



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

# Deep Generative Modelling of Microscopy Image Data

Anastasia Stamatouli





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

# Deep Generative Modelling of Microscopy Image Data

## Generative Modelle für Mikroskopiebilder durch Deep Learning

Author:	Anastasia Stamatouli
Supervisor:	Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Dietrich
Submission Date:	30/03/2022



I confirm that this master's thesis in data engineering and analytics is my own work and I have documented all sources and material used.

Munich, 30/03/2022

Anastasia Stamatouli

## Acknowledgments

I would like to thank Dr. Felix Dietrich for giving me the opportunity to write my Master Thesis at the Chair of Scientific Computing in Computer Science at Technical University Munich. Dr. Dietrich has been very helpful, responsive and understanding during this time. Additionally, I owe the biggest thank to my family and Wiktor for their endless support during my long studying journey. Last but not least, TUM has greatly helped me in shaping my research and academic interests and developing myself into a well rounded engineer. Munich will always remain an important part of my professional and personal life.



# Abstract

Collection of big datasets is extremely challenging for many fields (e.g healthcare and biology) and goals have to be achieved with limited amount of data. Data Augmentation (DA) techniques can help us mitigate this issue. The recent rise of Deep Generative Models has made them very attractive models to perform DA. In this Master Thesis we explore the field of Deep Generative Modelling of Microscopy Image Data that depict the Arbuscular Mycorrhiza Fungi (AMF). We develop a Variational Autoencoder (VAE) in order to generate verisimilar synthetic Microscopy Image Data which can later be used to increment the size of AMF datasets and allow them to be utilized further for classification or segmentation purposes. We provide a full generation pipeline, where the original dataset is being preprocessed, split into smaller parts, passed to the model for training and used afterwards for generating new image data. The VAE has a new architecture, developed from scratch and inspired by Residual Network (ResNet). Additionally, a novel training approach for the ResNet based VAE has been developed and explored which is inspired by Progressive Generative Adversarial Network (ProGAN). The results are encouraging and motivate future research in the field of Deep Generative Modelling.

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 State of the Art</b>	<b>3</b>
2.1 Arbuscular Mycorrhiza Fungi . . . . .	3
2.2 Generative Modelling versus Discriminative Modelling . . . . .	3
2.3 Autoencoders . . . . .	7
2.4 Variational Autoencoders . . . . .	8
2.5 Generative Adversarial Networks (GAN) . . . . .	9
2.6 Progressive Generative Adversarial Network (ProGAN) . . . . .	12
2.7 Residual Network (ResNet) . . . . .	15
<b>3 Deep Generative Modelling of Microscopy Image Data</b>	<b>19</b>
3.1 Dataset . . . . .	19
3.2 Image Preprocessing . . . . .	19
3.3 ResNet Based Variational Autoencoder . . . . .	22
3.4 Progressive Variational Autoencoder (proVAE) . . . . .	24
3.5 Evaluation metrics . . . . .	26
3.6 Experiments & Results . . . . .	27
3.6.1 Results ResNet-18 Variational Autoencoder . . . . .	28
3.6.2 Results Progressive ResNet Variational Autoencoder (proVAE) .	29
3.7 Implementation Details . . . . .	30
<b>4 Conclusion</b>	<b>36</b>
<b>List of Figures</b>	<b>38</b>
<b>List of Tables</b>	<b>40</b>
<b>Bibliography</b>	<b>42</b>

# 1 Introduction

Over the last decade, *Deep Neural Networks (DNN)* have produced unprecedented performance on a number of tasks, given sufficient data. *DNNs* have been demonstrated in a variety of domains spanning from image classification [18] to machine translation [45]. In all above mentioned cases very large datasets have been utilized.

However, in reality the collection of big datasets is extremely challenging for many fields (e.g healthcare and biology) and goals have to be achieved with limited amount of data. For example, collecting *Microscopy Images* which depict the *Arbuscular Mycorrhiza Fungi (AMF)* is a complex process which is time consuming (staining arbuscular-mycorrhizal fungal colonizations in root tissues and prepare the AMF samples for the microscope) but also involves some risk in case the staining is not successful. Another example is healthcare where practitioners have to deal most of the time with small patient cohorts (low sample size) along with very high dimensional data (e.g 3D neuroimaging data). Additionally, small datasets make *DNNs* and statistical methods unreliable. A way to address those issues is to perform data augmentation (DA) [18]. In a nutshell, DA is the art of increasing the size of a given data set by creating synthetic data. For instance, the easiest way to do this on images is to apply simple transformations such as the addition of Gaussian noise, cropping or padding, rotation [18]. While such augmentation techniques have been very useful, they remain strongly data dependent and limited. The recent rise in performance of generative models such as *Generative Adversarial Networks (GAN)* [8] or *Variational Autoencoders (VAE)* [16] has made them very attractive models to perform DA (Fig. 1.1). *GANs* have already been explored towards this direction [1] and demonstrated promising results.

In this Master Thesis we explore further the field of *Deep Generative Modelling* of Microscopy Image Data. We develop a *Variational Autoencoder* and explore a novel training method in order to generate verisimilar synthetic *Microscopy Image Data* that depict the *Arbuscular Mycorrhiza Fungi (AMF)*. The synthetic image data can later on be used to increment the size of AMF datasets and allow them to be utilized further for classification or segmentation purposes.

We initially start by providing the biological background of the dataset and the technical foundations needed for the understanding of a *Variational Autoencoder* (Ch. 2). More specifically, in Sec. 2.1 we explain what is the *Arbuscular Mycorrhiza Fungi (AMF)* and how it is beneficial for the plants. Afterwards, in Sec. 2.2 we analyze the

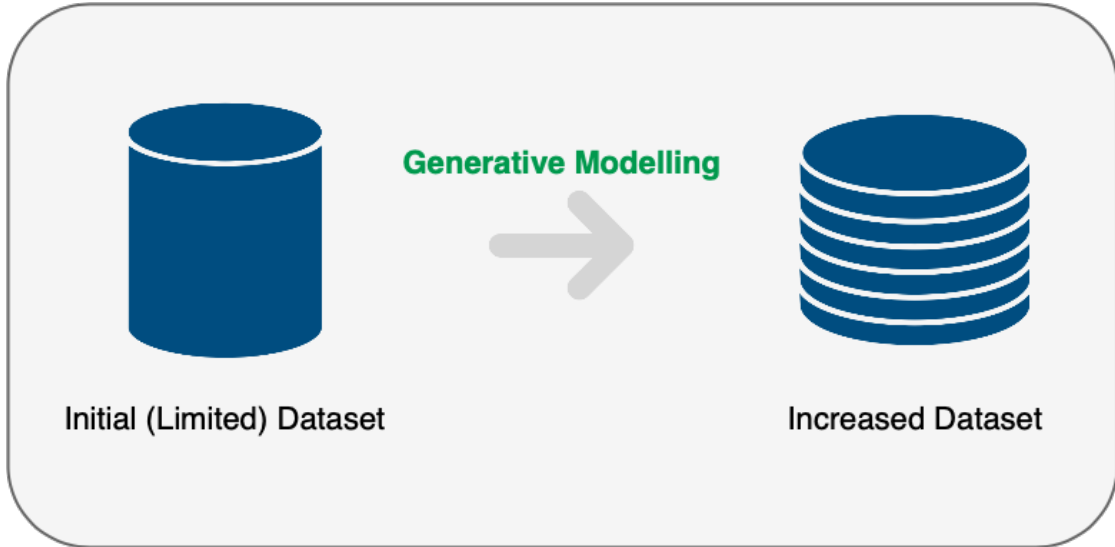


Figure 1.1: Data Augmentation using Generative Modelling.

difference between *Generative* and *Disriminative* modeling and in Sec. 2.3 we give an introduction to *Autoencoders*. Next, in Sec. 2.4 and Sec. 2.5 we dive deeper into the two most popular generative models, *Variational Autoencoders (VAE)* and *Generative Adversarial Networks (GAN)* respectively. In Sec. 2.6 and Sec. 2.7 we present the two main methods that our contributions have been inspired from: the *Progressive Generative Network (ProGAN)* and *Residual Network (ResNet)* respectively. Following the theoretical foundations we outline in detail our own developed method in Ch. 3. First, we provide information about the dataset (Sec. 3.1) and the techniques used for preprocessing the data (Sec. 3.2). Thereafter, in Sec. 3.3 we analyze in depth our first contribution which is a new architecture for *Variational Autoencoder* inspired by ResNet and in Sec. 3.4 we present our second contribution which is the development and exploration of a novel training approach for the model. Consequently, in Sec. 3.5 we present the loss function that we used for training the model and in Sec. 3.6 we present the results for our two contributions. Last but not least, in Sec. 3.7 we provide the implementation details about setting up the required infrastructure for running and testing the models. Finally, in Ch. 4, we summarize our findings and discuss potential future directions.

## 2 State of the Art

The goal of this chapter is to provide the biological background and technical foundations needed for the understanding of the Deep Generative Model that has been developed for the purpose of this Master Thesis.

### 2.1 Arbuscular Mycorrhiza Fungi

Arbuscular Mycorrhiza Fungi (AMF) is the most common symbiotic association of terrestrial plants with microbes [40]. Spores that are present in soil grow, infect the root system, and form arbuscule structures inside the cells [40] (Fig. 2.1 ). Arbuscules are the sites where the plant and the fungal exchange nutrients. The presence of a vast mycorrhizal network around the root system is another feature of this relationship [40]. AMF can be found in almost all natural settings and provide a variety of critical ecological functions, including improved plant nutrition, stress resistance and tolerance, soil structure, and fertility [5]. AMF also interacts with most crop plants, including cereals, vegetables, and fruit trees, therefore their potential use in sustainable agriculture is gaining traction [5]. Microscopy Images that depict AMF are visualized in Fig. 3.1.

### 2.2 Generative Modelling versus Discriminative Modelling

A *Generative Model* is a statistical model that captures the joint probability distribution  $P(X, Y)$  on given observable variable  $X$  and target variable  $Y$  or simply  $P(X)$  if there are no labels present [14]. *Generative Models* can generate new data instances (Fig. 2.2) and model the data distribution of the original space. Typical Generative Models are: *Variational Autoencoders* (explained in Sec. 2.4), *Generative Adversarial Networks* (explained in Sec. 2.5), *Normalizing Flows* [36].

*Discriminative Models*, on the other hand, capture the conditional probability  $p(Y|X)$  or map the given unobserved variable (target)  $x$  to a class label  $y$  dependent on the observed variables (training samples) [25]. Discriminative models try to define data space boundaries. Typical discriminative models are: *Logistic Regression* [24] and *Decision Trees* [37].

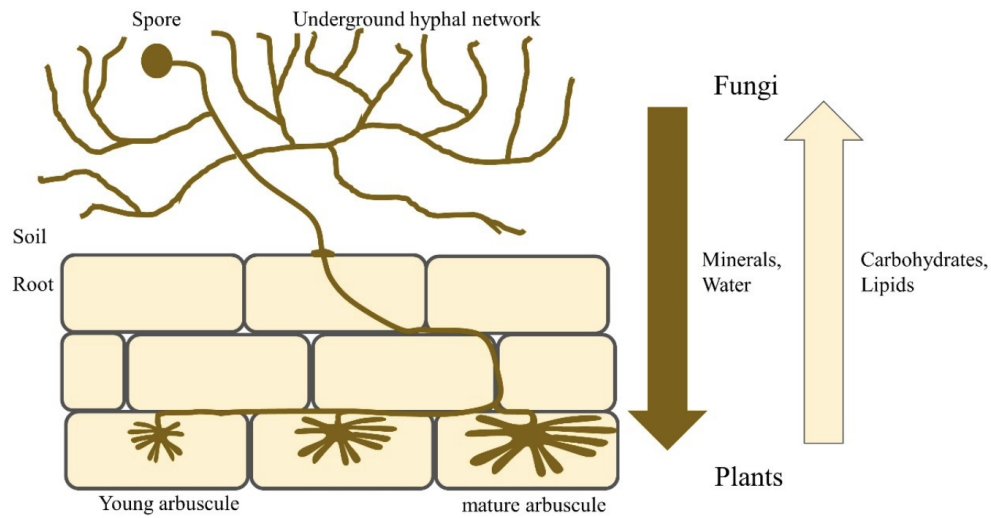


Figure 2.1: Schematic representation of AMF establishment inside a host and the known exchange between the two partners. Figure credit: Florence Sessoms [40]

For example, in the Figure 2.3 the Discriminative Model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space. Even if it doesn't know exactly where the data is in the space, it can tell the difference between 0s and 1s if the line is drawn correctly. In contrast, the Generative Model has to model the data distribution across the entire space and if successful generates digits that are near to the real ones, making it hard in this way to distinguish between them [9].

