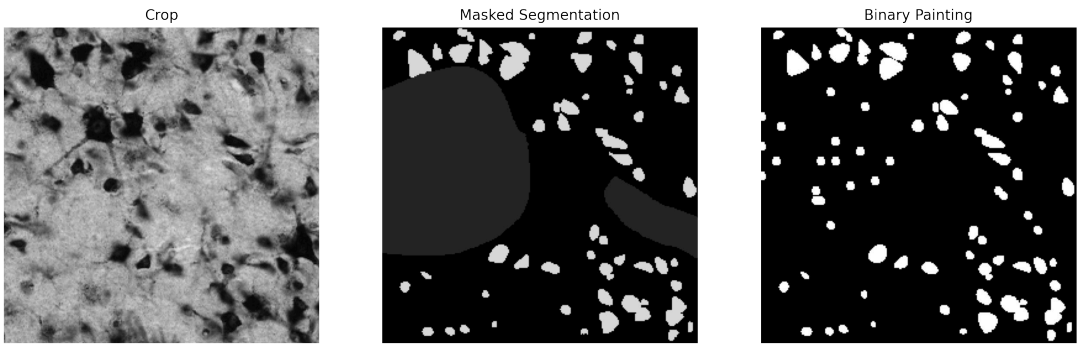


Figure 36: VAEAC Shape Inpainting inference example with the mean-return setting.

While there is more variety to generated cell shapes than with the mean-return settings, the generated cell shapes don't match those of real segmented cells and often become thin

and stretched out. This is a very consistent shortcoming and clearly places this model below the cGlow model with respect to visual quality.

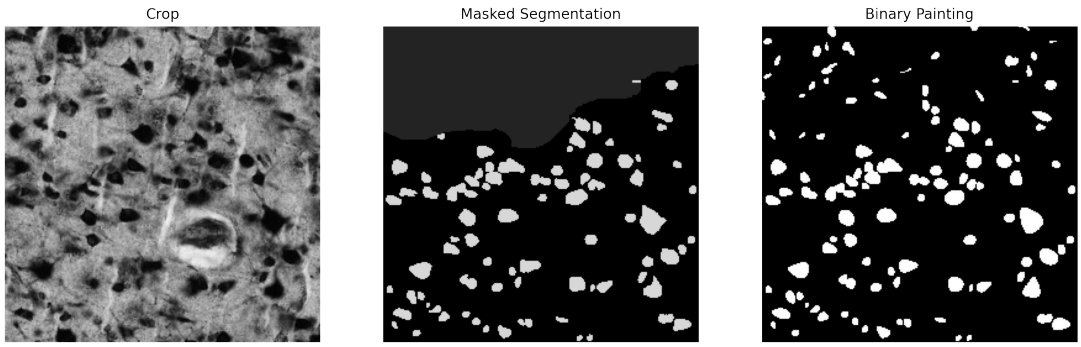
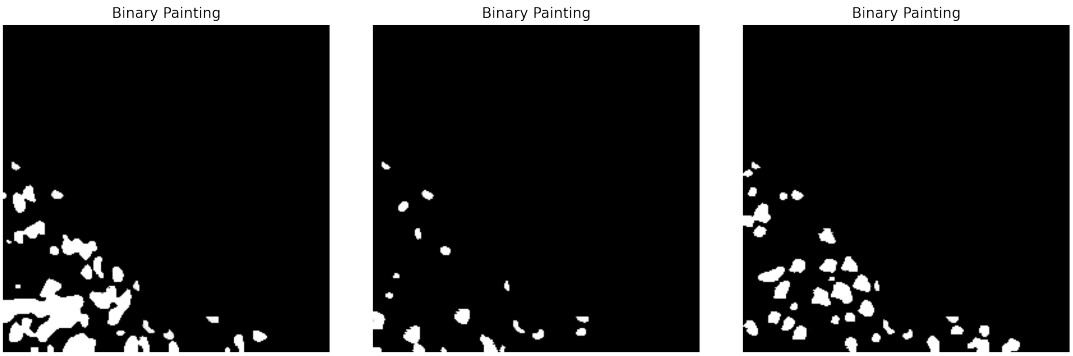


Figure 37: VAEAC Shape Inpainting inference example with the sample-return setting.

3.6.4 Binary Inpainting GAN

This was the naive approach to the Binary Inpainting stage. It approaches the task as a regular inpainting task with a model similar to the regular II model. The experiments here show that the GAN has severe problems with mode collapse on this task. The core idea of doing inpainting on a segmentation image first, is to focus on specific statistics of an image, such as the density and sizes of the segmented cells. The GAN however was not able to focus on these statistics and showed no convergence behavior regarding them. Figure 38 shows the same inference example during training from different epochs.

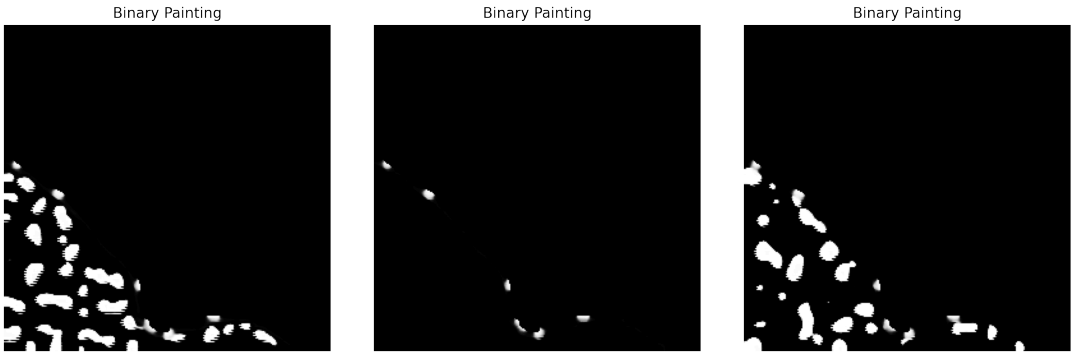
For better comparability, the figures below only show the corrupted region of the painting.



(a) Epoch 90 (b) Epoch 93 (c) Epoch 98

Figure 38: The same inference example during training in different epochs. These are not from the initial stages of training anymore and the model is cycling from large amounts of segmented cells to fewer and back to a large amount.

The GAN effectively cycles through different states, where the paintings contain some amount of segmented cells, from very little to too many, and of varying sizes, from very small to very large. While it was possible to get a 'good' model with decent statistics when stopping the training at exactly the right time, the model never displayed any form of convergence behavior regarding these statistics.



(a) Epoch 22 (b) Epoch 24 (c) Epoch 25

Figure 39: The same inference example during training in different epochs. In epoch 24, the model didn't generate any new cells, apart from the border region of the corrupted area where the reconstruction loss provides feedback and cells that were cut off by the mask are completed in the painting.

There also was a second form of mode collapse, where the generator wouldn't generate any cells at all, apart from the border regions of the mask that are affected by a reconstruction loss. This occurred at seemingly random points during training and happened in various model settings. This might happen only for a few epochs or the model can stagnate in this state. Figure 39 shows an example of this behavior.

3.7 Qualitative Evaluation of Image Inpainting Models

3.7.1 Image Inpainting

The Image Inpainting model is only evaluated qualitatively, because of the existence of the Binary Inpainting stage. All efforts towards quantitative performance in the Neural Inpainting pipeline are focused on statistical properties of the crops with respect to cells. These properties are fully captured in the cell segmentation images and the Binary Inpainting model should be used when these are of interest. That being said, one interest here besides visual sample quality is the correct inclusion of the information provided by the binary painting into the image painting.

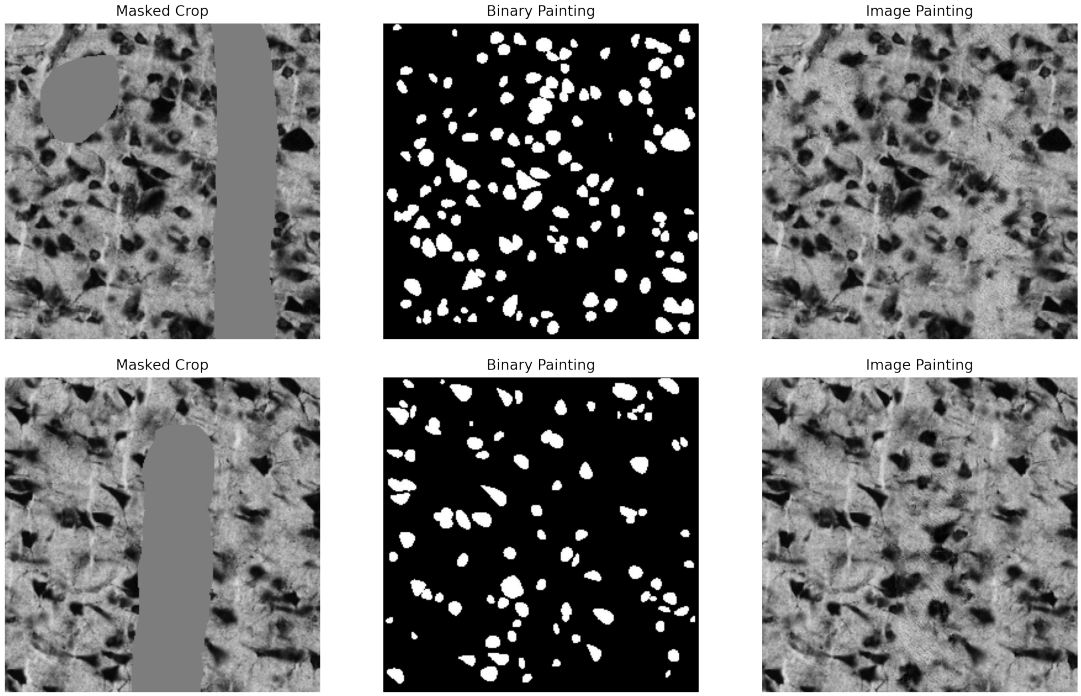


Figure 40: Two inference examples of the Image Inpainting model. These are examples from the intact dataset that were not seen during training. The paintings display good visual quality.

Paintings from the Image Inpainting model display overall good visual quality, with mostly visually convincing examples. At best, the paintings are comparable or even indistinguishable to the ground truth at first glance. This applies especially to masks that are relatively small compared to the crop dimension. Depending on the training run of the model, some examples show undesirable visual artifacts. Although the final model is able to produce good sample quality most of the time, it also suffers from severe artifacting on specific examples.

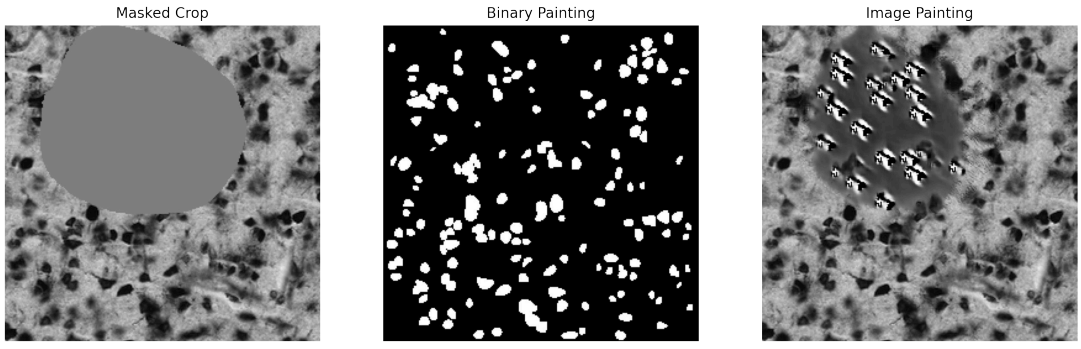


Figure 41: An inference example of the Image Inpainting model. It is from the intact dataset and was not seen during training. The painting displays strong visual artifacting.

When examples with visual flaws occur, it is likely that either the mask is relatively large or the crop has an unusual texture that doesn't appear frequently, if at all, during training. The latter also causes problems when using the Neural Inpainting pipeline as a whole on data from the artifact dataset. Section A.3 provides a larger number of random examples.

As mentioned, the correct inclusion of the information provided by the binary painting into the image painting is also a desired quality of the Image Inpainting model here. Figure 42 shows two comparisons between the binary painting and a cell segmentation image of the image painting. The segmentation images only shows the masked regions. As visible, some segmented cells can clearly be matched between the segmentations, but there are plenty of differences, in shapes and count. During the early phases of training, the image paintings include the binary paintings faithfully, but as the GAN improves its textural features, it tends to ignore some of the input from the binary painting.

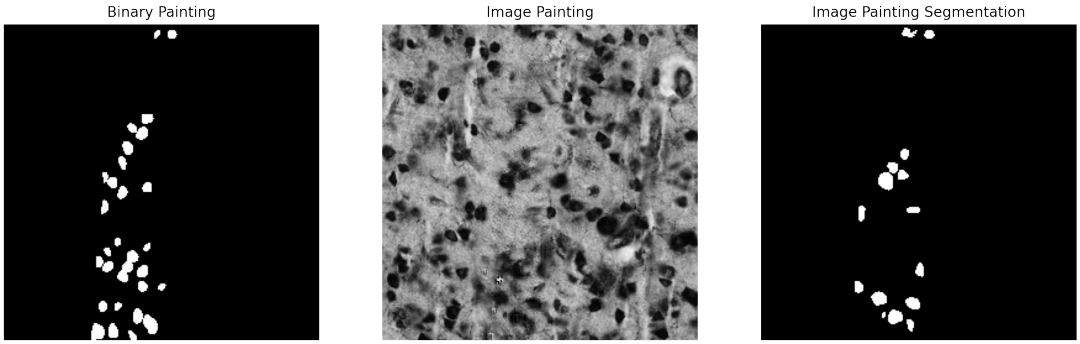


Figure 42: An inference examples of the Image Inpainting model comparing the binary painting to a segmentation image of the image painting. The same masked area is cut out in the binary painting and segmentation image of the image painting. This example was not seen during training.

3.7.2 Image Inpainting on the Artifact Dataset

All evaluation runs so far have been conducted on the same dataset that was used for training of the model, altough with data unseen during training. Evaluating the performance of the Image Inpainting model on the artifact datasets indicates to what extend the model is able to transfer its performance to unseen artifact data.

The results show that the models performs signifcantly worse on the artifact dataset, especially the Image Inpainting model. While some samples still show decent quality, see figure 43, these appear less frequent. More frequently appear examples with visual artifacting, particularly on larger masks, as well as examples with a mismatch between generated and original content. The generated region looks decent, but doesn't fit the rest of the image. See figure 44 for an example of this. Section A.3 in the appendix contains a large number of random examples for the Neural Inpainting pipeline as a whole. The models clearly perform better when evaluated individually and when test and training data come from the same dataset.

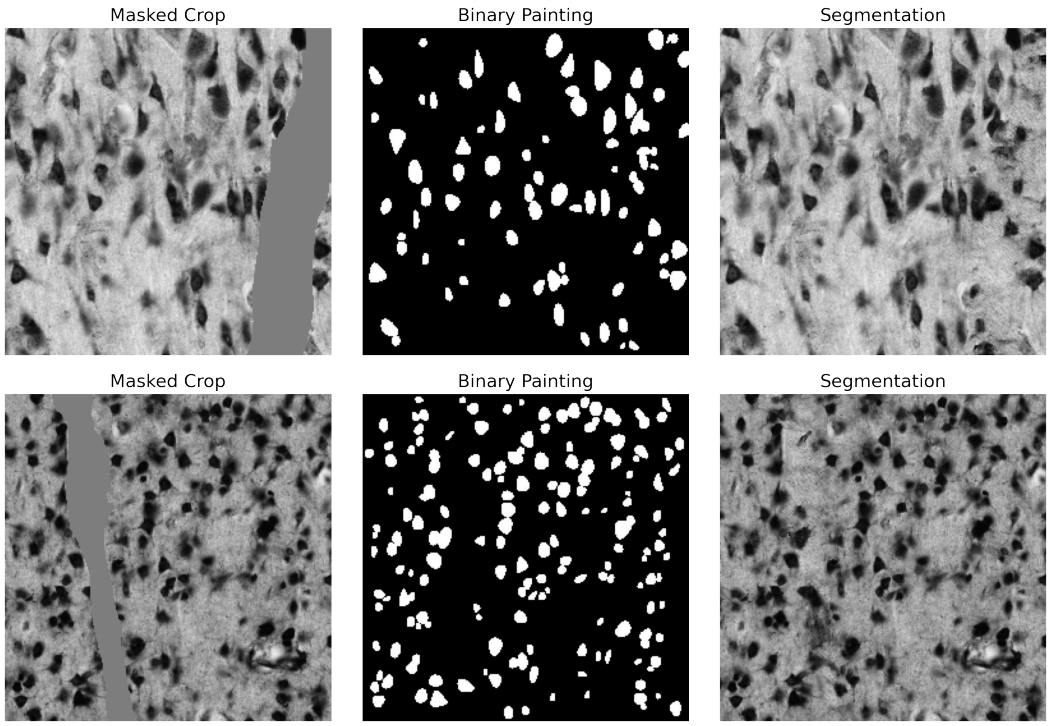


Figure 43: Two inference examples of the Image Inpainting model from the artifact dataset. They display good visual quality.

The BinaryInpainting model als has problems to properly adapt to a lot of examples from the artifact dataset. In the first examples in figure 44, the generated cells are clearly too large for the composition of the crop and the Image Inpainting model subsequently generates large cells as well.

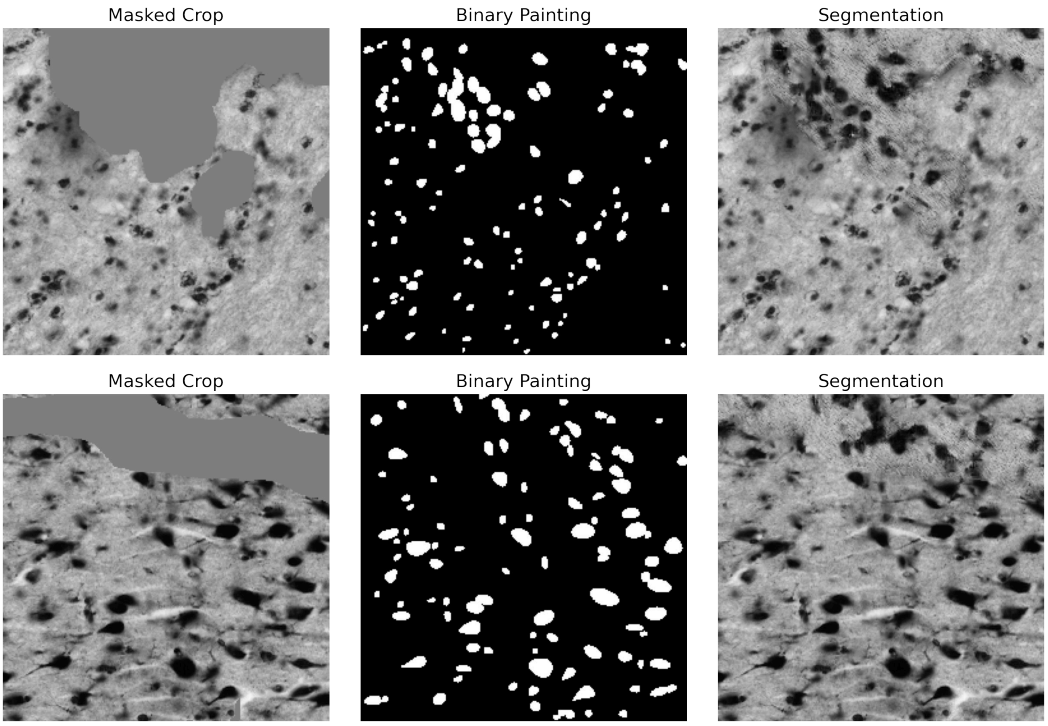


Figure 44: Two inference examples of the Image Inpainting model from the artifact dataset. They display lacking quality from both, the Image and Binary Inpainting models.

3.7.3 Image Inpainting without Binary Inpainting

The model here serves as a comparison for the full Image Inpainting model. As mentioned earlier, it was tuned and trained without the additional inputs from the Binary Inpainting stage. It is expected to perform well in terms of sample quality, since the information from the binary painting doesn't help much with this.

Visually, the quality of the generated content is good for most examples. The paintings are comparable in texture to the ground truth for the corrupted region. Especially for medium and smaller masks, the corrupted and uncorrupted regions are indistinguishable at first glance. Similar to the Image Inpainting stage, visual imperfections can occur when the mask is large relative to the crop dimension or when the crop has an unusual texture that doesn't appear often or at all during training. Figure 45 shows some good examples and figure 46 displays problematic samples. A larger number of random examples can be found in section A.3.

Although this specific model doesn't show artifacts like in figure 41 for the Image Inpainting model, it has similar problems with specific examples. The type of artifacts differs between training runs, but they always appear in some form.

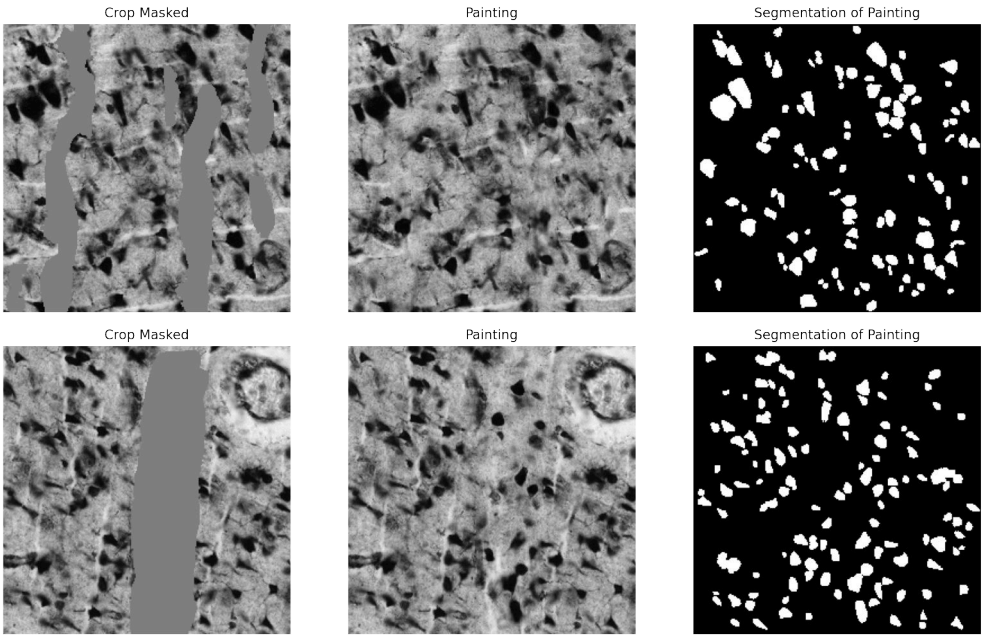


Figure 45: Two inference examples of the Image Inpainting model when trained without binary paintings as auxiliary input. These are examples from the intact dataset that were not seen during training, with good visual quality.

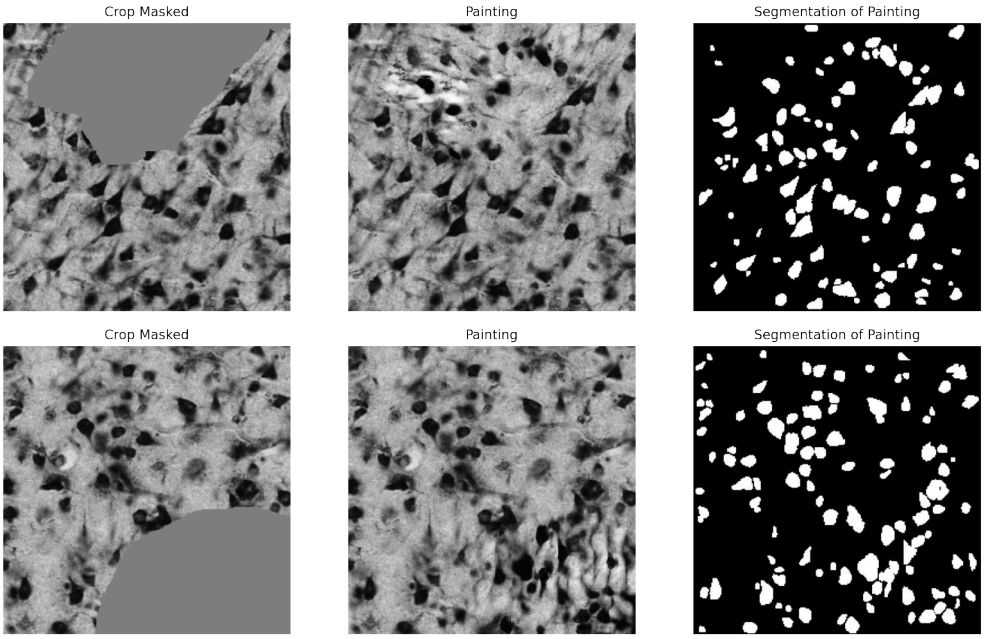


Figure 46: Two inference examples of the Image Inpainting model when trained without binary paintings as auxiliary input. These are examples from the intact dataset that were not seen during training, with visual artifacting and imperfections.