For the purpose of this Master Thesis, we will concentrate on *Generative Modelling* and particularly on *Deep Generative Modelling*. As a result of the rise of *Deep Learning*, a new family of Generative Models has emerged from the intersection of Generative Models and Deep Learning, known as *Deep Generative Models (DGMs)*. DGMs have advanced greatly in the past few years [3] [27]. This is due to architectural advances as well as computational development that allows them to be trained at a bigger scale in terms of both model depth and data size [34]. Their applications range from data augmentation [4] to WaveNet which is a generative model for raw audio [27]. There are two main types of Generative Models: *Likelihood Based Models* and *Implicit Generative Models* [34]. The Likelihood Based Models include *Variational Autoencoders* [43] [10], *Flow Based* [36] and *Autoregressive Models* [44]. The Implicit Generative Models include models like *Generative Adversarial Networks (GANs)*[8].



Figure 2.2: New data instances generated by VQ-VAE 2. The model generates realistic looking faces that respect long-range dependencies such as matching eye colour or symmetric facial features, while covering lower density modes of the input dataset (e.g., green hair). Picture taken from [34].

GANs [8] optimize a min-max objective. The *Generator*, which is a neural network, produces images by transforming random noise input into a meaningful output. The *Discriminator*, also a neural network, classifies the samples produced by the Generator as real or fake. Larger scale GAN models can now generate high-quality and high-

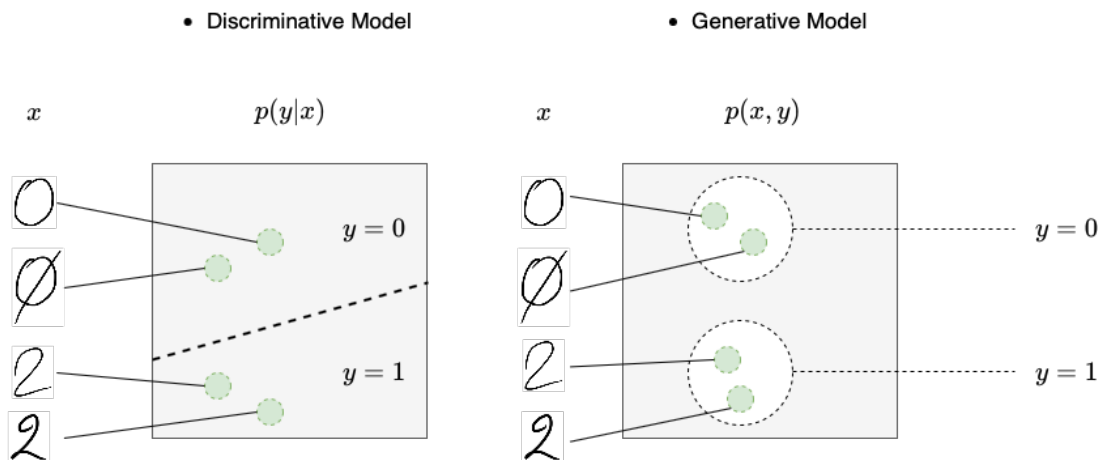


Figure 2.3: Discriminative vs Generative Model of handwritten digits.

resolution images [3]. However, it is well known that the variety of the original distribution is not fully captured by those models. Furthermore, GANs are hard to evaluate, and currently there is not a satisfactory evaluation metric on a test set for measuring overfitting [34]. For model comparison and selection, researchers have used image samples or proxy measures of image quality such as *Inception Score (IS)* [38] and *Fréchet Inception Distance (FID)* [13] [34].

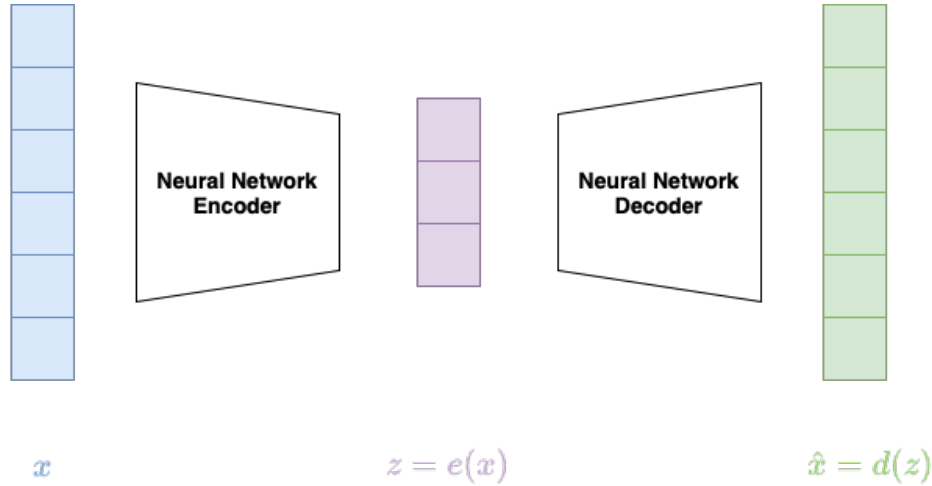
In contrast, *Likelihood Based Models* optimize negative log-likelihood (NLL) of the training data. This objective allows model comparison and measuring generalization in previously unseen data. Moreover, since the probability that the model assigns to all examples in the training set is maximized, Likelihood Based Models cover all modes of the data, and do not suffer from the problems of mode collapse and lack of diversity seen in GANs [34]. However, maximizing likelihood directly in the pixel space can be a challenging task. NLL in pixel space is not always a useful indicator of sample quality [42], and it can't be used to compare different model classes reliably [34]. Some of these issues are mitigated by introducing inductive biases such as multi-scale [42] or by modeling the dominant bit planes in an image [17] [34].

All in all, there's not a perfect generative model as each of these models offer several trade-offs such as sample quality, diversity, speed, etc. and the choice of it depends on the application.



## 2.3 Autoencoders

An *Autoencoder (AE)* is an unsupervised learning technique that uses neural networks to learn a non-linear representation (encoding) for a set of data, typically for dimensionality reduction or feature learning, by training the network to ignore insignificant data (noise).



$$loss = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$

Figure 2.4: Illustration of an Autoencoder with its loss function.

The AE consists of two parts: an *Encoder Network*,  $z = e(x)$  and a *Decoder Network*  $\hat{x} = d(z)$  (Fig. 2.4). The input data is defined by  $x$  and  $y$  is the latent vector representation of  $x$ ,  $\hat{x}$  is the reconstruction of the input data  $x$  from the latent space. The *Encoder*  $e(x)$  and the *Decoder*  $d(x)$  are both neural networks. The entire model can be described by the equation:

$$\hat{x} = d(e(x)) \quad (2.1)$$

We expect the Decoder to be able to accurately reconstruct the raw data from the latent vector representation. If the model is able to learn such reconstruction then we assume that the latent space represents the raw data quite well. In order to enforce this objective, we train the model end to end with a reconstruction loss between  $x$  and  $\hat{x}$

like the  $l_2$  loss  $\|x - \hat{x}\|_2^2$ . It's also important to mention here that latent space should have lower dimension than the input space (raw data). The reason for that is that the model should be able to find the most representative features of the data or in other words a lossy compression of the data.

In order to ensure that the reconstructions from latent space are meaningful we need to ensure a regular latent space with the following two properties: *continuity* and *completeness*. *Continuity* means that two close points in the latent space should not give two completely different contents once decoded. *Completeness* means that for a chosen distribution a point sampled from the latent space should give meaningful content once decoded.

However, in the AEs the dimensionality reduction performed on the input space has no supervision, meaning is not regulated by a loss or a term in the general loss function (lack of regularity). The AE is solely trained to encode and decode with the smallest loss possible, no matter how the latent space is organised. So, it is pretty difficult (if not impossible) to ensure, a priori, that the encoder will organize the latent space in a smart way compatible with the needed regularity. *Variational Autoencoders* fill the gap of regularity and they are able to use the decoder as a generator too.

## 2.4 Variational Autoencoders

A *Variational Autoencoder* (VAE) can be defined as being an *Autoencoder* (Sec. 2.3) whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable the generative process. Similarly to an *Autoencoder*, the VAE also consists of two parts: an *Encoder* and *Decoder network*.

Let  $x \in \mathbb{R}^D$  be our input data. The input data can be for example color images. VAE aims at maximizing the likelihood of a given parametric model  $P_\theta, \theta \in \Theta$  [4]. It is assumed, that there are latent variables  $z$  which live in a lower continuous dimension space, the so-called *latent space*. The main advantage of making  $z$  continuous is that is easier to sample with reparametrization from continuous distributions. The marginal distribution of the data can be now written as:

$$p_\theta(x) = \int_Z p_\theta(x|z)q(z) dz \quad (2.2)$$

where  $p_\theta(x|z)$  is the *Decoder* that models the parameters of a parametrized distribution (e.g Gaussian, Bernoulli) and  $q$  is the prior distribution over the latent space which acts as a regulating factor.

By applying the Bayes Theorem  $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$  the posterior distribution  $p_\theta(z|x)$  is formulated as follows:

$$p_{\theta}(z|x) = \frac{p_{\theta}(x|z)q(z)}{\int_{\mathcal{Z}} p_{\theta}(x|z)q(z) dz} \quad (2.3)$$

However, the calculation of the posterior becomes very difficult because of the intractable integral of Eq. 2.2. This integral requires exponential time to compute as it needs to be evaluated over all configurations of latent variables. We therefore need to approximate the posterior distribution by using techniques like *Variational Inference* [2]. Therefore, we introduce a variational distribution  $q_{\phi}(z|x)$  which aims at approximating the true posterior distribution  $p_{\theta}(z|x)$  [4]. In order to know how well the variational distribution  $q_{\phi}(z|x)$  approximates the true posterior  $p_{\theta}(x)$  we can compute the *Kullback - Leibler divergence* as follows [11]:

$$KL(q_{\phi}(z|x)||p_{\theta}(z|x)) = \int q_{\phi}(z|x) \log \frac{q_{\phi}(z|x)}{p_{\theta}(z|x)} \quad (2.4)$$

Now our objective function can be formulated as follows [11]:

$$\log p_{\theta}(x) = \underbrace{E_{z \sim q_{\phi}} \left[ \log \frac{p_{\theta}(x,z)}{q_{\phi}(z|x)} \right]}_{L(\theta,q)} + \underbrace{KL(q_{\phi}(z|x)||p_{\theta}(z|x))}_{\geq 0} \quad (2.5)$$

Since *Kullback - Leibler divergence* is nonnegative, then  $L(\theta, q)$  is a lower bound on  $\log p_{\theta}(x)$ . The expression  $\log p_{\theta}(x)$  is often called *Evidence*, so we call  $L(\theta, q)$  *Evidence Lower Bound (ELBO)* [11].

Therefore, we can equivalently rewrite the ELBO as:

$$L(\theta, q) = -KL(q_{\phi}(z|x)||p_{\theta}(z|x)) + \log p_{\theta}(x) \quad (2.6)$$

Finally, in order to sample from the latent space which is modelled by the variational distribution  $q_{\phi}(z|x)$ , we use the so-called *Reparametrization Trick*. The trick will allow us to compute gradients wrt  $\phi$  [11]. For example, let  $q_{\phi}(z|x) = N(z|\mu, RR^T)$  be a multivariate normal distribution whose parameters  $\mu_{\phi}$  and  $\Sigma_{\phi}$  are modelled by the *Encoder Network*. The sampling from  $q_{\phi}(z|x)$  is equivalent to [11]:

1. Drawing a sample  $\epsilon \sim N(0, I)$ .
2. Obtaining  $z = T(\epsilon, \phi = \mu, R) = R\epsilon + \mu$

## 2.5 Generative Adversarial Networks (GAN)

A *Generative Adversarial Network (GAN)* is a Deep Generative model introduced by Ian Goodfellow in 2014 [8].

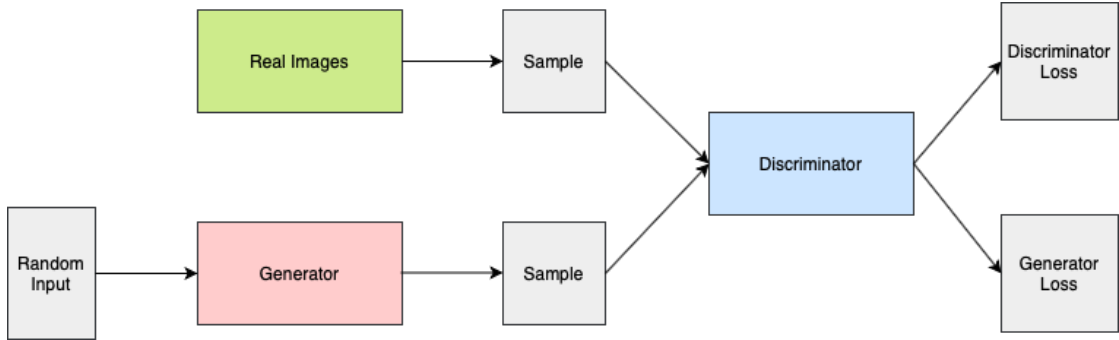


Figure 2.5: Architecture of Generative Adversarial Network.