4 Discussion and Conclusion

4.1 Discussion

The purpose of this section is to interpret the experimental findings from section 3, relate them to the research objective and give possible explanations for the shortcomings in different areas. Each stage of the Neural Inpainting pipeline is briefly covered, including comparisons to alternative approaches and then the model as a whole is discussed.

4.1.1 Artifact Localization

The Artifact Localization is the simplest stage of the pipeline. The overall performance here meets the expectations, but individual examples are segmented very poorly, see figure 24. This inconsistency is the main downfall here, however it is difficult to avoid because the various kinds of artifacts are so different. Fully black or fully white ones are easy to detect accurately, ones including some form of blur are more difficult to segment correctly. The upside of this is that the difficult artifact are difficult because they still contain a good amount of semantic and textural information. If the network doesn't detect a blurry region as an artifact, this region is likely to not distort segmented cell statistics as much, if at all.

An increase in variety of artifact types should help to improve the model's performance, since 45 examples are not enough to cover a wide range of artifact scenarios. Different factors influence the composition of a crop signifcantly, including the artifact type itself as well as the brain of origin and the brain region.

4.1.2 Binary Inpainting

Image Inpainting without Binary Inpainting The motivation for the inclusion of the Binary Inpainting stage in the reparation process is an improvement of the Image Inpainting model. We can compare the results from section 3.7 between the Image Inpainting model with and without help from the Binary Inpainting stage, to see whether there is a meaningful improvement. The main research objective is to generate paintings with convincing visual quality and good statistical properties, such as segmented cell density and sizes. The model here delivers visual sample quality on par, if not even a bit better than that of the full model and displays statistical results only slightly worse than the Binary Inpainting stage. The main drawback of the GAN here is its inability to optimize for the different statistical properties separately. It is very difficult to optimize for sample quality in the first place and there is no direct way to tune the model for specific features, e.g. segmented cell count. This is exactly what the Binary Inpainting stage provides by outsourcing this task to specialized models and excluding the GAN from this process. Another issue with this model was inconsistency between runs, even with the same training and architectural setup. Some runs performed very well with respect to the segmented cell sizes, but failed to learn the segmented cell count distribution properly, vice versa for other runs. The final model was simply the best when considering all aspects, with the only clear shortcoming being a bias in the segmented cell size experiments.

Density Inpainting The results here show that the network is able to learn the distribution of segmented cell locations within the ground truth data to a reasonable degree. The performance in terms of raw generated cell count is satisfactory, with an average error of 3.9%, relative to the total cell count in the ground truth crops, in 10000 examples. The distribution of segmented cell counts was adequately reproduced by the model, the two main shortcomings are a lack of variance and a slight bias towards a smaller amount of generated cells than in the ground truth images. See figure 25 (b) for the best illustration of these results. The reason for this bias might be the overall lower amount of probability mass in the masked KDE compared to the ground truth KDE. Because each segmented cell has a global influence on the KDE image, the uncorrupted area in the ground truth KDE contains significantly more probability mass than the uncorrupted area in the masked KDE image. Another reason might be slight imbalances in the average amount of segmented cells per crop between the training and test data, since these datasets were rather small.

The design choice of zero padding for the Fourier matrices is not a good solution for the variable sequence length problem, as it prevents the model from ever seeing unusually dense examples during training. This might play a role in the network's adversity towards producing high cell count samples, as seen in the boxplot in figure 25 (b).

The second goal of the Density Inpainting step, for the generated content to adhere to high and low density regions within the crops, did not work as well as expected. The KDE paintings never match the ground truth KDE images visually and the generated area is always easy to spot by eye. This is probably another result of the above described problem. Also, while the data representation in form of a KDE image does help with the superposition problem, the qualitative evaluation shows that it is not completely eliminated. The KDE format merely offers a way of controlling the sharpness of the center locations by literally blurring their position in the ground truth image. By using higher values for the standard deviation of the Gaussian kernel when computing the KDE, the superposition problem disappears, but the KDE's become more uniform and lose information as a result. Using smaller values leads to sharper feedback and more information on the correct center locations, but the superposition problem interferes stronger in the learning process.

While the results overall don't quite meet the expected performance, they provide a sufficient foundation for the Shape Inpainting and Image Inpainting to work properly. As mentioned in section 3.6.1 already, without the split of the Binary Inpainting stage into a two-step process, the Shape Inpainting via a Normalizing Flow model wouldn't be possible in its current form. The NF model can't sample the number of new cells, hence this has to be done in a prior step. A re-design of this step is certainly possible and might improve the results. This is also the only step in the Neural Inpainting pipeline that doesn't require a deep neural network, since the distribution of cell locations is only 2-dimensional.

Shape Inpainting The Shape Inpainting step is the most difficult step of the Neural Inpainting pipeline. A network needs to very accurately learn a high dimensional distribution of segmented cell shapes in a setup that exhibits a strong case of the superposition problem. Additionally, it is a conditional task where the network needs to adhere to the

information of already known segmented cell shapes from an input crop. The results here are sufficient for the whole model pipeline to work, but leave a lot of room for improvement, as evident from the evaluation in section 3.6.2.

A big problem during the development of this step was an unexpected collapse behavior for certain hyperparamter settings. When using low batch sizes ($\leq$16) together with low learning rates ($\leq$1e-3), the generated segmented cell contours collapsed to very small segmented cells, even down to single pixels. This corresponds to the values in the Fourier matrices converging to 0. Interestingly, the settings of hyperparameters that result in such a collapse are the ones that achieve the best results in terms of the NLL-loss. Since this is unsupervised learning, good loss values don't directly correspond to good inference performance, still it is unclear why this happens and why these results achieve a low NLL. The use of column-wise data normalization for the Fourier matrices resulted in a slightly different collapse behavior to the mean segmented cell size instead of extremely small sizes. For more information on the appearance of this behavior can be found in section A.4.3 in the appendix.

Similar to the segmented cell count, the learned segmented cell size distribution shows a bias towards smaller segmented cells and an overall lack of variance. The lack of variance is not a surprising result, especially not on this task with the superposition problem present. A possible explanation for the bias is that the network's inability to generate larger segmented cells causes a bias in the average segmented cell size towards smaller sizes. Another factor might be a difference in the overall segmented cell size distribution between the test set and the training set. Since both were rather small, they don't share the exact same segmented cell size distributions.

The segmented cell eccentricity distribution was learned very accurately, especially compared to the VAEAC results. As seen in the qualitative evaluation however, this doesn't necessarily result in good generated cell shapes.

In its current form, the Shape Inpainting model is not conditioned on any cell locations, not from the known segmented cells, nor from the ones generated in the Density Inpainting step. While these are used for the placement of the cells in the binary painting and they determine the number of cells that the Shape Inpainting model generates, it does not receive them as input. The direct implication here is that relations between segmented cells and their shapes, based on their location, can't be modeled in the Shape Inpainting step. Instead, an overall distribution of feasible cell shapes based on the shapes of all real, segmented cells from the input is learned and sampled from. How to condition the model correctly on the location information is unclear. They were treated separately from the Fourier matrix, because simply concatenating them to the front of each row didn't improve the results. This issue is related to the problem, that the conditioning network of the cGlow model is a convolutional one. A redesign of this conditioning network, with an architecture that employs a more fitting inductive bias for Fourier matrices, is certainly an interesting proposition, but was outside of the scope of this work. The inclusion of cell locations in this process, separate from the shape coefficients, should play a significant role in the architectural design here.

In comparison to the cGlow model, the VAEAC model performed significantly worse. While the mean-return setting is not usable at all due to its complete lack of variance,

the visual sample quality with the sample-return setting doesn't hold up to the real segmented cell shapes. The model is simply not able to approximate the segmented cell shape distribution well enough. A big problem here is that the decoder parameterizes a multivariate Gaussian distribution with diagonal covariance matrix. When sampling from this, the parameters are sampled independent of each other, resulting in odd generated cell shapes, even if the overall variance is accurate. The cGlow model doesn't have this independence assumption and subsequently produces more accurate samples.

Comparison to Binary Inpainting GAN A GAN proved to be inherently the wrong approach to the Binary Inpainting stage. It showed severe problems with mode collapse in the form of cycling between states containing different amounts of sizes of cells, both from too little to too much, and by randomly outputting black areas that don't contain any cells. The cycling between solutions appears to be a result of the GANs inability to deal with the superposition problem on this dataset. The black output problem appeared to be a reasonable solution to the discriminator, since black areas are often part of the real data. It is difficult for the model to distinguish at what size these areas become unlikely. While the cycling is an inherent problem of the GAN approach, the black output problem might be fixed by using a different data format than 2d-segmentation images.

The main reason why a GAN is inherently the wrong approach here is its feature selection. Since the goal of the Binary Inpainting task is to get specific features (like density and sizes of segmented cells) right, the model and objective should reflect this. While a GAN does tend to pick out specific features from the input data, it's not always the desired features and it constantly changes its focus, hence the mode collapse problem. Trying to remedy this with the unrolledGAN [25] approach or a Discriminator pool resulted in much slower learning and amplified the black output problem since this seems to be a solution that always works for the Discriminator.

4.1.3 Image Inpainting

The Image Inpainting stage is solely responsible for the visual quality of the final image painting, which was one of the two criteria of the research objective. The modular U-Net, described in section 2.1, was developed primarily for this task, as a mixture of tricks and features that were picked to fit the task and data, with everything based on the original U-Net for biomedical image segmentation [30]. Besides good visual quality, the second goal here was the faithful inclusion of the information provided in the binary painting.

The final model is able to produce convincing visual sample quality most of the time, with the main shortcoming being strong artifacting on specific examples. This problem is expected to improve with a larger crop dimension, specifically a larger ratio of unmasked to masked pixels, because such improvements were visible during the development. Because the Shape Inpainting model restricts the crop dimension to be constant, the Image Inpainting model had to be tuned and improved in other ways though. A tuning and retraining of the whole pipeline on more data and with a larger crop dimension is certainly desirable.

The second goal, the faithful inclusion of information from the binary painting, was not

achieved by any variant of the Image Inpainting model during development. The discriminator also receives the binary painting and can use this information to discriminate the previously corrupted area if the generator doesn't include it correctly. However, the generator tends to ignore the binary painting in the later stages of training and focus on better textural quality instead. The implication of this is that the image painting can't be used to extract accurate cell statistics, although they are likely still better than leaving artifacts in. Instead, cell statistics should be taken directly from the binary painting, as these are much more comparable to the ground truth data used for the experiments. As the only effort to enforce the correct inclusion of the binary painting is to also give it to the discriminator, this aspect can certainly be improved by incorporating additional measures here, e.g. a loss term similar to the weighted reconstruction loss, but for the binary painting. This remains for future work.

Notably, the addition of the binary painting to the inputs for the Image Inpainting stage lead to more instable training behavior and worse results. With the same training and model setup, the Image Inpainting model without additional inputs displayed more stable training. Again, a better way of incorporating the binary painting into the learning process here might alleviate this problem.

Lastly, the inconsistency between training runs of the same model and training settings is important to mention here. Not only can small changes in hyperparameter or architectural setup have strong effects on the results, simply redoing a training run can result in a model with different results. This includes different kinds and frequency of artifacting problems and different overall visual quality. Part of the reason for this is the inconsistency within a training run of a GAN, they often have their best results during a training run and not necessarily at the end of it. Finally, the two best models show different tradeoffs between visual quality and artifacting problems. The model from the experiments in section 3.7 has overall good visual quality, but strong artifacting problems, when artifacting occurs. The other model has overall mediocre visual quality, but much less severe artifacting problems. The only difference between the two is that the former has a larger number of learnable parameters, because its width unit parameter is larger. Despite its artifacting problems though, the first model demonstrates that good visual quality is possible with this architecture and training setup and the artifacting problem is left to be solved in future work, perhaps with a higher crop dimension.