The network consists of two models. A *Generative* model  $G$  that learns the data distribution and a *Discriminative* model  $D$  that estimates the probability that a sample came from the real domain (data distribution) rather than  $G$  (a fake sample) (Fig. 2.5)

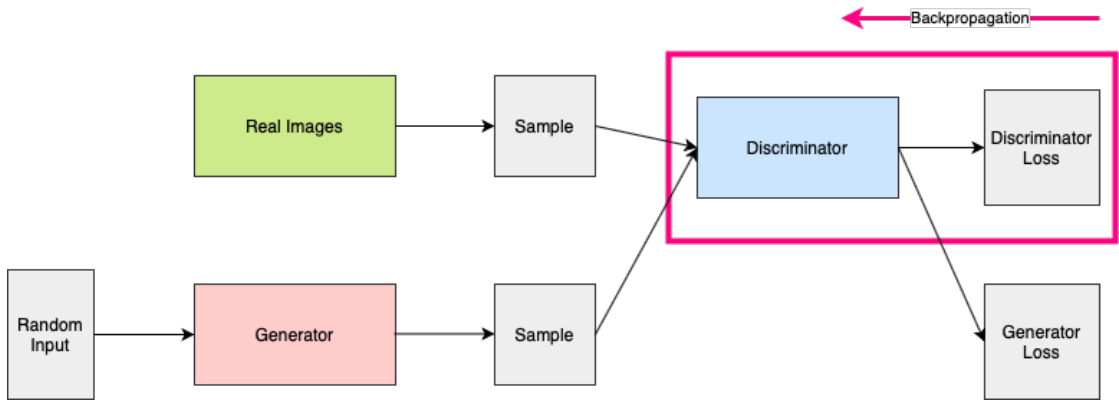


Figure 2.6: Backpropagation in Discriminator Training.

The *Discriminator*  $D$  is a neural network with parameters  $\theta_D$  which performs classification [7]. The Discriminator's training data comes from two sources: real and fake data instances (Fig. 2.5). The Discriminator uses the real instances as positive examples during training while fake data instances are created by the Generator and are used as negative examples during training. The Discriminator connects to two loss functions (Fig. 2.5). During Discriminator training, the Discriminator ignores the Generator loss and just uses the Discriminator loss (Fig. 2.6). The Discriminator classifies both real data and fake data from the Generator. The Discriminator loss penalizes the Discriminator for misclassifying a real instance as fake or a fake instance as real. Finally, it updates its weights through backpropagation from the Discriminator

loss through the Discriminator network (Fig. 2.6).

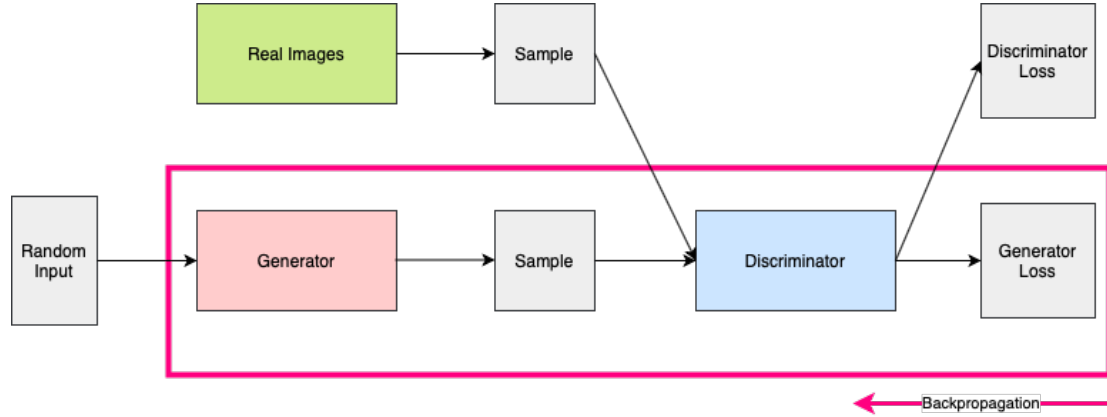


Figure 2.7: Backpropagation in Generator Training.

The *Generator*  $G$  is a neural network with parameters  $\theta_G$  [7].  $G$  takes random noise (e.g uniform) as its input and transforms this noise into a meaningful output. The *Generator*  $G$  is not directly connected to the loss that's why learns how to create fake data by incorporating feedback from the Discriminator. The Generator feeds into the Discriminator, and the Discriminator produces the output we're trying to affect. The Generator loss penalizes the Generator for producing a sample that the Discriminator network classified as fake. Afterwards, we backpropagate through both the Discriminator and Generator to obtain gradients but we use gradients to change only the *Generator* weights (Fig. 2.7).

In other words, *Discriminator* and *Generator* play the following two-player minimax game with value function  $V(G, D)$  [8]:

$$\min_G \max_D V(D, G) = E_x[\log D(x)] + E_z[\log(1 - D(G(z)))] \quad (2.7)$$

where  $D(x)$  is the Discriminator's estimate of the probability that real data instance  $x$  is real,  $E_x$  is the expected value over all real data instances,  $G(z)$  is the Generator's output when given noise  $z$ ,  $D(G(z))$  is the Discriminator's estimate of the probability that a fake instance is real,  $E_z$  is the expected value over all random inputs to the Generator. The Generator can't directly affect the  $\log(D(x))$  term in the function, so, for the Generator, minimizing the loss is equivalent to minimizing  $\log(1 - D(G(z)))$ .

While GANs are a breakthrough in generative modelling with really good results, in practice is often quite difficult to train. This is largely due to the famous problem which is called *mode collapse*. *Mode collapse* can occur when the discriminator essentially "wins" the game, and the training gradients for the Generator become less and less useful.

This can happen relatively quickly during training, and when it does, the Generator starts outputting nearly the same sample every time, meaning it stops getting better. In the years since, the research community has come up with many ways to make training more reliable, with one of the most successful being the famous *Progressive GAN* [15] (2.6).

## 2.6 Progressive Generative Adversarial Network (ProGAN)

*Progressive Growing of GANs* is a training methodology for *Generative Adversarial Networks* introduced by NVIDIA [15]. The key idea is to increase the depth of *Generator* and *Discriminator* progressively and is inspired by the work in [19], which uses multiple discriminators that operate on different spatial resolutions. The training starts with low-resolution images (4x4), and then progressively the resolution is increased by adding layers to the networks as visualized in Fig. 2.8. The benefit of the progressive growing comes from the fact that by increasing gradually the resolution, the networks are asked to learn a much simpler piece of the overall problem.

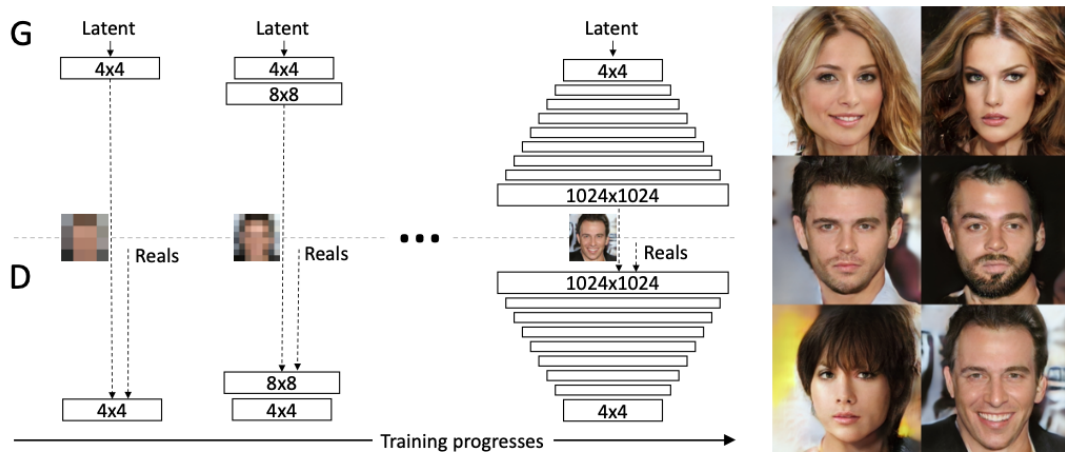


Figure 2.8: Training setup in ProGAN. On the right six example images are generated using progressive growing at  $1024 \times 1024$ . Picture taken from [15]

More specifically, the authors first shrunk their training images to the starting resolution (4x4). Then, they create a Generator with just a few layers to synthesize images at this low resolution, and a corresponding Discriminator of mirrored architecture. These networks are small, therefore training is relatively quick and focuses only on high level structures which are visible in the images [15]. When the first layers are trained

completely, another layer is added to the Generator and Discriminator respectively. After the addition of the new layer the learned resolution is now doubled to  $8 \times 8$ . The trained weights in the earlier layers are kept and the new layer is faded in gradually in order to stabilize the transition [15].

To prevent shocks in the network from the sudden addition of a new layer, the new layer is linearly *faded in*. This fading in is controlled by a parameter  $\alpha$ , which is linearly interpolated from 0 to 1 over many training iterations [15] (Fig. 3.7).

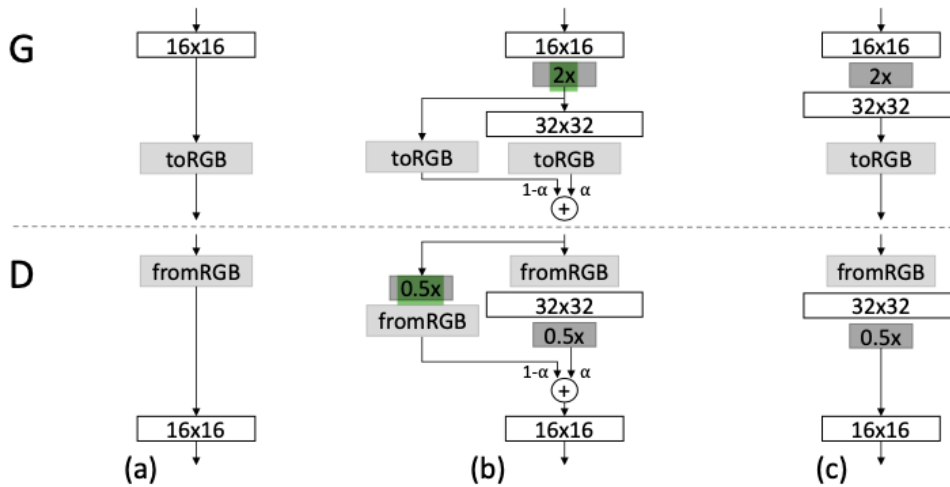


Figure 2.9: Illustration of Fading in. In this figure we observe the transition from  $16 \times 16$  images (a) to  $32 \times 32$  images (c). Here  $2x$  and  $0.5x$  refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The `toRGB` represents a layer that projects feature vectors to RGB colors and `fromRGB` does the reverse; both use  $1 \times 1$  convolutions. When training the discriminator, the real images that are fed in are downscaled to match the current resolution of the network. During a resolution transition, interpolation is happening between two resolutions of the real images, similarly to how the generator output combines two resolutions. Picture taken from [15].

Additionally, instead of using batch normalization, the authors introduced *Pixel Normalization*. *Pixel Normalization* layer has no trainable weights. It normalizes the feature vector in each pixel to unit length, and is applied after the convolutional layers in the *Generator*. This is done to avoid signal magnitudes from spiraling out of control during training [15].

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{C} \sum_{j=0}^C a_{x,y}^j + \epsilon}} \quad (2.8)$$

where  $\epsilon = 10^{-8}$ ,  $N$  is the number of feature maps, and  $\alpha_{x,y}$  and  $\beta_{x,y}$  are the original and normalized feature vector in pixel  $(x, y)$  respectively.

In general, GANs tend to produce samples with less variation than that found in the training set. One approach to mitigate this issue is to have the Discriminator compute statistics across the batch, and use this information to help distinguish the *real* training data batches from the *fake* generated batches. This encourages the Generator to produce more variety, such that statistics computed across a generated batch are closer to the ones from a training batch. In ProGAN, a *minibatch standard deviation* layer is inserted near the end of the discriminator exactly for this reason. This layer has no trainable parameters. It computes the standard deviations of the feature map pixels across the batch, and appends them as an extra channel [15].

Moreover, the authors observed that in order to ensure a healthy competition between the generator and discriminator, it is essential that layers learn at a equivalent speed. To achieve an *equalized learning rate*, they scale dynamically the weights of a layer according to how many weights that layer has [15].

Furthermore, the authors used the improved *Wasserstein loss function*, also know as *WGAN - GP* [10].

$$Loss_G = -D(x') \quad (2.9)$$

$$GP = (\|\nabla D(ax' + (1 - a)x)\|_2 - 1)^2 \quad (2.10)$$

$$Loss_D = -D(x) + D(x') + \lambda * GP \quad (2.11)$$

Here,  $x'$  is the generated image,  $x$  is an image from the training set, and  $D$  is the Discriminator.  $GP$  is a gradient penalty that helps stabilize training. The  $a$  term in  $GP$  refers to a tensor of random numbers between 0 and 1, chosen uniformly at random. It is common to set  $\lambda = 10$ . Since we usually train in batches, the above losses are usually averaged over the minibatch. It is important to note that the *WGAN-GP loss function* does not expect the output of the discriminator to be a value between 0 and 1. This is slightly different than the traditional GAN formulation, which views the output of the discriminator as a probability.



## 2.7 Residual Network (ResNet)

*Residual Network (ResNet)* is a Convolutional Neural Network (CNN) that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper *Deep Residual Learning for Image Recognition* [12]. ResNet-152 (152 layers) is the deepest network ever presented on Imagenet and the one that won the 1st place in ILSVRC 2015 classification competition [12]. The deep representations also have excellent generalization performance on other recognition tasks (1st place on ImageNet Detection, ImageNet localization) which provides a strong evidence that the residual learning principle is generic and can be applied to other problems [12]. *Deep Neural Networks* incorporate low to high level features and classifiers end to end. Additionally, the learned features can be enriched by the number of stacked layers (depth). *Network depth* is of crucial importance however stacking more layers doesn't mean learning better networks. One obstacle to that is the famous problem of vanishing/exploding gradients. Another problem is the degradation of training accuracy problem which appears when networks start converging: when depth increases, accuracy gets saturated and then degrades rapidly. One might think that degradation issue is caused by overfitting but it actually indicates that not all systems are similarly easy to optimize [12]. *ResNet* is addressing the degradation problem by introducing a deep residual learning framework. The core idea is the so called *identity shortcut connection* (residual connection). *Identity Shortcut Connections* simply perform identity mapping and their outputs are added to the outputs of stacked layers. *Identity Shortcuts* add neither extra parameters nor complexity. The entire network can still be trained end to end by using stochastic gradient descent and backpropagation. Another important component in the architecture is the residual or building block visualized in Fig. 2.10.

A building block is defined as:

$$y = F(x, W_i) + x. \quad (2.12)$$

Here,  $x$  and  $y$  are the inputs and outputs respectively of the considered layers. The function  $F(x, W_i)$  represents the residual mapping to be learned. For example in Fig. 2.10 that has two layers,  $F = W_2\sigma(W_1x)$  in which  $\sigma$  denotes RELU [46] and the biases are omitted for simplifying notations. The operation  $F + x$  is performed by a shortcut connection and element - wise addition. We adopt the second non-linearity after the addition  $\sigma(y)$ .

However, it can happen that the dimensions of the input  $x$  vary from the dimensions of the output  $F$  (when changing input/output channels). This problem is overcome by performing a linear projection  $W_s$  by the shortcut connections to match the dimensions:

$$y = F(x, W_i) + W_sx. \quad (2.13)$$

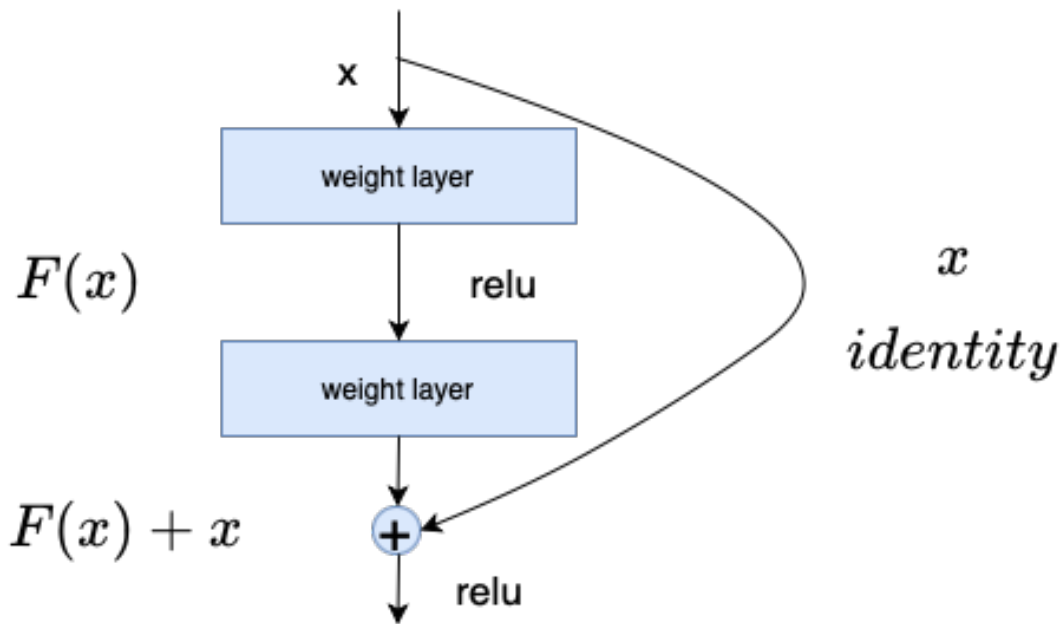


Figure 2.10: Residual Learning: A building Block.

ResNet architecture is mainly inspired by the philosophy of VGG Nets [41] with the new addition of the shortcut connection [12]. These shortcut connections then convert the network into the residual network. The VGG-19 model [41] has 19.6 billion FLOPs [12]. A plain network with 34 parameter layers has 3.6 billion FLOPs [12]. The residual counterpart with 34 layers and skip connections has exactly the same amount of FLOPs (3.6 billion FLOPs) with the plain network [12] (Fig. 2.11).

The authors of ResNet [12] have tested a plain 18-layer and a 34-layer convolutional neural network against two residual networks with the same depths. The plain 34-layer CNN performs worse than the 18-layer plain CNN. The ResNet architecture on the other hand achieved better results for the 34-layer network than for the 18-layer network, and altogether a better result than the plain network (Fig. 2.12).

## 2 State of the Art

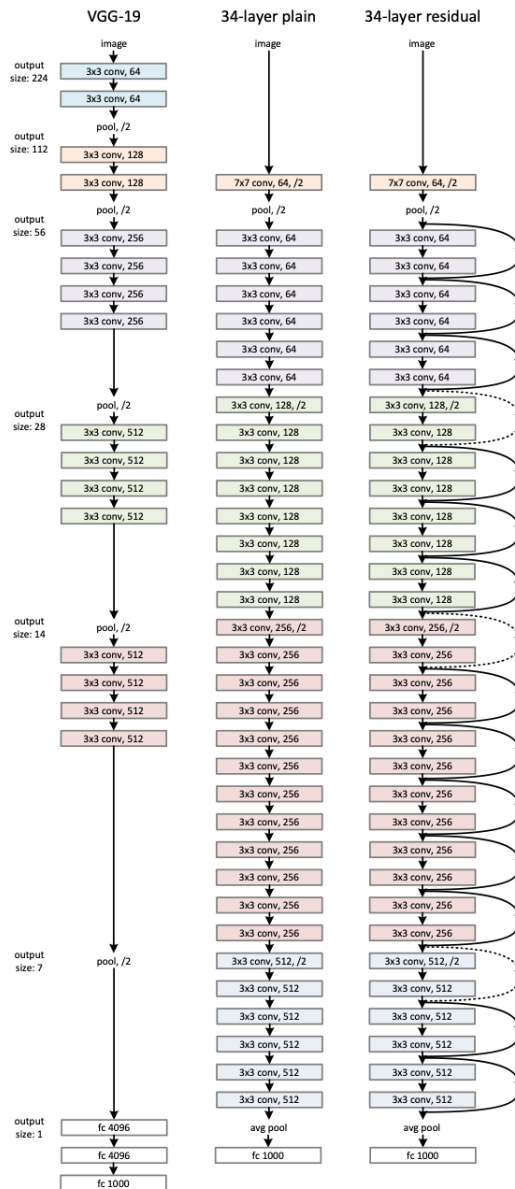


Figure 2.11: On the left we see the VGG-19 which was the baseline for ResNet. In the middle we see a plain network with 34 layers. On the right we see a residual network (ResNet) with 34 layers. Picture taken from [12].

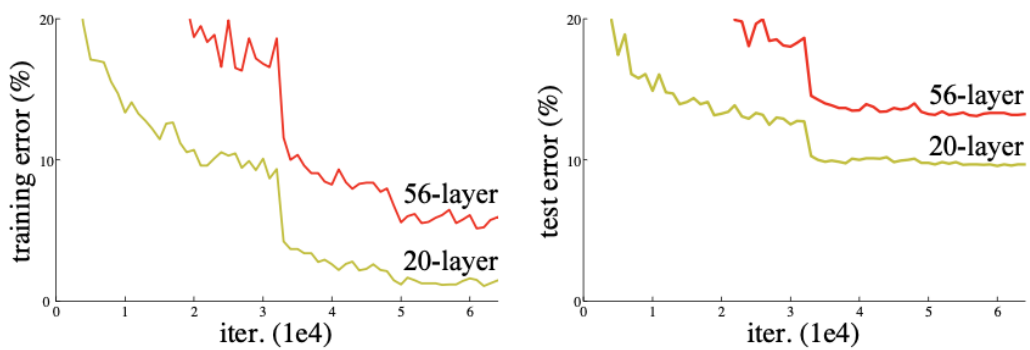


Figure 2.12: Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts. Diagrams taken from [12]

## 3 Deep Generative Modelling of Microscopy Image Data

In this chapter, we present our methodology end to end in order to generate synthetic Microscopy Image Data. First, we start by providing information about the Dataset and the Data Preprocessing techniques we used. Afterwards, we analyze our first contribution which is a new architecture for a Variational Autoencoder inspired by ResNet. Following our first contribution, we describe our second contribution which is the development and exploration of a novel training approach inspired by ProGAN for the ResNet based Variational Autoencoder. Next, we present evaluation metrics and the loss function. Finally, we provide the results and discuss a few implementation details.

### 3.1 Dataset

The Dataset has been provided by Dr. Catarina Cardoso from the Gutjahr Lab at the TUM School of Life Sciences. It consists of images that depict the soil microorganism *Arbuscular Mycorrhiza Fungi (AMF)* (Sec. 2.1). The images were taken by a Leica DM6 B wide-field microscope using a 10x magnification objective at 10-14 different focal planes across the root diameter. In total our dataset consists of 21 Microscopy Images of size 1536 x 2048 and a sample of them is visualized in Fig. 3.1.

### 3.2 Image Preprocessing

The Microscopy Images used for the training of the models are grayscale. A grayscale image is one in which the value of each pixel is a single sample representing only an amount of light and are composed of shades of gray. The intensity of a pixel is expressed within a given range between a minimum and a maximum. This range is represented from 0 (total absence, black) and 255 (total presence, white), with any fractional values in between. Although these pixel values can be presented directly to neural network models in their raw format, this can result in challenges during modeling, such as in the slower than expected training of the model. Instead, there can be great benefit in preparing the image pixel values prior to modeling, such as simply

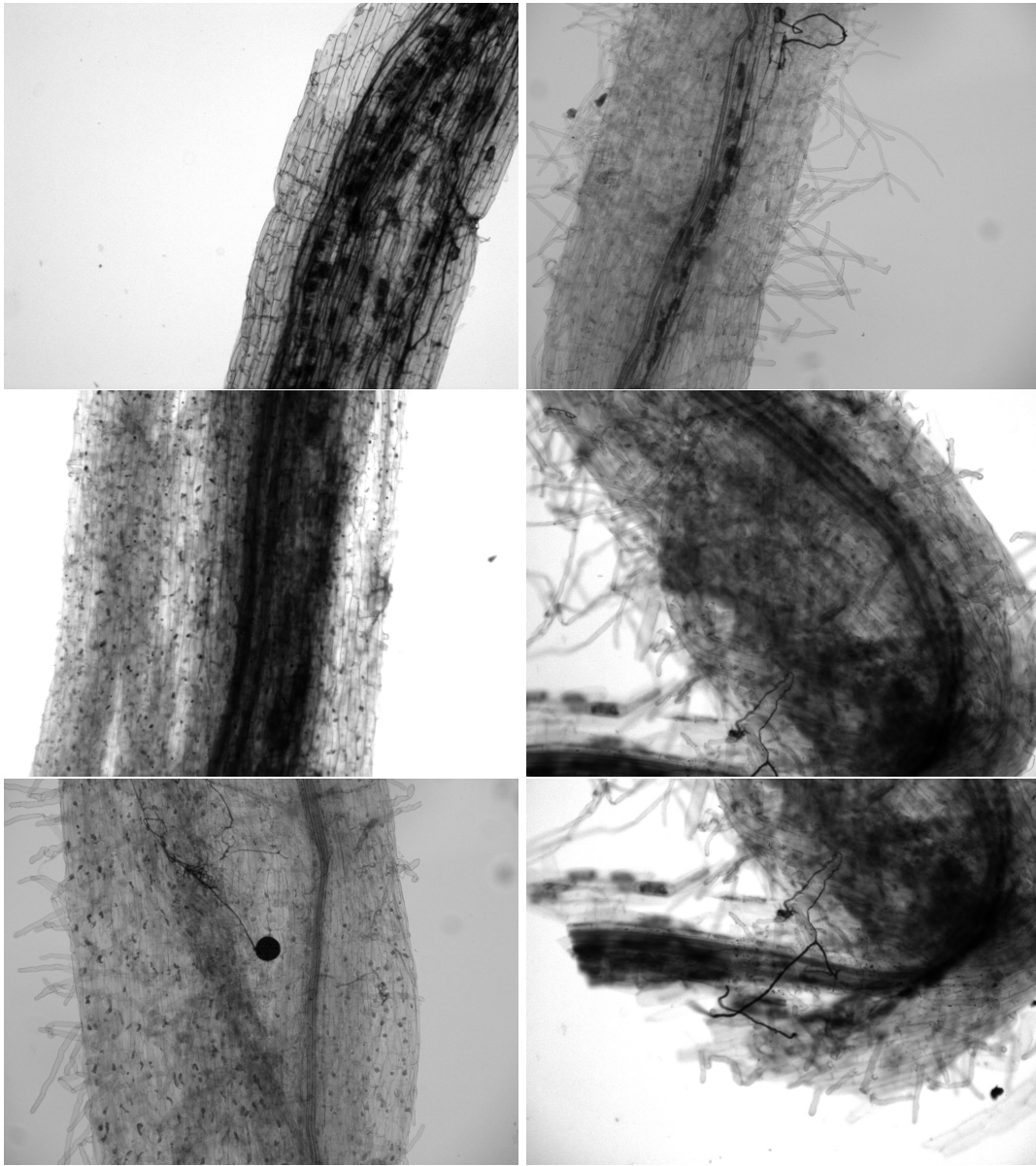


Figure 3.1: Six example Microscopy Images from the Dataset.

scaling pixel values to the range 0-1 to centering and even standardizing the values. In our case we positively standardized the pixel values. *Positive Standardization* means that the pixel values to remain in the positive domain in order to be able to visualize them later. A popular way of achieving this is the following: first we use the *z-score normalization* in order to standardize the data (mean  $\mu = 0$  and standard deviation  $\sigma = 1$ ). The z-score normalization of an image  $X_i$  is defined as:

$$Z(X_i) = \frac{X_i - \bar{X}_i}{\sigma_i} \quad (3.1)$$

where  $\bar{X}_i$  is the mean pixel value of the image  $X_i$  and  $\sigma_i$  the standard deviation across the pixel values of the image  $X_i$ . Afterwards, we clip the pixel values in the range  $(-1, 1)$  and then rescale the values from  $(-1, 1)$  to  $(0, 1)$ .