4.1.4 Neural Inpainting

A very important distinction in the evaluation of this method is the dataset used. There is no choice for the Artifact Localization, but the other stages can be evaluated on the dataset of intact scans or on the dataset of scans with real artifacts. For the evaluation in section 3, the test sets were always from the same dataset as the training sets, so that ground truth data could be used to evaluate the results. In theory, going from the intact dataset to the artifact dataset should not make a difference for the Binary and Image Inpainting stages, since both receive incomplete crops with uncorrupted areas and a mask. In practice however, this proves to be a challenge, especially for the Image Inpainting stage. When evaluating the complete Neural Inpainting pipeline on the artifact dataset, many paintings display the same characteristic: Even though the generated part looks

realistic sometimes, it clearly doesn't fit the original part of the image.

The problem here is that the uncorrupted parts in both datasets are not alike, because the scans in the artifact dataset were specifically picked for their artifacts and they come from different brains and brain regions, while the scans in the intact dataset were picked for their even and large amount of cortical area and come from similar regions of the same brain. The visual composition of a scan is determined by multiple factors, including which brain it is from, the region within the brain or the cutting angle relative to the surface. All of these and more aspects are not properly balanced within and between the two datasets, resulting in problems for the Image Inpainting network on the artifact dataset. Especially when it comes to the textural component of the images, the generated content of the Image Inpainting model often doesn't fit into the image at all.

An easy solution here would be to also use the artifact dataset for training of the Image Inpainting model. This is not directly possible however, because the model employs a reconstruction loss around the edges of the corrupted areas. Unfortunately, data augmentation doesn't solve this problem either, because it doesn't affect the textural information in the crops. A missed opportunity for potentially useful data agumentation is some form of color jitter. This could help to transfer the performance to different brains, which have different overall brightness and contrast, due to differences in the staining and scanning process. The best solution is still the inclusion of much more data to balance out the contents of the two datasets.

While the different stages in the pipeline perform quite well individually, this problem severely limits the model's usability as an inpainting model in its current state. The results when using the Neural Inpainting pipeline as a whole are unsatisfactory and do not meet the goal of visually convincing sample quality. The Binary Inpainting stage, and with it the statistical part of the model, should be less affected by this. Because this can't be quantitatively evaluated however, there is no guarantee for this.

4.2 Conclusion

This master thesis project aimed to develop a novel approach to the task of repairing artifacts in histological brain sections, which has previously been done manually or with methods that estimate simple features, like cell locations, based on information from neighboring sections [7]. The results demonstrate that it is possible to learn various properties of crops in histological brain scans and to repair damaged areas of crops with deep generative models. By no means does it exhaust the possibilities to achieve this, nor does it provide results that can't be improved upon. Cell statistics, comparable to statistics from ground truth data when evaluated on artificially masked examples, can be extracted from the Binary Inpainting stage, while the Image Inpainting model is able to generate visually convining examples in the right circumstances.

The crucial steps in making the model pipeline work are the addition of the Binary Inpainting stage and the efficient parameterization of cell countours via the Fourier series. The Binary Inpainting stage performs a separate reparation on a cell segmentation image and generates information on the locations and shapes for new cells. This allows for the design and optimization of the reparation process specific to statistical correctness

regarding cell bodies in the scans. With a single Inpainting model, this would be much harder to do. The use of the Fourier format results in a data representation of cell segmentation images that is much smaller than 2d-images. A single image can be described in a matrix of Fourier coefficients, with each row parameterizing the contour of a segmented cell. This enabled the use of Normalizing Flow models to infer the shapes of new cells from the segmented cells in the uncorrupted area.

While each individual model in the Neural Inpainting pipeline performs well on their respective test data, the model as a whole produces unsatisfactory results when applied to test images with real artifacts. This is due to the limited size of the datasets and their different compositions. Especially the Image Inpainting model is not able to transfer its performance to examples that differ too much from its training set. A tuning and re-training with more data and a larger crop dimension should alleviate some of the problems of the Image Inpainting model. In its current state, the results from this model on unseen artifact data are not usable.

The development of this model also showed various opportunities for further improvements. These include a better way of incorporating the binary painting into the image painting, a better solution for the variable length of the Fourier matrices than zero padding, the use of a non-convolutional conditioning network in the cGlow model and better conditioning of the Shape Inpainting model on information about the locations of segmented and new cells.

Apart from improvements of the existing Neural Inpainting framework, the idea of conditional models for histological inpainting can be extended to focus on more features than the cell bodies. Leaving the first and last stages as is, a possible expansion of this pipeline is be the inclusion of additional models that specialize on learning specific properties of the brain scans. These could then be provided as additional conditional inputs to the last step and improve the realism and statistical reliability of the final inpainting result.

A Appendix

A.1 Model and Training Details

This section contains model and training details for all models featured in this thesis. The first subsection provides them for the models in the Neural Inpainting pipeline, the second subsection for the additional models that were featured in the experiments section.

A.1.1 Neural Inpainting Models

	No. of Parameters
Artifact Localization	471,809
Density Inpainting	229,113
Shape Inpainting	37,787,956
Image Inpainting G	7,182,945
Image Inpainting D	3,369,777

Below is a table that describes the different U-Net variants used in this thesis. Denote model width, skip connections, normalization, activation, gated convolutions and dilated convolutions in the bottleneck as width, skip, norm, act, gated and dilated respectively.

	Width	Skip	Norm	Act	Gated	Dilated
Artifact Localization	16	✓	Batch	ReLU	X	X
Density Inpainting	8	✓	Batch	ReLU	✓	X
Image Inpainting G	32	✓	Batch	ReLU	✓	✓
Image Inpainting D	32	✓	Batch, Spectral	LeakyReLU	✓	X

The layer configuration for all variants of the modular U-Net is as follows. Denote kernel size, dilation, stride and number of output channels as K, D, S and C respectively. For readability, the width unit was set to 16 here. A different width unit, e.g. 32, would result in double the number of channels for every convolutional layer.

Modular U-Net Encoder: K3S1C16 - K3S1C16 - K2S2C16 - K3S1C32 - K3S1C32 - K2S2C32 - K3S1C64 - K3S1C64 - K2S2C64 - K3S1C128 - K3S1C128

Bottleneck: K3D2S1C128 - K3D4S1C128 - K3D8S1C128

Modular U-Net Decoder: NN-Upsample - K1S0C64 - K3S1C64 - K3S1C64 - NN-Upsample - K1S0C32 - K3S1C32 - K3S1C32 - NN-Upsample - K1S0C16 - K3S1C16 - K3S1C16 - K1S0C1

Every convolutional layer with kernel size 3 is followed by a normalization layer and an activation layer. If spectral normalization is used, the batch normalization layer is omitted. Depending on the task, a sigmoid or tanh activation is used after the last layer, e.g. tanh for the II generator and sigmoid for the II discriminator.

The table below describes the training setups for the four networks of the Neural Inpainting pipeline. Denote batch size, learning rate, PyTorch's Adam weight decay parameter and learning rate decay by Bs, Lr, Wd, LrDc respectively.

	Epochs	Bs	Lr	Wd	LrDc	Loss	Optimizer
Artifact Localization	300	16	4e-4	5e-3	✓	BCELoss	Adam
Density Inpainting	100	64	2e-3	0	✓	L1-Loss	Adam
Shape Inpainting	200	64	5e-3	0	✓	NLL	Adam
Image Inpainting G	200	32	4e-4	0	X	MU-Net GAN [2]	Adam
Image Inpainting D	200	32	1e-4	0	X	MU-Net GAN	Adam

The Image Inpainting models were the only ones to not use the default beta values in the Adam optimizer. They employed $\beta_1 = 0.5$.

Learning rate schedules:

Artifact Localization: Step decay with factor 0.9 every 10 epochs
Density Inpainting: Multi step decay with factor 0.5 in epochs 5, 10, 15 and 25
Shape Inpainting: Step decay with factor 0.9 every 10 epochs

Here are the additional hyperparameters needed for the cGlow configuration.
The flow has 3 levels with depth 16 each (L=3, K=16). The conditioning network has 128 channels in the convolutional part and 64 parameters in the linear hidden layer. The convolutional network in the affine coupling layer has 256 hidden channels.

A.1.2 Additional Models

Binary Inpainting GAN The GAN model employed for the Binary Inpainting stage was very similar to the GAN model from the Image Inpainting stage, in that both models are a variation of the modular U-Net from section 2.1. The generator contained 7.183.521 and the discriminator 118.049 learnable parameters. Below is a description of the architectural and hyperparameter setup.

	Width	Skip	Norm	Act	Gated	Dilated
Binary Inpainting GAN G	32	✓	Batch	ReLU	✓	✓
Binary Inpainting GAN D	8	✓	Spectral	LeakyReLU	X	X

	Epochs	Bs	Lr	Wd	LrDc	Loss	Optimizer
BI GAN G	100	16	2e-4	0	X	MU-Net GAN	Adam
BI GAN D	100	16	1e-4	0	X	MU-Net GAN	Adam

VAEAC The VAEAC model used in this thesis is a slightly modified version of the model from the paper by [13]. Because the Fourier format has a smaller feature dimension than regular images, the last downsampling step including 4 res-blocks, average pooling and a 1x1 convolution, was omitted in the proposal and prior network. Accordingly, the corresponding first upsampling step in the generative network was omitted as well. The optimizer was kept and an alternative return option was added to the sampler, that applies the '.sample()' function to the PyTorch distribution object which is parameterized by the decoder's output. The implementation of the forward pass and the objective was modified to take Fourier matrices as input and target, as well as accommodate masks for the masked entries and padding to the maximum number of cells per crop.

[2] see section 2.1 for the U-Net GAN loss and section 2.6 for the weighted reconstruction loss.

The final model had 1.404.882 learnable parameters and the following hyperparameter setup.

	Epochs	Bs	Lr	Wd	LrDc	Loss	Optimizer
VAEAC	100	32	4e-4	0	X	VLB	Adam

Below are model and training details for the models that are features in the experiments section, but are not part of the Neural Inpainting pipeline.

Image Inpainting without Binary Inpainting To provide a comparison for the final Image Inpainting model, one without the additional inputs from the Binary Inpainting stage was tuned and trained separately. Similar to the Image Inpainting model, it is a variation of the modular U-Net from section 2.1. It features the same architectural setup, except for hyperparameter choices, like the width unit. The generator contained 1.798.673 and the discriminator 911.825 learnable parameters. Below is a description of the architectural and hyperparameter setup.

	Width	Skip	Norm	Act	Gated	Dilated
II w/o BI GAN G	16	✓	Batch	ReLU	✓	✓
II w/o BI GAN D	16	✓	Batch, Spectral	LeakyReLU	✓	X

	Epochs	Bs	Lr	Wd	LrDc	Loss	Optimizer
BI GAN G	200	16	1e-4	0	X	MU-Net GAN [3]	Adam
BI GAN D	200	16	1e-4	0	X	MU-Net GAN	Adam

Like the regular Image Inpainting model, $\beta_1 = 0.5$ was used for both Adam optimizers.

A.2 Representation of Cell Contours via Fourier Format

The following is not a proof of the existence or the convergence of the Fourier series but an illustration of why the computation of the Fourier coefficients is an elegant way to parameterize a shape. We will now derive the formulas for the Fourier coefficients from section 2.5.1.

Consider the function $x_N(t)$, everything works analogously for $y_N(t)$ with $f_y(t)$. Denote $f(t) := f_x(t)$ for simplicity. Expressing the equations with the help of complex numbers makes the math a lot cleaner and easier to understand at the cost of a little work up front. Recall Euler's formula and it's direct implications:

$$
\begin{aligned}
e^{ix} &= \cos(x) + i\sin(x) \quad \text{(Euler's formula)} \\
\cos(x) &= \frac{e^{ix} + e^{-ix}}{2} \\
\sin(x) &= \frac{e^{ix} - e^{-ix}}{2i}
\end{aligned}
$$

Let's start with the Fourier series in exponential form:

$$
x_N(t) = \sum_{n=-N}^{N} c_n e^{n2\pi it}
$$

We assume $x_N(t) = f(t)$, i.e. the existence of the exponential Fourier series for some $\{c_n\}_{n=-N}^{N}$ and a large enough N (can be infinity). Note that a complex exponential of the form $e^{n2\pi it}$ corresponds to a unit circle and changing t corresponds to a rotation around this circle with frequency $2\pi n$. This means that the exponential Fourier series above represents a contour as a sum of circles with different radii c_n. As we increase t from 0 to 1, the circles turn at different frequencies $2\pi n$ and 'draw' the contour $f(t)$.