Furthermore, the initial images are split further into smaller ones, the so-called *chunks*. In this way, we are increasing the size of our dataset and focusing on learning smaller parts of an image rather a larger one. The chunks are extracted in an overlapping manner from each image. The total number of chunks extracted from each image is calculated as follows:

$$C_W = \frac{W - C}{S} + 1 \quad (3.2)$$

$$C_H = \frac{H - C}{S} + 1 \quad (3.3)$$

$$Total = C_W \times C_H \quad (3.4)$$

where  $W = 1536$  is the input width of the original image,  $H = 2048$  is the height of the input image,  $C = 224$  is the desired size of the extracted image, and  $S = 16$  is the stride, meaning how much overlap we would like the extracted images to have.

Last but not least, we observed that the root is usually centrally placed in the original images. Hence, there are extracted chunks which will be completely white. We found out that by removing the chunks that have mean pixel value large than 0.85 (completely white with no root info depicted) and smaller than 0.10 (completely black) reduces the noise while keeps a good content variation in the images. The thresholds are flexible and can be anytime adapted based on the quality of the initial images. The final size of our dataset after removing noise is 99791 images.

### 3.3 ResNet Based Variational Autoencoder

Our first contribution is a new architecture for Variational Autoencoder. The architecture is based on ResNet-18 [12] and developed from scratch. The architecture of the Encoder follows the logic of standard ResNet-18[12], meaning we input an image from a high dimension space and we project it in a lower dimension space. However, the Decoder is performing the exact opposite process, it takes as input a vector from the latent space and outputs an image in a much larger dimension space. Therefore, all the layer operations of the ResNet-18 Decoder have to be adapted to perform the opposite (mirrored) process of the ResNet-18 Encoder equivalent and still respect the ResNet-18 design (residual connection, residual blocks).

Specifically, the first operation which needs to be adapted for the ResNet-18 Decoder is the Convolutional layers of various kernel sizes. The Convolutional layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, we slide each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position [20]. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position (Fig. 3.2) [20]. The output dimensions are smaller than the input dimension. The PyTorch [31] function performing the convolution operation is `torch.nn.Conv2d` [32]. The convolution operation is described formally by the following equation:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(c_{out_j}, k) * input(N_i, k) \quad (3.5)$$

where  $*$  is the 2D cross-correlation operator [32],  $N$  is batch size,  $C$  denotes number of channels,  $H$  is height in pixels and  $W$  is width in pixels [32]. The output dimensions are calculated as follows:

$$W_{out} = (W_{in} - F + 2P) / S + 1 \quad (3.6)$$

$$H_{out} = (H_{in} - F + 2P) / S + 1 \quad (3.7)$$

where  $W$  is width of the image in pixels,  $H$  is the height of the image in pixels,  $F$  is the filter/kernel size,  $P$  the amount of zero padding and  $S$  is the stride [32].

For the ResNet-18 Decoder we need a convolution operation which gives larger output dimensions than the input. This operation is called Transposed Convolution and in PyTorch is implemented by the function `torch.nn.ConvTranspose2d` [33]. In the Transposed Convolution, we multiply every element of the input with every element



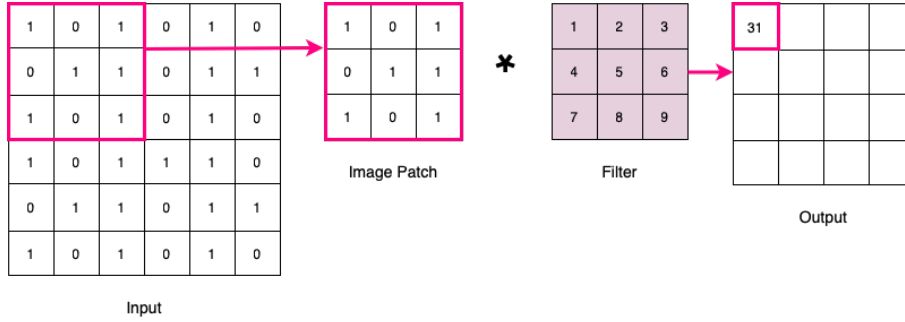


Figure 3.2: Convolution operation

of the kernel starting from the upper left element as shown in Fig. 3.3. Some of the elements of the resulting upsampled feature maps are over-lapping. To solve this issue, we simply add the elements of the over-lapping positions [47]. The output dimensions of the Transposed Convolution are calculated as follows:

$$W_{out} = S * (W_{in} - 1) + F - 2 * P \tag{3.8}$$

$$H_{out} = S * (H_{in} - 1) + F - 2 * P \tag{3.9}$$

where  $W$  is width of the image in pixels,  $H$  is the height of the image in pixels,  $F$  is the filter/kernel size,  $P$  the amount of zero padding and  $S$  is the stride [47].

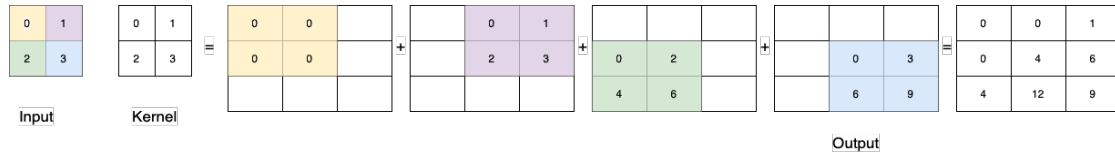


Figure 3.3: Transposed Convolution. A 2x2 input upsampled to 3x3.

The second operation that needs to be adapted for the Resnet-18 Decoder is pooling [20]. There’s a maxpooling happening at the beginning of ResNet-18 but also average pooling which is happening at the end before the fully connected layer [12] (Fig. 3.4). Pooling reduces the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting [20]. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max or average operation (Fig. 3.4). The output dimensions of pooling layers are calculated as follows:

$$H_{out} = (H_{in} - F + 1) / S \quad (3.10)$$

$$W_{out} = (W_{in} - F + 1) / S \quad (3.11)$$

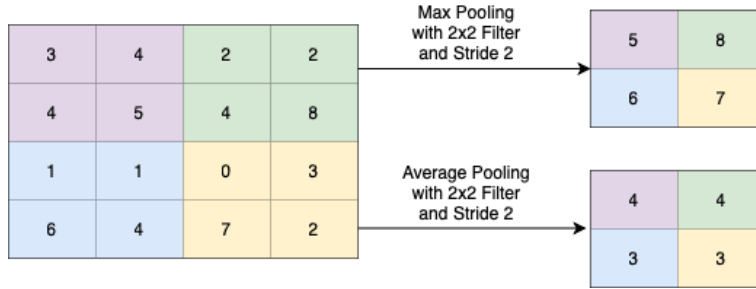


Figure 3.4: Illustration of Max and Average Pooling.

For the ResNet-18 Decoder we need an un-pooling operation, in order to revert the effect of the max/avg pooling operation which happens in ResNet-18 Encoder. However, a traditional un-pooling operation would be problematic for the following two reasons: First of all, we would have to save the exact pixel values at each pooling operation in order to be able to retrieve them in the un-pooling, which makes the computation less efficient. Secondly, in case we did not keep the original pixel values in each pooling layers then in the un-pooling we would have to fill them with zeros which would introduce a lot of noise. Therefore, we decided to replace the un-pooling with a Transposed Convolution [33] layer. In the case want to double the output dimensions we are using Transposed Convolution with kernel 2 and stride 2.

Last but not least, in ResNet-18 Encoder we downsample the input at the first residual block of each layer by using convolutional operation with stride 2. Except the first layer where the input is downsampled using maxpool with *stride* 2. We are following the exact same logic for the ResNet-18 Decoder but mirrored and instead of downsampling we are performing upsampling. The architecture of the both the ResNe-18 Encoder and ResNet-18 Decoder is visualized in Fig. 3.5. The results from training the Variational Autoencoder with the new architecture are presented in Sec. 3.6.1.

### 3.4 Progressive Variational Autoencoder (proVAE)

Our second contribution is the development and exploration of a novel training approach for the ResNet-18 Variational Autoencoder (Sec. 3.3). The training method has

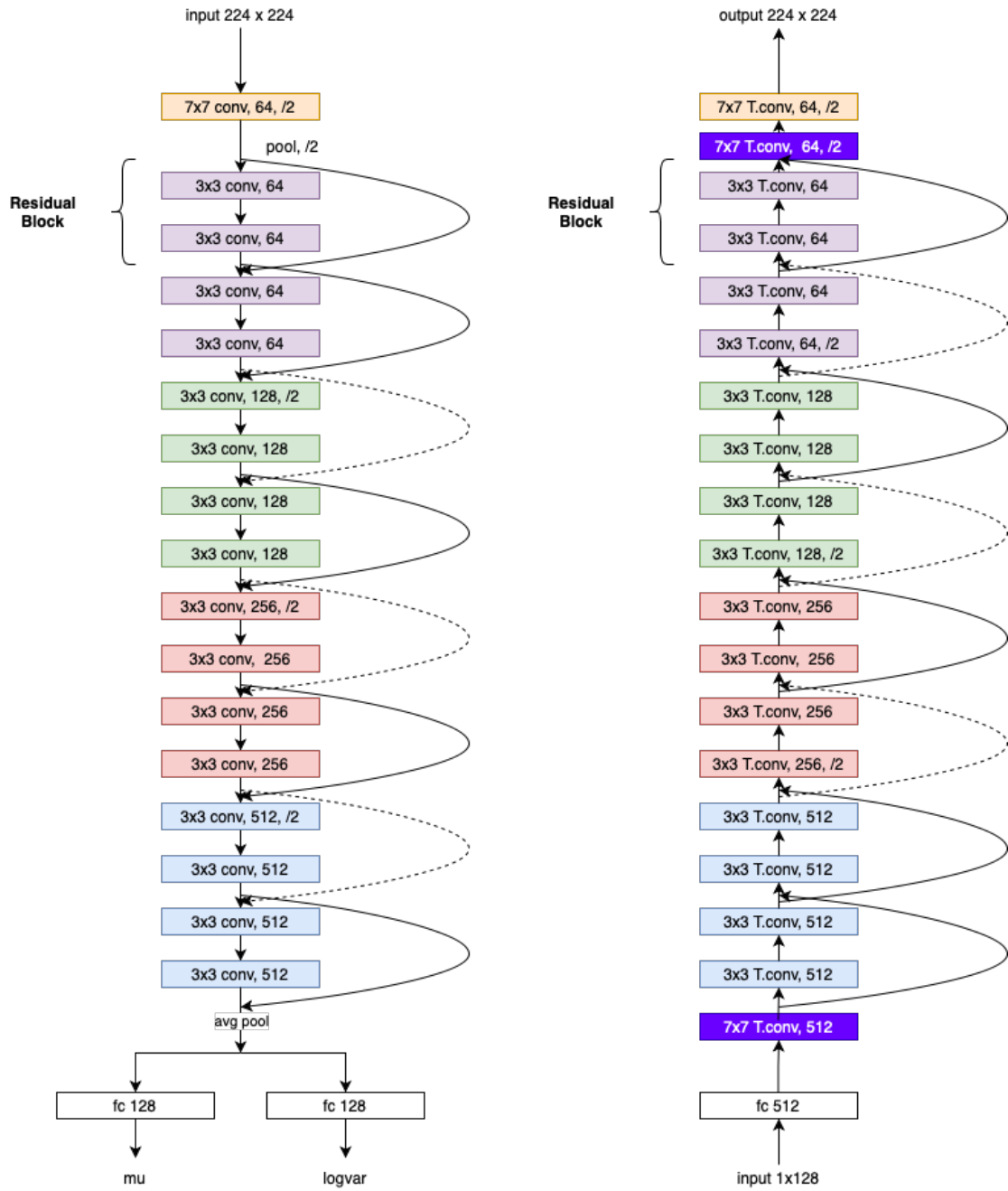


Figure 3.5: ResNet Encoder and ResNet Decoder network architectures.

been inspired by Progressive Generative Adversarial network (ProGAN) (Sec. 2.6) is called proVAE (progressive Variational Autoencoder).

The training of ResNet-VAE in the new setup is happening in a progressive manner. We train the Encoder and Decoder end to end and we start the training at a low resolution image (7x7). In the first iteration only the last block of the Encoder and the first block of the Decoder are trained. Since our VAE is ResNet based, when referring to a block we mean a residual block (Fig. 2.10). As soon as the training is satisfactory, we proceed by appending one more block to the (left of) Encoder and one more layer to the (right of) Decoder. The trained weights in the earlier layers are kept and used in the next appended layer. We repeat the same steps until we have reached the full size of the model. However, in order to prevent the network from having big shocks when inserting the new untrained block, instead of inserting it directly to the flow we are instead fading it in. The fading in is controlled by a parameter  $\alpha$  which is linearly interpolated from 0 to 1 over many training iterations. Additionally, we also implemented the equalized learning rate and the pixel normalization (which replaces the batch normalization) as described in Sec. 2.6. Last but not least, as each appended block requires a different image size than the initial one (224x224), we resize the image to the needed dimension and feeding in to the model. The progressive training process is visualized in Fig. 3.6. The visualization follows the same color conventions as in Fig. 3.5 for better explainability and understanding.

The reason that led us in the development and exploration of this method is twofold. First of all, we want to overcome the common blurring issue of VAEs which we also observed in our own results. Secondly, we would like our model to improve further in learning and generating more detailed images. Therefore, we investigate whether the progressive training and the fact that the networks are asked to learn a much simpler piece of the overall problem can help us mitigate the above mentioned issues. The results of this method are presented in Sec. 3.6.2.

### 3.5 Evaluation metrics

The loss function of Variational Autoencoder consists of two terms, one which maximizes the probability of the data and a second term the Kullback - Leibler Divergence which encourages our approximate posterior distribution  $q_\phi(z|x)$  (learnable) to be similar to the true posterior  $p_\theta(z|x)$ :

$$ELBO = \log p_\theta(x) - KL(q_\phi(z|x)||p_\theta(z|x)) \quad (3.12)$$

In order to maximize the first term, we chose to minimize the Mean Squared Error (MSE) loss.

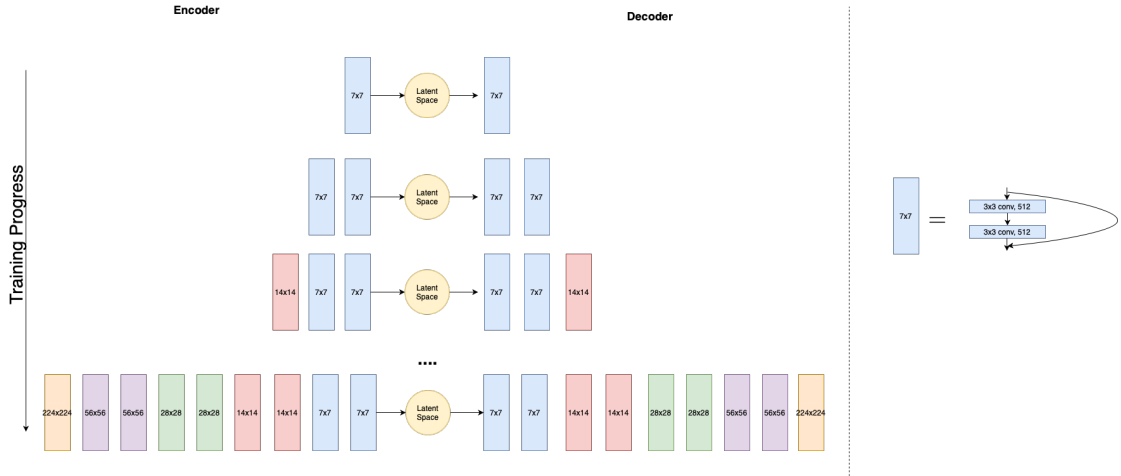


Figure 3.6: Progressive Training of the ResNet-18 Variational Autoencoder. The training starts with both the Encoder (E) and Decoder (D) operating at low spatial resolution of  $7 \times 7$ . As the training advances, we incrementally add layers to E and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here  $N \times N$  (e.g  $7 \times 7$ ) refers to residual blocks (as visualized in the right side of the diagram) that receive as input an  $N \times N$  image size. The colors convention follow the one in Fig. 3.5 for consistency and better explainability.

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^N (y - \hat{y})^2 \quad (3.13)$$

where  $\hat{y}$  is the predicted pixel value,  $y$  is the original pixel value and  $N$  is the size of the image (total number of pixels).

### 3.6 Experiments & Results

In this section, we present the results from training the ResNet-18 VAE using the standard conventional approach as described in Sec. 3.3) and by training the ResNet VAE using the progressive method as described in Sec. 3.4. Finally, we provide information about the implementation and the infrastructure used for the training.

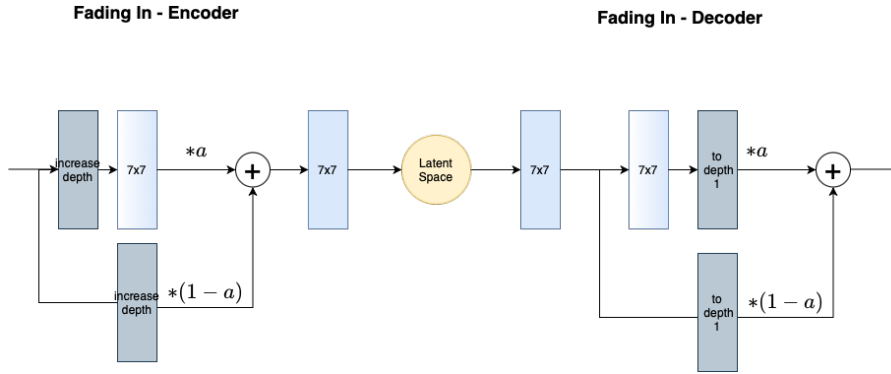


Figure 3.7: The example illustrates the fading-in of the 7x7 residual block (blue gradient) in the Encoder (left) and Decoder (right) (see progressive VAE architecture in Diag 3.6 ). Here, the 7x7 residual block have similar meaning to the one explained in Diag. 3.6. The gray blocks 'increased depth' and 'to depth 1' are used in order to adapt the input depth to the needed block depth.

### 3.6.1 Results ResNet-18 Variational Autoencoder

We run a series of experiments in order to find the best learning rate and the best number of epochs by performing grid search. First, we split our dataset in *Training*, *Validation* and *Test* Set (Table 3.1). Our hyperparameter optimization for the learning rate and number of epochs happened on the validation set. Evaluation metrics for unsupervised generative models still remain an open challenge, therefore the evaluation happened manually by comparing the quality of reconstructed images on validation set. The range that we experimented for the *learning rate* was  $[0.0005, 0.002]$  with  $\mathbf{lr} = 0.0005$  being the best value. The range that we experimented for the *Number of Epochs* was  $[70, 110]$  with  $\mathbf{epochs} = 90$  being the best value. After finding the best value for the learning rate and the number of epochs we used them in order to perform inference (generation) on the Test Set. The results are presented in the Tables 3.2 & 3.3 & 3.4.

Data Split	No. Images	Image Size
Training	97791	224x224
Validation	1000	224x224
Test	1000	224x224
<b>Total</b>	99791	224x224

Table 3.1: Information about Dataset split.

In both reconstructed and generated images we observe a blurriness which is a

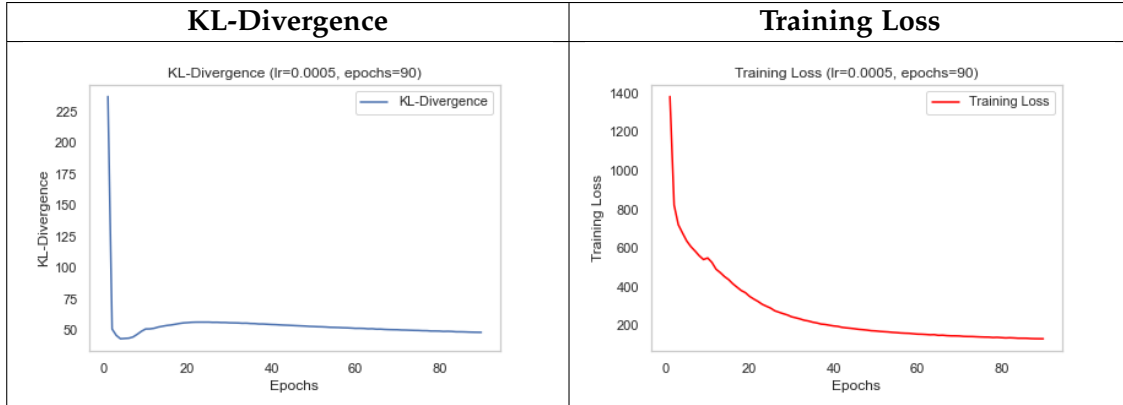


Table 3.2: KL-Divergence and Training Loss of the best model.

very common phenomenon in variational autoencoders. However, the results are rather promising and the model is able to approximate really well the true posterior distribution.

### 3.6.2 Results Progressive ResNet Variational Autoencoder (proVAE)

We run a series of experiments in order to find the best number of epochs we should train per each faded in block. First, we split our dataset in *Training*, *Validation* and *Test* Set according to the Table 3.1. Afterwards, we tested different number of epochs by performing grid search while keeping the **learning rate = 0.0005** for all the experiments. The range that we experimented for the *Number of Epochs* was [30, 40] per block with the best value being **epochs = 30**. As the problem is still unsupervised, the challenge with the evaluation metric still remains as in the previous results. Therefore the evaluation happened manually by comparing the quality of reconstructed images on validation set. After having identified the best hyperparameters for the model, we used them in order to perform inference (generation) on the *Test Set*. The results are presented in the tables 3.5 & 3.6 & 3.7.

To sum up, the results of proVAE do not outperform the results of the regular training method 3.6.1. The blurry effect is much more visible in proVAE results and also the model learned less details in proVAE compared to the conventional training of Resnet-18 VAE. In general, developing and tuning the proVAE has been quite a challenge since it is an advanced architecture with a lot of parameters that need to be developed and taken care of (e.g pixelnorm versus batchnormalization, alpha parameter, equalized learning rate, image resizing, weight clipping).

### 3.7 Implementation Details

The training of Variational Autoencoder has happened at two places: in Google Cloud Platform (GCP) [28] and at Home One Cluster installed in one of RBG server rooms and owned by the Chair of Scientific Computing in Computer Science at TUMunich. Specifically, the training of our first contribution, the Variational Autoencoder with the ResNet inspired architecture happened at GCP while the training of our second contribution, the Progressive Variational Autoencoder (ProVAE) happened at Home One Cluster. The specifications for both of the machines are provided in the table [add reference]. In GCP, we configured a Virtual Machine Instance with GPU and exposed the jupyter notebook to an external IP address in order to be able to utilize the GUI. In order to enable training with GPUs in our VM instance in GCP, we installed all the needed CUDA drivers [6]. Furthermore, we used neptune.ai [23] for keeping track of all the experiments, saving all the hyperparameters, storing metrics and versioning the models. Neptune is a metadata store for MLOps, built for research and production teams that run a lot of experiments [23]. The data is stored in a central Metadata Database hosted by the providers [23]. The database can be accessed from any place that you run your code by providing your authentication credentials and the name of the project that you would like to register the data.

All the pipeline (from reading the images to training the networks and visualizing the results) has been implemented in Python3 [30]. For data preprocessing and visualization we used libraries like OpenCV [21], NumPy [26], Matplotlib [22], seaborn [39]. The Variational Autoencoder is implemented using the Pytorch library [31]. Last but not least, we used the venv module [29] in order to create lightweight “virtual environments” with their own site directories, isolated from system site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories [29].