What is left now is to determine the coefficients $\{c_n\}_{n=-N}^{N}$. All we have to do is to multiply the function $f(t)$ with $e^{-m2\pi it}$ and integrate over t from 0 to 1:

$$\begin{aligned}
&\int_0^1 f(t)e^{-m2\pi it}dt \overset{(1)}{=} \int_0^1 x_N(t)e^{-m2\pi it}dt \\
&\overset{(2)}{=} \int_0^1 \sum_{n=-N}^{N} c_n e^{n2\pi it}e^{-m2\pi it}dt \overset{(3)}{=} \sum_{n=-N}^{N} c_n \int_0^1 e^{(n-m)2\pi it}dt \\
&\overset{(4)}{=} 0 + ... + c_m \int_0^1 e^{2\pi(m-m)it}dt + ... + 0 \overset{(5)}{=} \int_0^1 c_m dt = c_m
\end{aligned}$$

There are a few things happening here:

(1) We write $f(t)$ as $x_N(t)$, which is a sum of complex exponentials with different frequencies $n2\pi$ and different weights c_n.

(2) Instead of multiplying $x_N(t)$ with $e^{-m2\pi it}$, we can multiply each summand of the series individually.

(3) This multiplication corresponds to a change in frequency of each complex exponential. The integral on the left side is effectively an average over the contour. By swapping summation and integration, we instead average over the circles first and then sum the results.

(4) For all circles with non-zero frequency $2(n-m)\pi$ the integral becomes 0, because the average over a circle is its center, 0.

(5) What is left is the only exponential with 0 frequency: c_m.

This YouTube video by [34] provides a geometric and very intuitive explanation of the above derivation. Now, armed with the formula for c_n:

$$c_n = \int_0^1 f(t)e^{-n2\pi it}dt$$

we can derive the formulas for a_n, b_n and the trigonometric Fourier series.

$$\begin{aligned}
c_n &= \int_0^1 f(t)e^{-n2\pi it}dt \\
&= \int_0^1 f(t)(\cos(-n2\pi t) + i\sin(-n2\pi t))dt \\
&= \int_0^1 f(t)\cos(n2\pi t)dt - i\int_0^1 f(t)\sin(n2\pi t)dt \\
&= \left(\frac{a_n}{2} - \frac{b_n}{2}i\right)
\end{aligned}$$

with

$$a_n := 2\int_0^1 f_x(t)\cos(n2\pi t)dt$$
$$b_n := 2\int_0^1 f_x(t)\sin(n2\pi t)dt$$

Analogously

$$c_{-n} = \int_0^1 f(t)e^{n2\pi it}dt = \left(\frac{a_n}{2} + \frac{b_n}{2}i\right)$$

Note that

$$c_0 = \int_0^1 f(t)dt = \frac{a_0}{2}$$

And finally

$$\begin{aligned}
x_N(t) &= \sum_{n=-N}^{N} c_n e^{n2\pi it} \\
&= c_0 + \sum_{n=1}^{N}\left(c_n e^{n2\pi it} + c_{-n}e^{-n2\pi it}\right) \\
&= \frac{a_0}{2} + \sum_{n=1}^{N}\left(\left(\frac{a_n}{2} - \frac{b_n}{2}i\right)e^{n2\pi it} + \left(\frac{a_n}{2} + \frac{b_n}{2}i\right)e^{-n2\pi it}\right) \\
&= \frac{a_0}{2} + \sum_{n=1}^{N}\left(\left(\frac{a_n}{2} + \frac{b_n}{2i}\right)e^{n2\pi it} + \left(\frac{a_n}{2} - \frac{b_n}{2i}\right)e^{-n2\pi it}\right) \\
&= \frac{a_0}{2} + \sum_{n=1}^{N}\left(a_n\left(\frac{e^{n2\pi it} + e^{-n2\pi it}}{2}\right) + b_n\left(\frac{e^{n2\pi it} - e^{-n2\pi it}}{2i}\right)\right) \\
&= \frac{a_0}{2} + \sum_{n=1}^{N}\left(a_n\cos(n2\pi t) + b_n\sin(n2\pi t)\right)
\end{aligned}$$

All of the above can be done for $x_N(t)$ and $y_N(t)$ separately to parameterize a 2-dimensional contour with $4N$ Fourier coefficients.

A.3 Inference Examples

This section contains large, random collections of inference examples for the three stages and the model pipeline as a whole. They all have crop dimension 256 and come from cutouts that were not used during training. The Artefact Localization and Neural Inpainting examples come from the artifact dataset, the Binary and Image Inpainting examples come from the intact dataset. All of them were generated randomly without a review of the results.

A.3.1 Artifact Localization

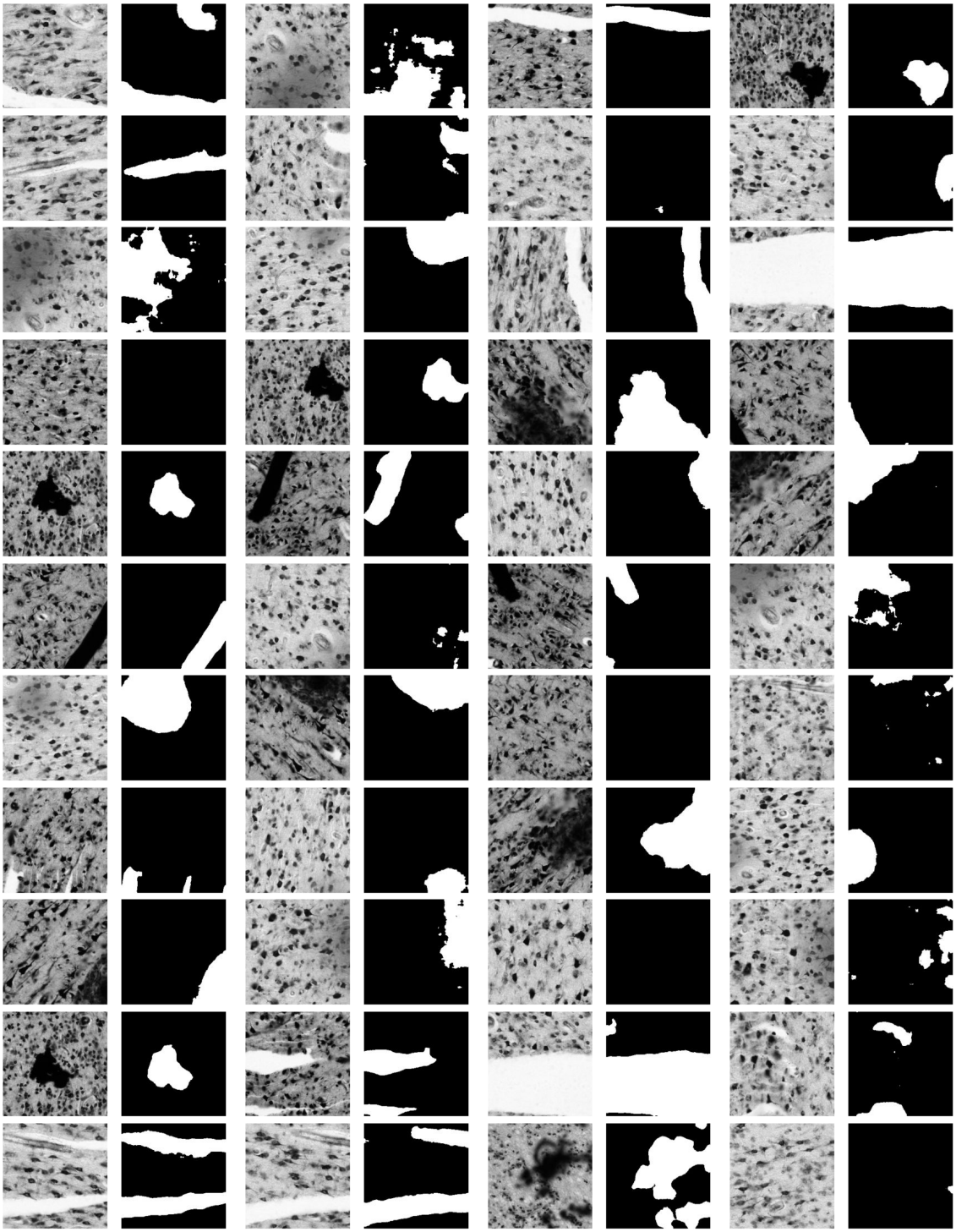


Figure 47: 44 inference examples for the Artifact Localization model from the artifact dataset, with crop dimension 256. These examples were not seen during training.

A.3.2 Binary Inpainting

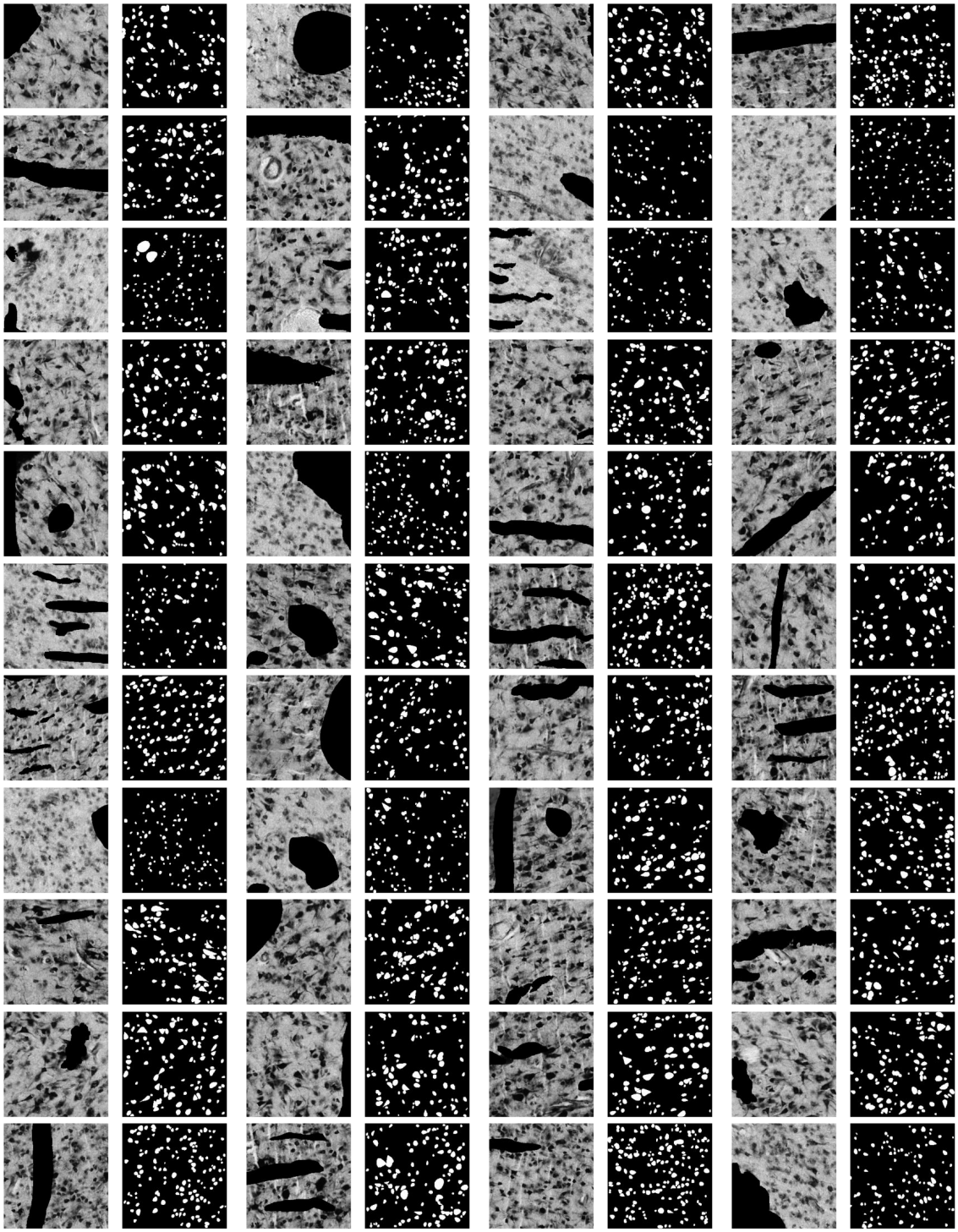


Figure 48: 44 inference examples for the Binary Inpainting model from the intact dataset, with crop dimension 256. These examples were not seen during training.