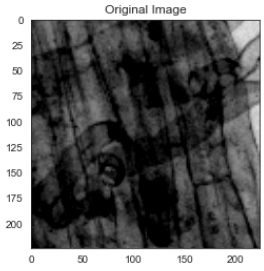
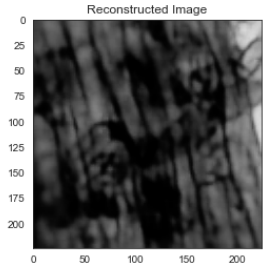
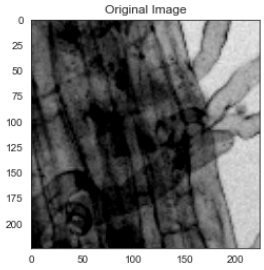
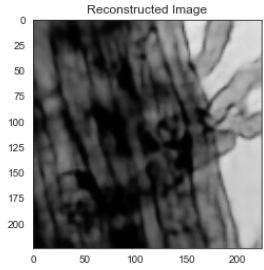
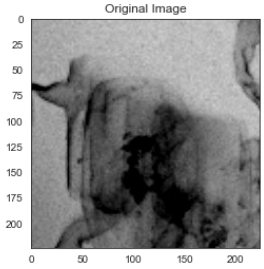
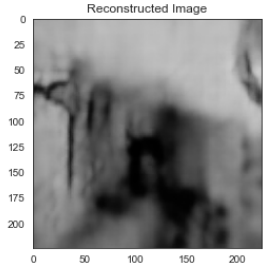
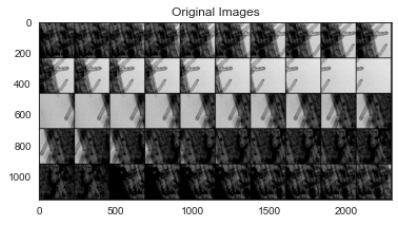
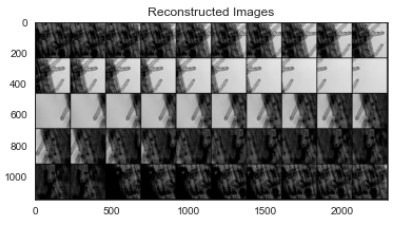
Original Image	Reconstructed Image
	
	
	
	

Table 3.3: Original and Reconstructed Images on Test set using the best model for ResNet-18 VAE. In the first three rows we provide single images and their reconstructed equivalent. In the last row we provide a grid of 50 original images and the grid with the 50 equivalent reconstructed images.

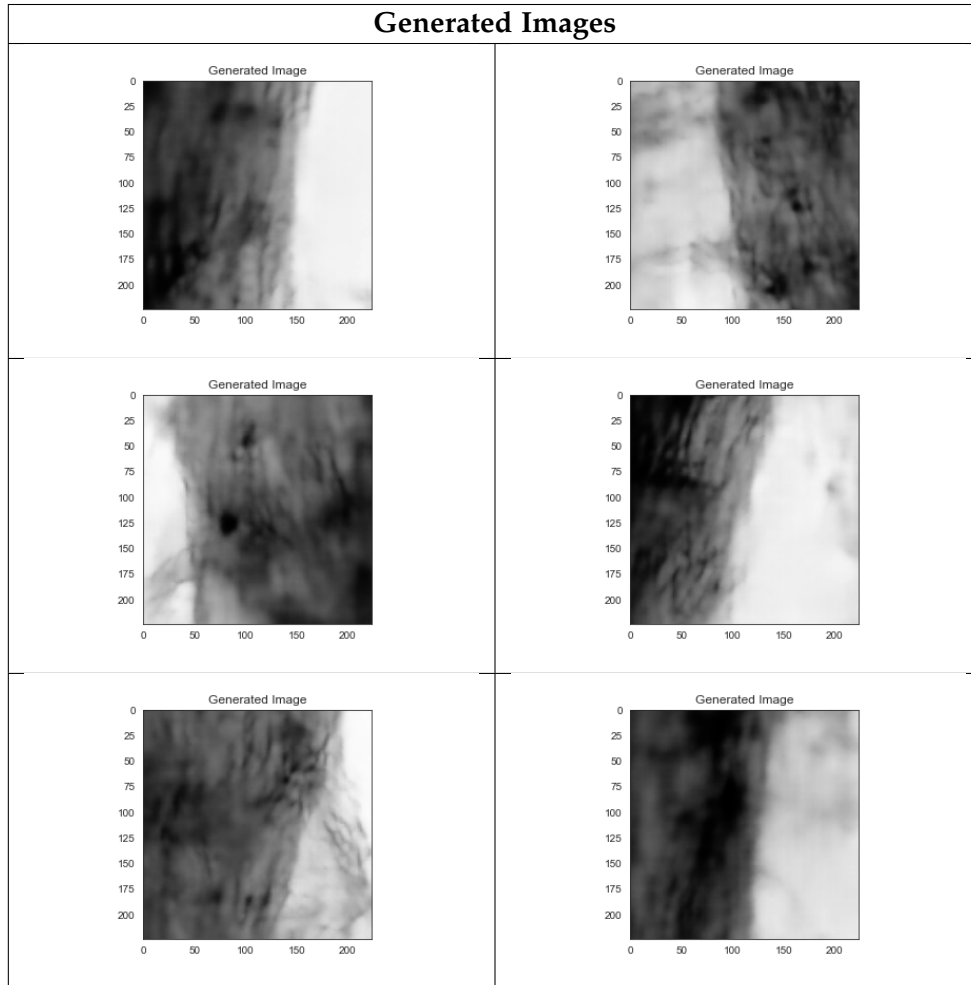


Table 3.4: Generated Images using the best model for ResNet-18 VAE. We observe that the model is able to learn very detailed pictures and also pictures that have different orientation. This is happening because in the training set for example we have chunks that are coming from the right part of the root (usually the root is centrally placed in the picture), meaning that in this case the root appears on the left side and then the white background is on the right hand-side.

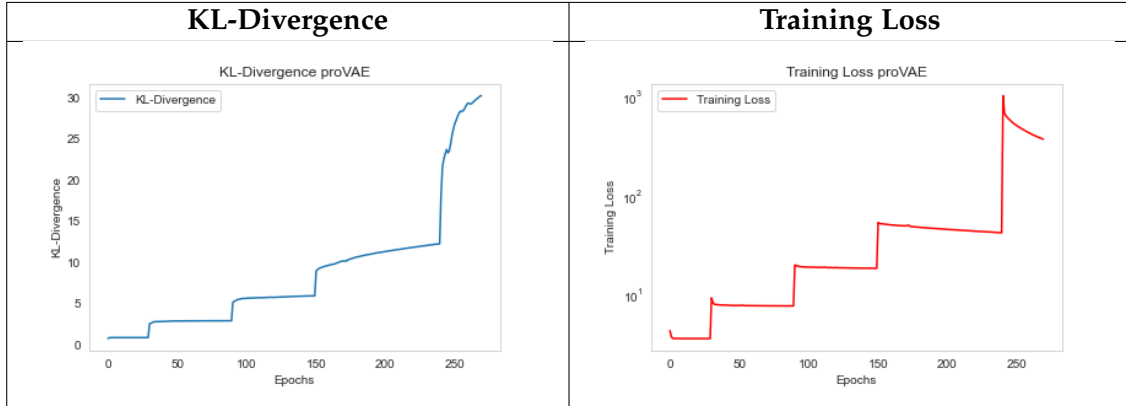


Table 3.5: KL-Divergence and Training Loss of the best model for proVAE. In those diagrams we observe the KL-Divergence and the Loss respectively for all the appended (faded in) blocks. In both diagrams we are observing 'steps' because we are having a new training round every time we append a new block. One training round lasts 30 epochs. The loss value in each training round might start from a different starting value. The starting value is higher in the cases in which the appended block operates in a higher spatial resolution (e.g we have trained the 7x7 and we add the 14x14) because we are using MSELoss without averaging over the total number of pixels. Therefore, in higher spatial resolution there are more pixels that we need to compare for the loss and therefore the loss is higher. The same applies for the KL-Divergence, however as the KL-Divergence is a regulating factor in the ELBO loss it might increase but it also might decrease (depending on what the gradient descent 'decides').

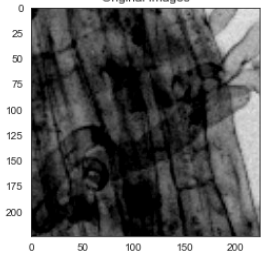
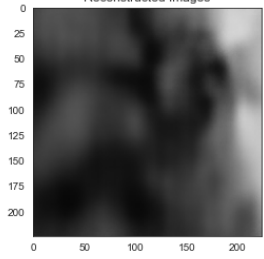
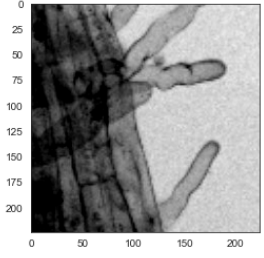
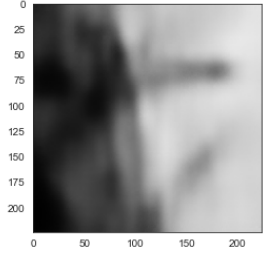
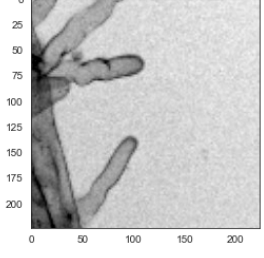
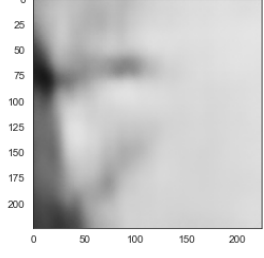
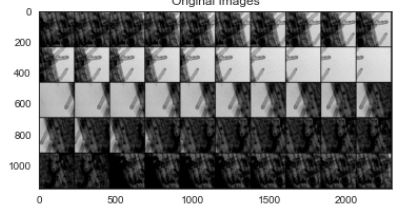
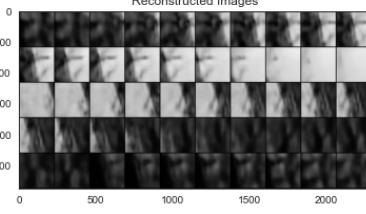
Original Image	Reconstructed Image
	
	
	
	

Table 3.6: Original and Reconstructed Images on Test set using the best model for proVAE. In the first three rows we provide single images and their reconstructed equivalent. In the last row we provide a grid of 50 original images and the grid with the 50 equivalent reconstructed images.

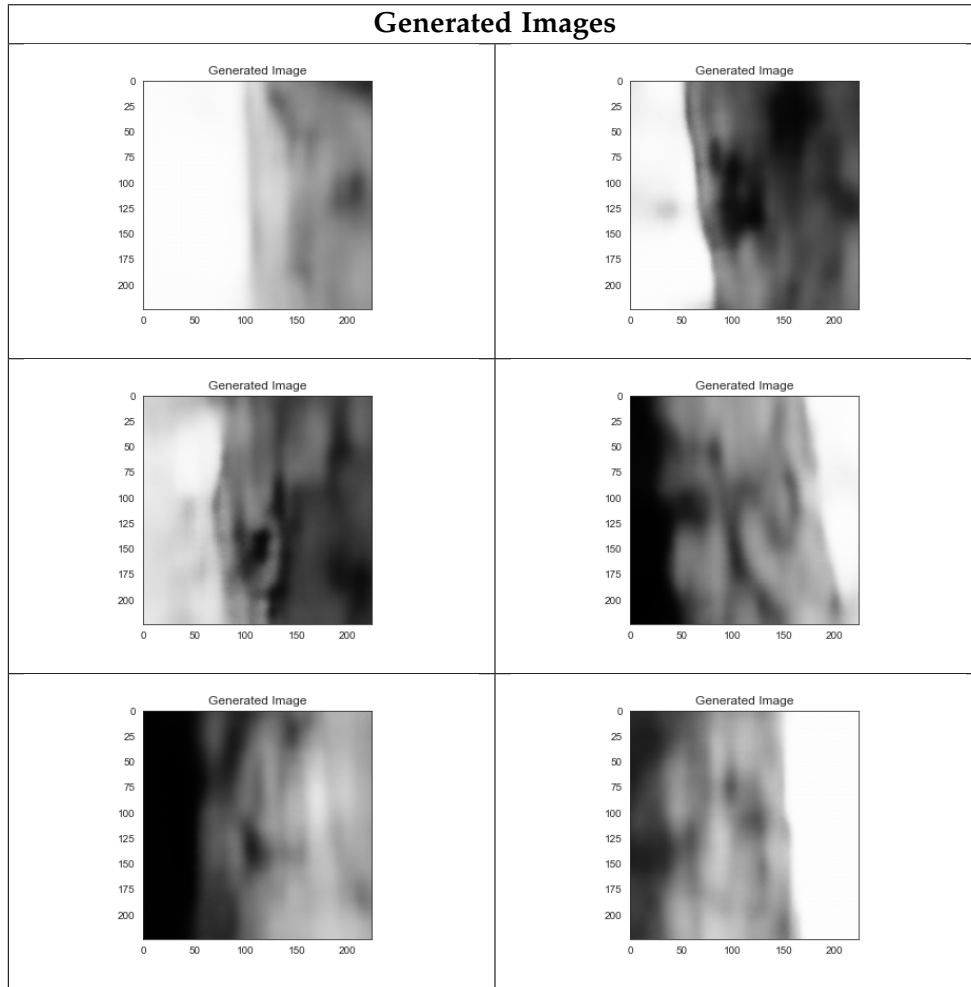


Table 3.7: Generated Images using the best model for proVAE.