A.3.3 Image Inpainting

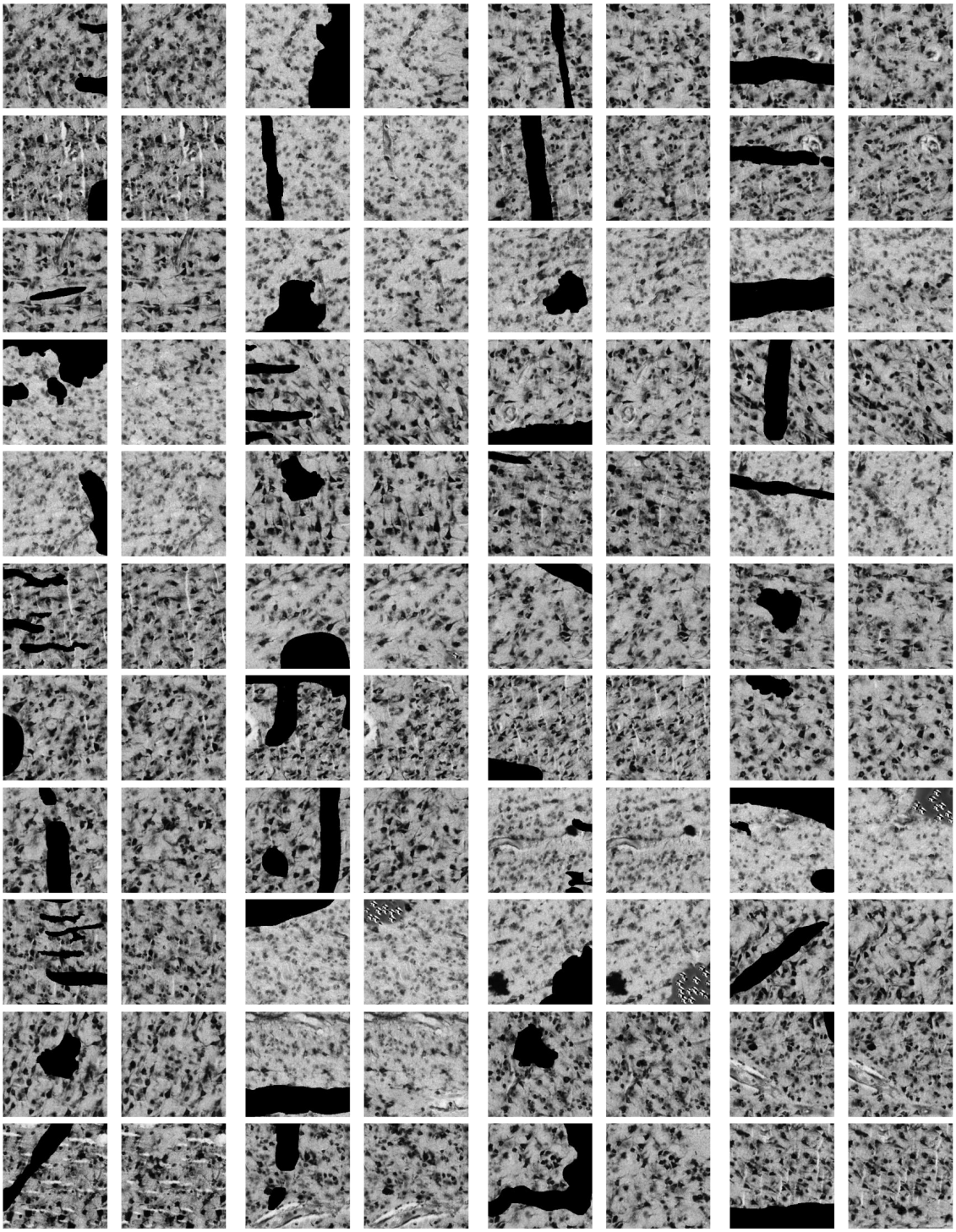


Figure 49: 44 inference examples for the Image Inpainting model from the intact dataset, with crop dimension 256. These examples were not seen during training.

A.3.4 Neural Inpainting

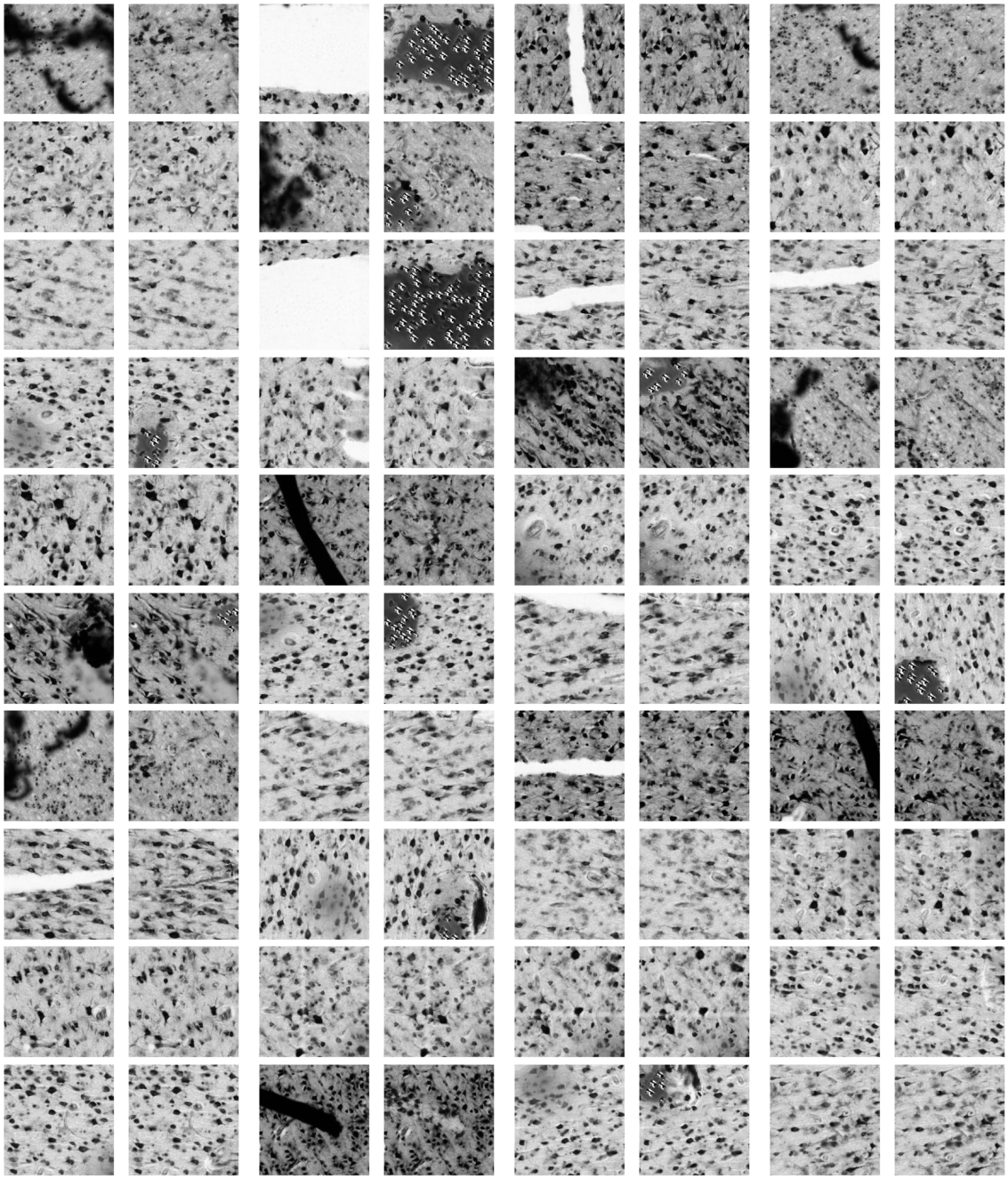


Figure 50: 40 inference examples for the complete Neural Inpainting model from the artifact dataset, with crop dimension 256. These examples were not seen during training by any model.

A.4 Hyperparameter Tuning

The following descriptions apply to all graphics in this section. A slightly transparent solid line depicts training loss, circles depict test loss and a dashed line depicts an exponential moving average of the test loss. The exception to this are graphics where training or test loss are explicitly specified, e.g. in the title. An epoch corresponds to 1000 random crops and masks from one of the two datasets.

A.4.1 Artifact Localization

All training runs of the Artifact Localization model employed exponential learning rate decay where the learning is multiplied by 0.9 every 10 epochs.

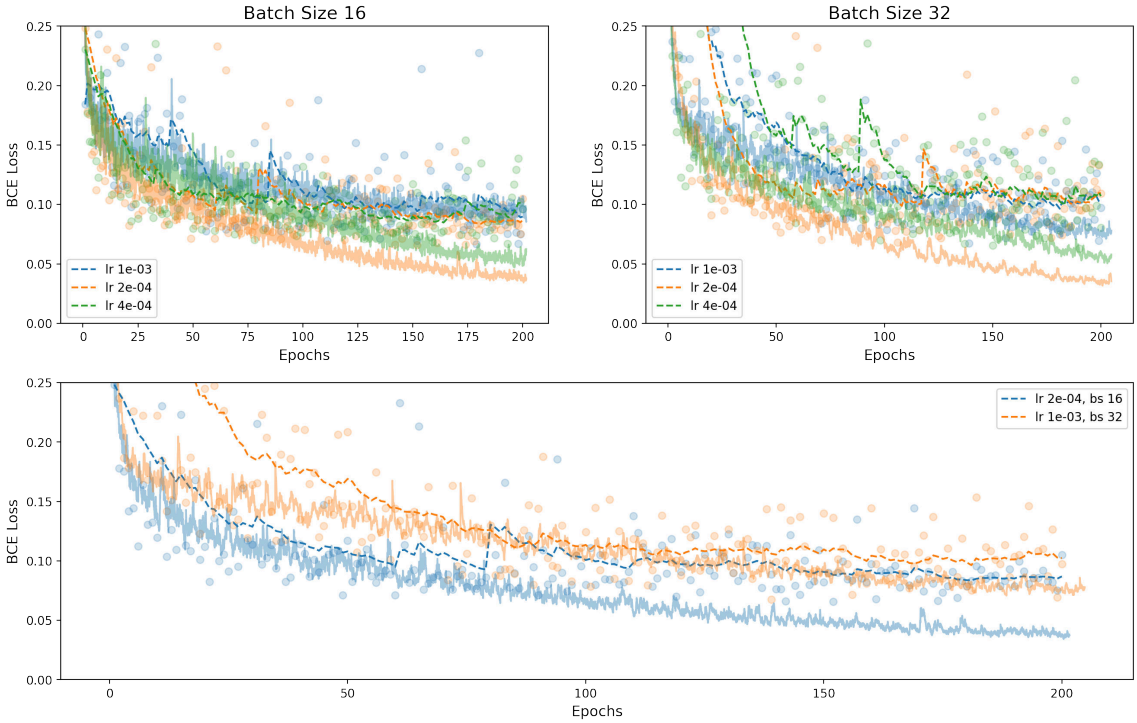


Figure 51: Artifact Localization learning rates and batch sizes comparison. **Top Left:** A comparison of three learning rates with batch size 16. **Top Right:** A comparison of three learning rates with batch size 32. **Bottom:** A comparison of the best combinations from the above graphics. For all runs the width unit was 64 and the Adam weight decay parameter 5e-3.

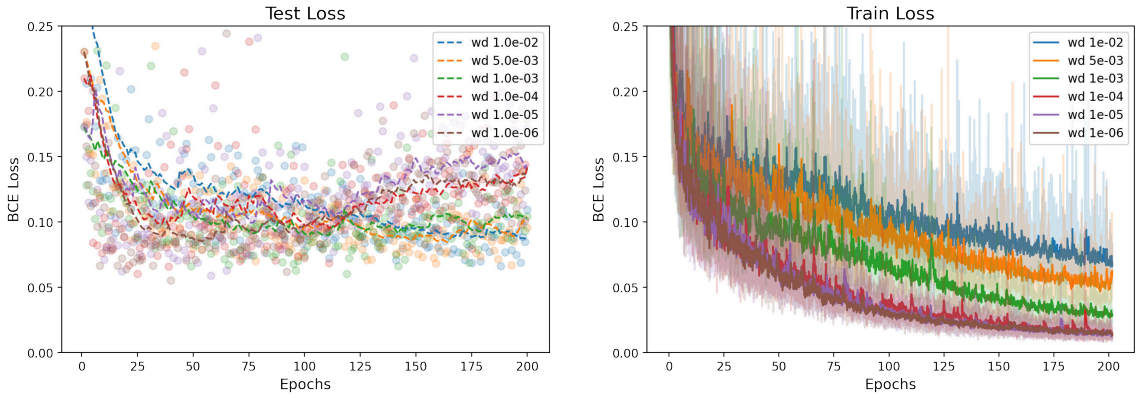


Figure 52: Artifact Localization weight decay comparison. Weight decay is adjusted via the weight decay parameter of PyTorch's Adam implementation. The slightly transparent circles (left) and plots (right) depict raw test and training data respectively. The solid lines in both graphics are exponential moving averages of the respective raw results. For all runs the batch size was 16, the width unit 64 and the learning rate 4e-4.

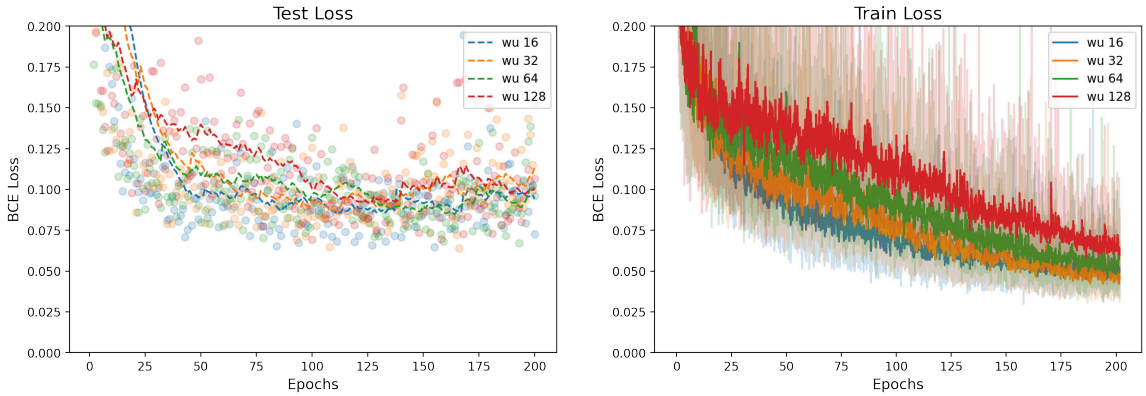


Figure 53: Artifact Localization width unit comparison. The slightly transparent circles (left) and plots (right) depict raw test and training data respectively. The solid lines in both graphics are exponential moving averages of the respective raw results. For all runs, the batch size was 16, the weight decay parameter 5e-3 and the learning rate 4e-4.

A.4.2 Density Inpainting

All runs in this section employ a learning rate scheduling where the learning rate is multiplied by 0.5 after the epochs 5, 10, 15 and 25. The Adam weight decay parameter is 0 and betas 0.9 and 0.999.

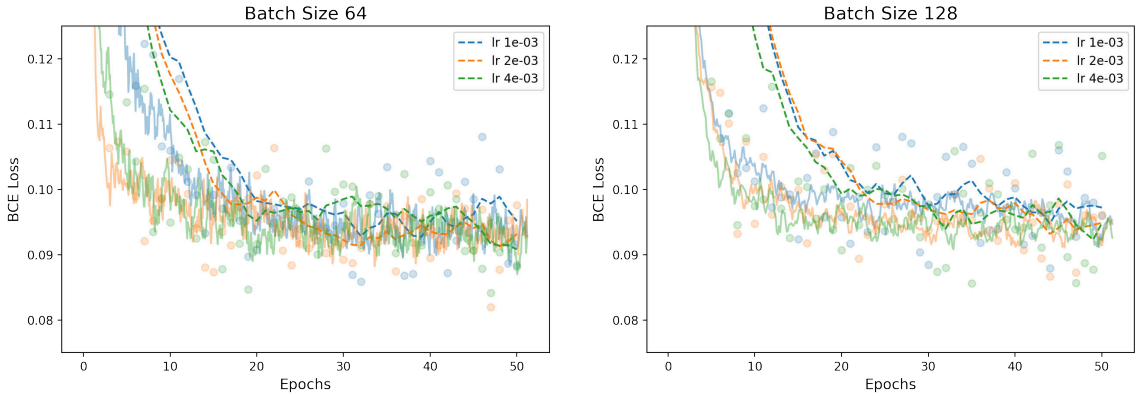


Figure 54: Density Inpainting batch sizes and learning rates comparison. **Left:** A comparison of three learning rates with batch size 64. **Right:** A comparison of three learning rates with batch size 128. The width unit was 8 for all runs.

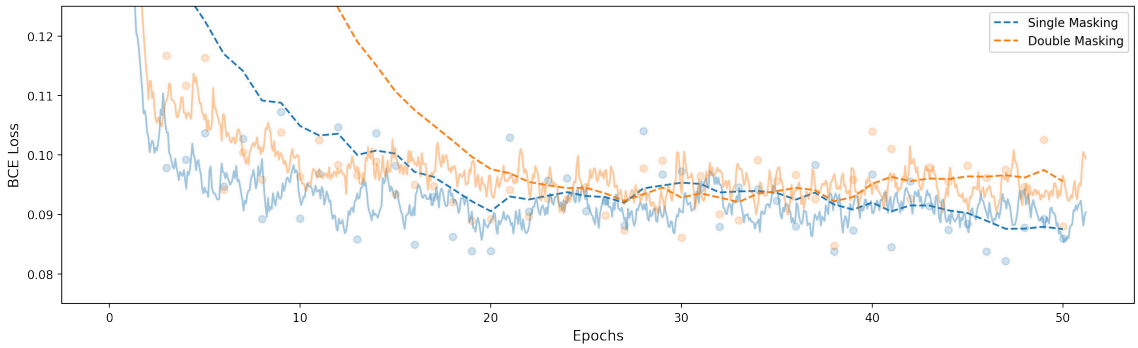


Figure 55: Density Inpainting double vs single masking comparison. Single masking means that the input crop during training is masked before the KDE is applied, which simulates the inference scenario where this information is missing in the first place. Double masking means that the masked KDE image is masked a second time right before it is fed to the network during training. For both runs the batch size was 64, the learning rate 2e-3 and the width unit 8.

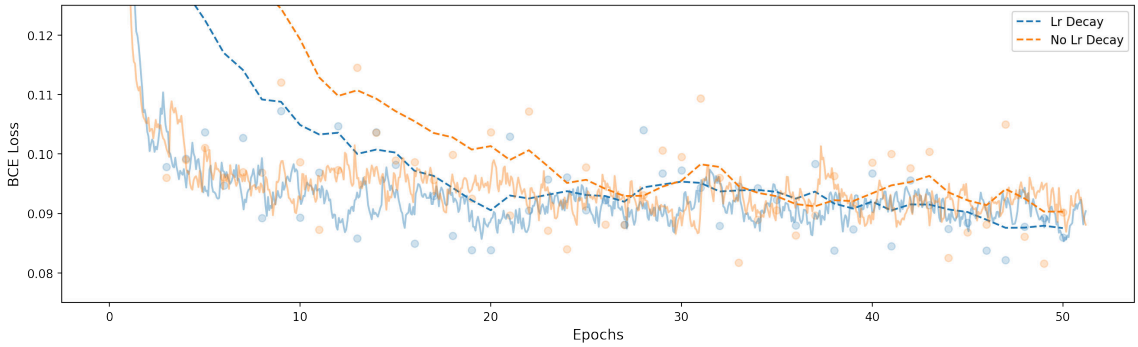


Figure 56: Density Inpainting learning rate decay comparison. Here, the regular learning rate scheduling was switched off. For both runs the batch size was 64, the initial learning rate 2e-3 and the width unit 8.

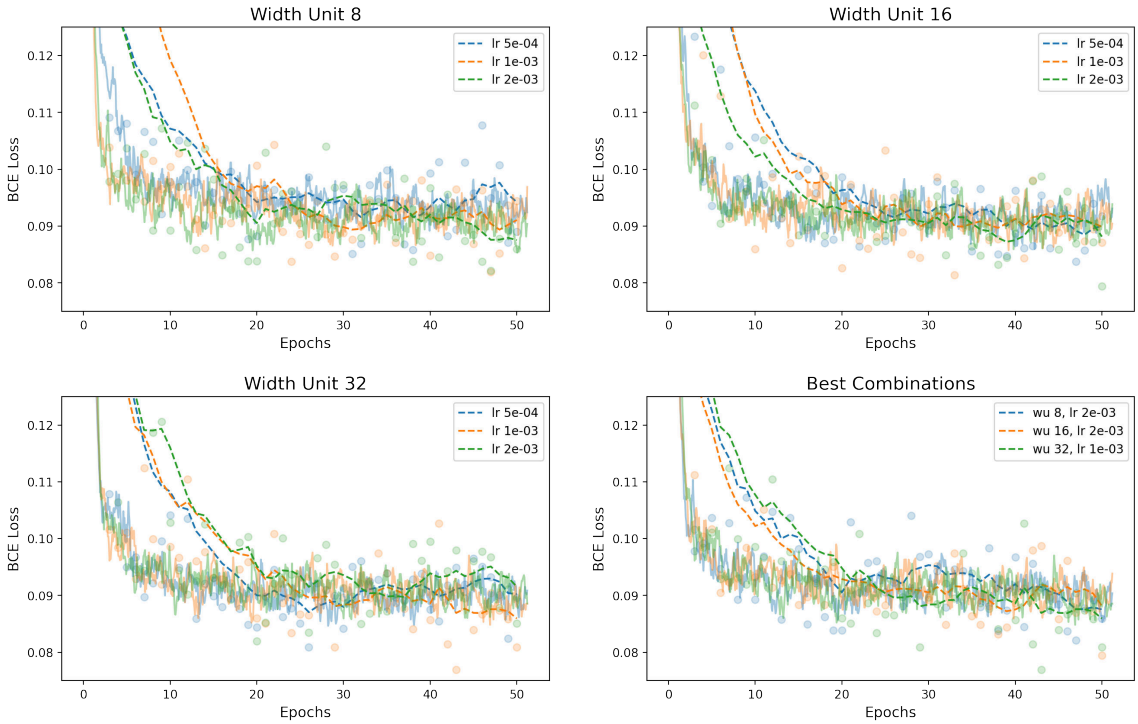


Figure 57: Density Inpainting width units and learning rates comparison. Different width units are compared with a range of learning rates. In the last graphic their best combinations are compared. Because the differences are so minor, but an increase in model width by a factor of 2 or 4, the final model uses width unit 8.

A.4.3 Shape Inpainting

For more information on the order parameter, see section 2.5.1. For all runs in this section the Adam weight decay parameter is 0 and the betas are 0.9 and 0.999. The maximum amount of cells for a crop is 192.

Training Collapse The cGlow model exerted a convergence behavior similar to mode collapse in GANs. When using low learning rates ($\leq$ 1e-3), especially in combination with low batch sizes ($\leq$ 16), all generated cells became small dots, even down to single pixels. When this happens, all entries of the Fourier matrix for the new cells are very small ($<$ 1e-2), much smaller than they should be. The use of column-wise data normalization for the Fourier matrices resulted in a classical collapse behavior to the mean cell size instead of extremely small cells. This collapse happens very early in training, right after the initial steep slope of the loss curve. Surprisingly, these were the hyperparameter settings that achieved the lowest NLL. When increasing the learning rate to a certain point the collapse behavior became more inconsistent. Results of experimenting with learning rate decay suggested the learning rate during the initial phase of training is important. When employing learning rate decay and an initially high learning rate, it is possible to decay the learning rate to previously unstable values without causing the collapse. It is unclear under what specific circumstances the collapse happens though. Adjusting the weight decay parameter of PyTorch's Adam implementation doesn't help here. Also, less severe cases of the collapse also happend with larger learning rates and batch sizes in later stages of the training, seemingly at random. Overall, larger batch sizes and higher learning rates seem to have a stabilizing effect in this regard.

This phenomenon unfortunately makes straight forward hyperparameter tuning impossible, because good results in terms of NLL and good model performance are not equivalent. Without further knowledge on what exactly is going on here, visual inspection of the results during the training process is needed to find a good set of hyperparemeters.

Order 2

Below are graphics of hyperparameter tuning without any form of learning rate decay. They illustrate the result in terms of NLL given different hyperparameter settings. Especially the batch size shows some unconventional behavior with respect to convergence speed.