	Google Cloud Platform	Home One Cluster
CPU	Intel(R) Xeon(R) CPU @ 2.30GHz	AMD EPYC 7402 @ 2.80GHz
RAM	15 GB	256 GB
GPU	NVIDIA Tesla T4	RTX 3080 Turbo with 10GB video RAM
OS	Ubuntu 16.04 Xenial	Ubuntu 20.04.2 LTS

Table 3.8: Technical specifications of the compute nodes used for training and image generation.

## 4 Conclusion

We presented an end-to-end pipeline on how to create Synthetic Microscopy Images using a Deep Generative Model which is a Variational Autoencoder (VAE). First, we introduced the image preprocessing techniques that we used and then we analyzed in depth our main two contributions. The first contribution is a new architecture for VAE inspired by ResNet (ResNet-18 VAE) and the second contribution is a novel way of training the ResNet-18 VAE inspired by the famous ProGAN. Finally, we described the loss function that we used for the training and presented results for our two contributions. The results are promising and motivate future research in the field. In conclusion, the overall aim of this Master Thesis, which is the generation of Synthetic Microscopy Image Data which look realistic has been completed successfully.

Currently, the end-to-end pipeline as described above is in place and ready to be used by a future researcher in order to generate realistic looking image chunks, but not a complete image. Those realistic looking patches can be created by using the ResNet-18 VAE which has given the best results so far. In this way, more training data can be created and increase the size of training data, allowing it in this way to be used further for classification or segmentation purposes.

As for the future, there would be two directions to be further developed. The first one is the improvement of the current performance of the models, especially the progressive training of ResNet-18 VAE. The current model for progressive training does not outperform the conventional training and we still observe the blurry effect in the results. Therefore it would be interesting to explore further its potential by using a different loss function, experimenting with chunks generation (increase or decrease overlap), merging them into one picture, trying simple pixel normalization instead of positive standardization, including minibatch standard deviation in proVAE as described in [15], including data augmentation techniques (e.g high contrast in order to boost the learning of smaller details) and even experimenting with different prior distributions.

The second direction would be the exploration of evaluation metrics for Variational Autoencoders. The metric will enable researchers with evaluating the quality and the diversity of the generated images in a formal way, limiting in this way the manual evaluation which is sensitive to subjectivity. One idea could be to adapt the famous Inception Score [38] for Variational Autoencoders. This would require to manually

#### 4 Conclusion

---

label some samples as *fake* or *real* or use methods like active learning to automate the labelling process [35].

# List of Figures

1.1	Data Augmentation using Generative Modelling. . . . .	2
2.1	Schematic representation of AMF establishment inside a host and the known exchange between the two partners. Figure credit: Florence Sessoms [40] . . . . .	4
2.2	New data instances generated by VQ-VAE 2. The model generates realistic looking faces that respect long-range dependencies such as matching eye colour or symmetric facial features, while covering lower density modes of the input dataset (e.g., green hair). Picture taken from [34]. . . . .	5
2.3	Discriminative vs Generative Model of handwritten digits. . . . .	6
2.4	Illustration of an Autoencoder with its loss function. . . . .	7
2.5	Architecture of Generative Adversarial Network. . . . .	10
2.6	Backpropagation in Discriminator Training. . . . .	10
2.7	Backpropagation in Generator Training. . . . .	11
2.8	Training setup in ProGAN. On the right six example images are generated using progressive growing at $1024 \times 1024$ . Picture taken from [15] . . .	12
2.9	Illustration of Fading in. In this figure we observe the transition from $16 \times 16$ images (a) to $32 \times 32$ images (c). Here 2x and 0.5x refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The toRGB represents a layer that projects feature vectors to RGB colors and fromRGB does the reverse; both use $1 \times 1$ convolutions. When training the discriminator, the real images that are fed in are downscaled to match the current resolution of the network. During a resolution transition, interpolation is happening between two resolutions of the real images, similarly to how the generator output combines two resolutions. Picture taken from [15]. . . . .	13
2.10	Residual Learning: A building Block. . . . .	16
2.11	On the left we see the VGG-19 which was the baseline for ResNet. In the middle we see a plain network with 34 layers. On the right we see a residual network (ResNet) with 34 layers. Picture taken from [12]. . . .	17



2.12	Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts. Diagrams taken from [12] . . . . .	18
3.1	Six example Microscopy Images from the Dataset. . . . .	20
3.2	Convolution operation . . . . .	23
3.3	Transposed Convolution. A 2x2 input upsampled to 3x3. . . . .	23
3.4	Illustration of Max and Average Pooling. . . . .	24
3.5	ResNet Encoder and ResNet Decoder network architectures. . . . .	25
3.6	Progressive Training of the ResNet-18 Variational Autoencoder. The training starts with both the Encoder (E) and Decoder (D) operating at low spatial resolution of 7x7. As the training advances, we incrementally add layers to E and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ (e.g 7x7) refers to residual blocks (as visualized in the right side of the diagram) that receive as input an $N \times N$ image size. The colors convention follow the one in Fig. 3.5 for consistency and better explainability. . . . .	27
3.7	The example illustrates the fading-in of the 7x7 residual block (blue gradient) in the Encoder (left) and Decoder (right) (see progressive VAE architecture in Diag 3.6 ). Here, the 7x7 residual block have similar meaning to the one explained in Diag. 3.6. The gray blocks 'increased depth' and 'to depth 1' are used in order to adapt the input depth to the needed block depth. . . . .	28

## List of Tables

3.1	Information about Dataset split. . . . .	28
3.2	KL-Divergence and Training Loss of the best model. . . . .	29
3.3	Original and Reconstructed Images on Test set using the best model for ResNet-18 VAE. In the first three rows we provide single images and their reconstructed equivalent. In the last row we provide a grid of 50 original images and the grid with the 50 equivalent reconstructed images.	31
3.4	Generated Images using the best model for ResNet-18 VAE. We observe that the model is able to learn very detailed pictures and also pictures that have different orientation. This is happening because in the training set for example we have chunks that are coming from the right part of the root (usually the root is centrally placed in the picture), meaning that in this case the root appears on the left side and then the white background is on the right hand-side. . . . .	32
3.5	KL-Divergence and Training Loss of the best model for proVAE. In those diagrams we observe the KL-Divergence and the Loss respectively for all the appended (faded in) blocks. In both diagrams we are observing 'steps' because we are having a new training round every time we append a new block. One training round lasts 30 epochs. The loss value in each training round might start from a different starting value. The starting value is higher in the cases in which the appended block operates in a higher spatial resolution (e.g we have trained the 7x7 and we add the 14x14) because we are using MSELoss without averaging over the total number of pixels. Therefore, in higher spatial resolution there are more pixels that we need to compare for the loss and therefore the loss is higher. The same applies for the KL-Divergence, however as the KL-Divergence is a regulating factor in the ELBO loss it might increase but it also might decrease (depending on what the gradient descent 'decides').	33
3.6	Original and Reconstructed Images on Test set using the best model for proVAE. In the first three rows we provide single images and their reconstructed equivalent. In the last row we provide a grid of 50 original images and the grid with the 50 equivalent reconstructed images. . . .	34
3.7	Generated Images using the best model for proVAE. . . . .	35

*List of Tables*

---

3.8	Technical specifications of the compute nodes used for training and image generation. . . . .	35
-----	---	----

## Bibliography

- [1] A. Antoniou, A. Storkey, and H. Edwards. “Data augmentation generative adversarial networks.” In: *arXiv preprint arXiv:1711.04340* (2017).
- [2] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. “Variational inference: A review for statisticians.” In: *Journal of the American statistical Association* 112.518 (2017), pp. 859–877.
- [3] A. Brock, J. Donahue, and K. Simonyan. “Large scale GAN training for high fidelity natural image synthesis.” In: *arXiv preprint arXiv:1809.11096* (2018).
- [4] C. Chadebec, E. Thibeau-Sutre, N. Burgos, and S. Allasonnière. “Data Augmentation in High Dimensional Low Sample Size Setting Using a Geometry-Based Variational Autoencoder.” In: *arXiv preprint arXiv:2105.00026* (2021).
- [5] M. Chen, M. Arato, L. Borghi, E. Nouri, and D. Reinhardt. “Beneficial Services of Arbuscular Mycorrhizal Fungi – From Ecology to Application.” In: *Frontiers in Plant Science* 9 (2018). ISSN: 1664-462X. DOI: 10.3389/fpls.2018.01270.
- [6] CUDA. *CUDA*. <https://developer.nvidia.com/cuda-toolkit>.
- [7] I. Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. <https://arxiv.org/pdf/1701.00160.pdf>. 2017.
- [8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial nets.” In: *Advances in neural information processing systems* 27 (2014).
- [9] Google. *Background: What is a Generative Model?* <https://developers.google.com/machine-learning/gan/generative>. May 2019.
- [10] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. “Pixelvae: A latent variable model for natural images.” In: *arXiv preprint arXiv:1611.05013* (2016).
- [11] S. Günnemann. *Machine Learning for Graphs and Sequential Data*. <https://www.in.tum.de/en/daml/teaching/mlgs/>.
- [12] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

- [13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium." In: *Advances in neural information processing systems* 30 (2017).
- [14] T. Jebara. *MACHINE LEARNING: Discriminative and Generative*. Springer, 2004.
- [15] T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive growing of gans for improved quality, stability, and variation." In: *arXiv preprint arXiv:1710.10196* (2017).
- [16] D. P. Kingma and M. Welling. "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (2013).
- [17] A. Kolesnikov and C. H. Lampert. "PixelCNN models with auxiliary variables for natural image modeling." In: *International Conference on Machine Learning*. PMLR, 2017, pp. 1905–1914.
- [18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.
- [19] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. "Photo-realistic single image super-resolution using a generative adversarial network." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.
- [20] F. F. Li. *Convolutional Neural Networks (CNNs / ConvNets)*. <https://cs231n.github.io/convolutional-networks/#conv>. 2021.
- [21] O. S. C. V. Library. *Open Source Computer Vision Library*. <https://opencv.org>.
- [22] Matplotlib. *Matplotlib: Visualization with Python*. <https://matplotlib.org>.
- [23] neptune.ai. *neptune.ai*. <https://neptune.ai>.
- [24] A. NG. *CS229: Lecture Notes*. <https://akademik.bahcesehir.edu.tr/~tevfik/courses/cmp5101/cs229-notes1.pdf>. 2012.
- [25] A. Ng and M. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." In: *Advances in neural information processing systems* 14 (2001).
- [26] Numpy. *Numpy*. <https://numpy.org>.
- [27] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "Wavenet: A generative model for raw audio." In: *arXiv preprint arXiv:1609.03499* (2016).

- [28] G. C. Platform. *Google Cloud Platform*. <https://cloud.google.com>.
- [29] Python. *Creation of virtual environments*). <https://docs.python.org/3/library/venv.html>.
- [30] Python. *Python*. <https://www.python.org/downloads/>.
- [31] PyTorch. *Pytorch*. <https://pytorch.org>.
- [32] PyTorch. *Pytorch Convolution 2D*. <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>. 2019.
- [33] PyTorch. *Pytorch Transpose Convolution*. <https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html#convtranspose2d>.
- [34] A. Razavi, A. Van den Oord, and O. Vinyals. "Generating diverse high-fidelity images with vq-vae-2." In: *Advances in neural information processing systems* 32 (2019).
- [35] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang. "A survey of deep active learning." In: *ACM Computing Surveys (CSUR)* 54.9 (2021), pp. 1–40.
- [36] D. Rezende and S. Mohamed. "Variational inference with normalizing flows." In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.
- [37] C. Rudin. *MIT 15.097 Prediction: Machine Learning and Statistics*. [https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15\\_097S12\\_lec08.pdf](https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec08.pdf). 2012.
- [38] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved techniques for training gans." In: *Advances in neural information processing systems* 29 (2016).
- [39] seaborn. *Seaborn: Statistical Data Visualization*). <https://seaborn.pydata.org>.
- [40] F. Sessoms. *Arbuscular mycorrhizal fungi: tiny friends with big impact*. <https://turf.umn.edu/news/arbuscular-mycorrhizal-fungi-tiny-friends-big-impact>. 2020.
- [41] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).
- [42] L. Theis and M. Bethge. "Generative image modeling using spatial lstms." In: *Advances in neural information processing systems* 28 (2015).
- [43] A. Van Den Oord, O. Vinyals, et al. "Neural discrete representation learning." In: *Advances in neural information processing systems* 30 (2017).

- [44] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel recurrent neural networks." In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.
- [45] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." In: *arXiv preprint arXiv:1609.08144* (2016).
- [46] F.-F. L. J. S. Yeung. *Lecture 6: Training Neural Networks*. [http://cs231n.stanford.edu/slides/2017/cs231n\\_2017\\_lecture6.pdf](http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf). 2017.
- [47] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. "Deconvolutional networks." English (US). In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010 ; Conference date: 13-06-2010 Through 18-06-2010. 2010, pp. 2528–2535. ISBN: 9781424469840. DOI: 10.1109/CVPR.2010.5539957.