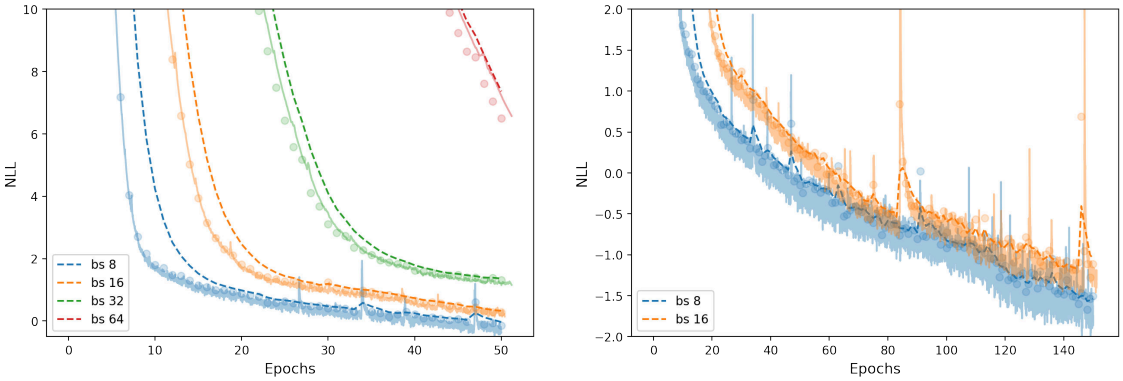


Figure 58: Shape Inpainting batch size comparison with different y-axis scaling. **Left:** A comparison of 4 batch sizes. **Right:** A comparison of the two best batch sizes with a longer training time of 10 epochs. While batch size 16 offeres a faster epoch time of 60s (vs 90s for batch size 8), batch size 8 stays ahead even in longer training scenarios. All runs employ a learning rate of 5e-4.

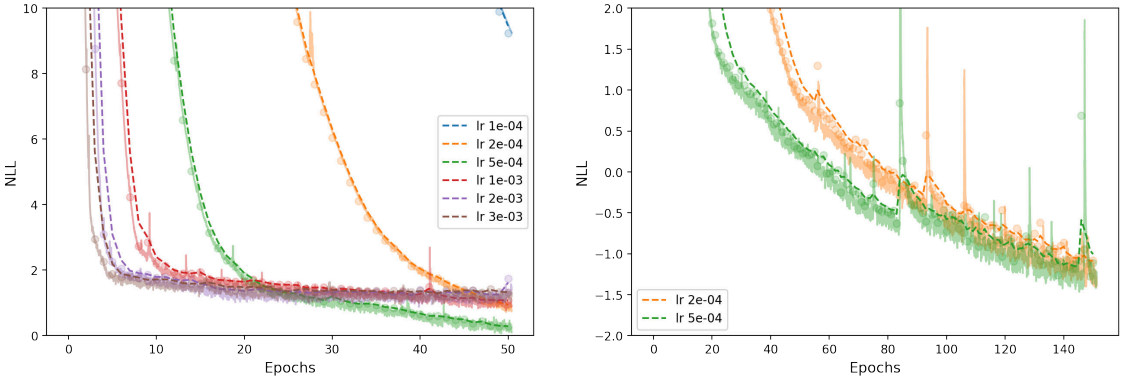


Figure 59: Shape Inpainting learning rate comparison. **Left:** A comparison of 6 different learning rates with different y-axis scaling. **Right:** A comparison of the two best learning rates. The slower learning rate of 2e-4 is able to catch up and is thus the preferred choice for longer training runs. All runs employ batch size 16.

Order 4

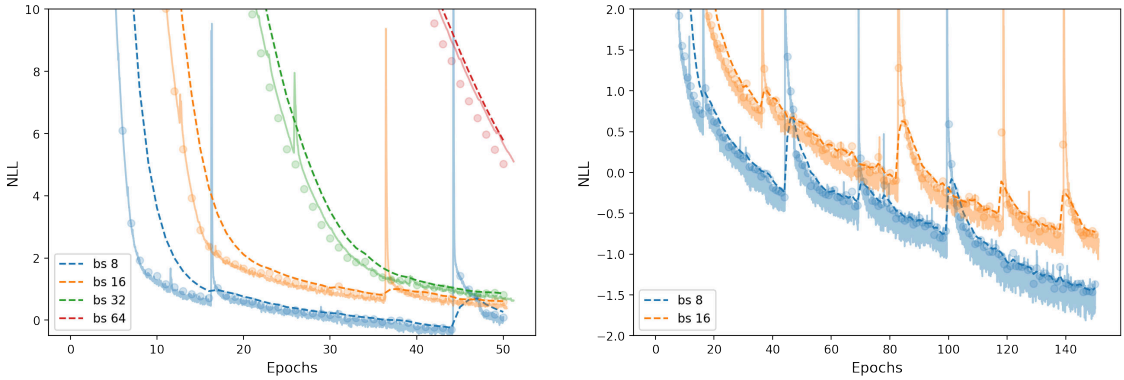


Figure 60: Shape Inpainting batch size comparison with different y-axis scaling. **Left:** A comparison of 4 batch sizes. **Right:** A comparison of the two best batch sizes with a longer training time of 10 epochs. Here batch size 8 is able to clearly outperform batch size 16. It still comes with a slower epoch time of 93s (vs 72s for batch size 16). All runs employ a learning rate of 5e-4.

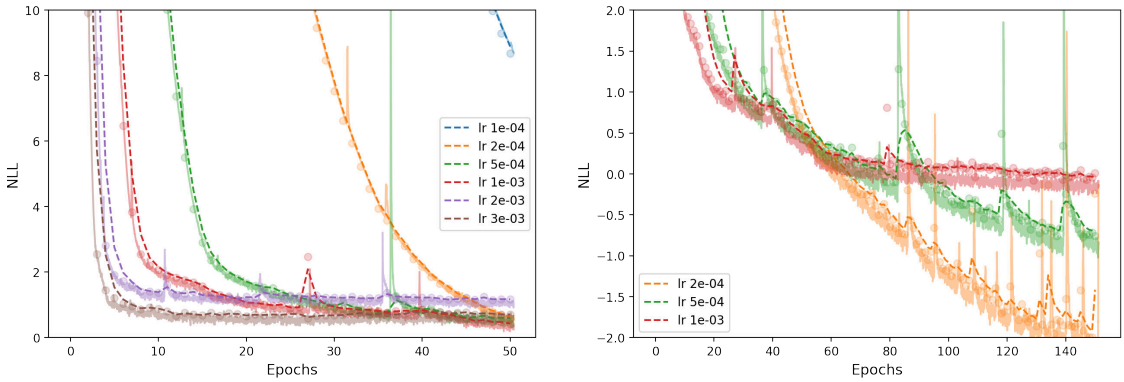


Figure 61: Shape Inpainting learning rate comparison. **Left:** A comparison of 6 different learning rates with different y-axis scaling. **Right:** A comparison of the three best learning rates from the left. Again, the slower 2e-4 is able to clearly outperform the faster learning rates. All runs emply batch size 16.

A.5 PyTorch and Hardware Setup

For the entirety of the project PyTorch version 1.9.0 and Cuda version 11.0 were used on Linux. All models were trained on a Quadro RTX 8000 with 45GB of memory. The CPU was an AMD EPYC 7742 64-core processor. Additionally, mixed precision training via the torch.cuda.amp package was used for the training of all models, except for the cGlow model in the Shape Inpainting step.

References

[1] Katrin Amunts, Claude Lepage, Louis Borgeat, Hartmut Mohlberg, Timo Dickscheid, Marc-Étienne Rousseau, Sebastian Bludau, Pierre-Louis Bazin, Lindsay B. Lewis, Ana-Maria Oros-Peusquens, Nadim J. Shah, Thomas Lippert, Karl Zilles, and Alan C. Evans. "BigBrain: an ultrahigh-resolution 3D human brain model". In: *Science (New York, N.Y.)* (2013). eprint: 340.6139.

[2] Amunts and Zilles. *Hirnsammlungen INM-1*. Structural and functional organisation of the brain (INM-1). 2021. URL: https://www.fz-juelich.de/inm/inm-1/DE/Home/_Schnellzugriff/Hirnsammlung/Hirnsammlung_node.html.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: (2017). arXiv: 1701.07875 [stat.ML].

[4] *BigBrain dataset release*. 2013. URL: https://bigbrain.loris.ca/main.php.

[5] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models". In: *CoRR* abs/2103.04922 (2021). arXiv: 2103.04922.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *CoRR* abs/1809.11096 (2018). arXiv: 1809.11096.

[7] Timo Dickscheid, Sarah Haas, Sebastian Bludau, Philipp Glock, Marcel Huysegoms, and Katrin Amunts. "Towards 3D reconstruction of neuronal cell distributions from histological human brain sections". In: *HPCC proceedings*. Braincomp 2019, Cetraro (Italy), 15 Jul 2019 - 19 Jul 2019. IOS press, July 15, 2019, pp. 1–17.

[8] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". In: (2015). arXiv: 1410.8516 [cs.LG].

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *CoRR* abs/1605.08803 (2016). arXiv: 1605.08803.

[10] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks". In: (2014). arXiv: 1406.2661 [stat.ML].

[12] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167.

[13] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. "Variational Autoencoder with Arbitrary Conditioning". In: (2019). arXiv: 1806.02382 [stat.ML].

[14] Abdul Jabbar, Xi Li, and Bourahla Omar. "A Survey on Generative Adversarial Networks: Variants, Applications, and Training". In: (2020). arXiv: 2006.05132 [cs.CV].

[15] Tim Kaiser. *Neural Inpainting: Repairing Artifacts in Histological Brain Sections with Deep Generative Models*. GitHub. 2021. URL: https://github.com/KaiserTim/Neural-Inpainting-Repairing-Histological-Artifacts.

[16] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *CoRR* abs/1812.04948 (2018). arXiv: 1812.04948.

[17] Diederik P. Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 10236–10245.

[18] Diederik P. Kingma, Tim Salimans, and Max Welling. "Improving Variational Inference with Inverse Autoregressive Flow". In: *CoRR* abs/1606.04934 (2016). arXiv: 1606.04934.

[19] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[20] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. arXiv: http://arxiv.org/abs/1312.6114v10 [stat.ML].

[21] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (Nov. 2020), pp. 3964–3979.

[22] F. Kuhl and C. Giardina. "Elliptic Fourier features of a closed contour". In: *Computer Graphics and Image Processing* 18 (1982), pp. 236–258.

[23] Yann LeCun and Corinna Cortes. *MNIST handwritten digit database*. 2010. URL: http://yann.lecun.com/exdb/mnist/.

[24] You Lu and Bert Huang. "Structured Output Learning with Conditional Generative Flows". In: (2020). arXiv: 1905.13288 [cs.LG].

[25] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. "Unrolled Generative Adversarial Networks". In: *CoRR* abs/1611.02163 (2016). arXiv: 1611.02163.

[26] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. "Spectral Normalization for Generative Adversarial Networks". In: *CoRR* abs/1802.05957 (2018). arXiv: 1802.05957.

[27] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[28] Joseph Rocca. *Understanding Generative Adversarial Networks (GANs)*. towards data science. 2019. URL: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.

[29] Joseph Rocca. *Understanding Variational Autoencoders (VAEs)*. towards data science. 2019. URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597.

[31] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. "Temporal Graph Networks for Deep Learning on Dynamic Graphs". In: *CoRR* abs/2006.10637 (2020). arXiv: 2006.10637.

[32] Lars Ruthotto and Eldad Haber. "An Introduction to Deep Generative Modeling". In: *CoRR* abs/2103.05180 (2021). arXiv: 2103.05180.

[33] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved Techniques for Training GANs". In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498.

[34] Grant Sanderson. *But what is a Fourier series? From heat flow to drawing with circles | DE4*. Youtube. 2019. URL: https://www.youtube.com/watch?v=r6sGWTCMz2k.

[35] Edgar Schönfeld, Bernt Schiele, and Anna Khoreva. "A U-Net Based Discriminator for Generative Adversarial Networks". In: *CoRR* abs/2002.12655 (2020). arXiv: 2002.12655.

[36] Eric Upschulte, Stefan Harmeling, Katrin Amunts, and Timo Dickscheid. "Contour Proposal Networks for Biomedical Instance Segmentation". In: *arXiv:2104.03393 [cs]* (Apr. 2021). arXiv: 2104.03393. (Visited on 06/28/2021).

[37] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". In: *CoRR* abs/1711.11585 (2017). arXiv: 1711.11585.

[38] Wei Xiong, Jiahui Yu, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. "Foreground-aware Image Inpainting". In: *CoRR* abs/1901.05945 (2019). arXiv: 1901.05945.

[39] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. "Free-Form Image Inpainting with Gated Convolution". In: *CoRR* abs/1806.03589 (2018). arXiv: 1806.03589.

[40] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. "Generative Image Inpainting with Contextual Attention". In: *CoRR* abs/1801.07892 (2018). arXiv: 1801.07892.

List of Figures

List of Tables