

Bachelor Degree In Computer Engineering
Computer Science

Thesis

User friendly denoising based on deep learning

Author

Urtzi Beorlegui Pascal

2022

Bachelor Degree In Computer Engineering
Computer Science

Thesis

User friendly denoising based on deep learning

Author

Urtzi Beorlegui Pascal

Directors

Ignacio Arganda, Nagore Barrena

Abstract

When taking images with microscopes, it is almost inevitable not to generate noise in the process. As a result, systems have been proposed to correct this problem. With the rise of deep learning techniques, it was to be expected that solutions based on these methods would emerge. Although there are many and very diverse proposals, the aim of this project is to analyse and present one of the most recent proposals: Divnoising. This method is based on an unsupervised model in a variational auto-encoder (VAE) framework incorporating also a noise model, which can be directly measured or bootstrapped from the noisy images.

Furthermore, this work contains several experiments done with the Divnoising model and different datasets. Finally, there is a guide explaining a user-friendly Jupiter Notebook designed for people with no experience in the field of programming and computing.

Contents

Abstract	i
Contents	iii
List of Figures	v
Table index	xi
1 Introduction	1
1.1 Motivation & Objectives	1
1.2 Objectives of this project	4
1.3 Project outline	5
2 Related and previous work	7
2.1 Classical approach	7
2.1.1 Spatial domain filtering	8
2.1.2 Transform domain filtering	9
2.1.3 Methods in other domains	10
2.2 State of the art	11
2.2.1 Block-matching and 3D filtering (BM3D)	11
2.2.2 Deep Learning	14
	iii

3	Methodology	25
3.1	The <i>Divnoising</i> model	25
3.2	ZeroCostDL4Mic notebook	28
4	Experiments	39
4.1	Dataset description	39
4.2	Noise Model	40
4.2.1	Calibration Images	40
4.2.2	Bootstrap	42
4.3	Evaluation metrics	43
4.3.1	SSIM (structural similarity) map	43
4.3.2	RSE (Root Squared Error) map	44
4.4	Training	45
4.5	Results	46
4.5.1	Mouse nuclei results	46
4.5.2	Convallaria results	56
5	Conclusions	59
Appendixes		
A	Gantt Diagram	63
B	Experiment 2 results	65
Bibliography		73

List of Figures

1.1	Noise types: (a) original image, clean signal, (b) original signal with Additive white Gaussian noise (AWGN), (c) original signal with added salt pepper noise, and (d) original signal with added speckle noise	2
2.1	Basic methodology of averaging filter. First, a central pixel and the size of the window are selected. Then, the pixels belonging to that window are averaged and the value of the central pixel is replaced by that average. Finally, the center of the window is moved and the process for the whole image is repeated. Source: [7].	8
2.2	Schematic of the process followed by the wavelet transformation filter for denoising. The original signal is taken and transformed into wavelet space. The filter is applied in this space (soft or hard) and the inverse is done to return to the original signal space. Source: [7].	10
2.3	Illustration of grouping blocks from noisy natural images corrupted by white Gaussian noise with standard deviation 15 and zero mean. Each fragment shows a reference block marked with “R” and a few of the blocks matched to it. Source: [6].	13
2.4	BM3D example results. Results for different values of σ for each step of the algorithm are shown In particular: a noisy image with a high variance (left), a basic estimation in an intermediate phase of the algorithm (center), and the final result (right). Source: [21].	13
2.5	Different layers of a CNN. From left to right: input layer with color image, convolutional layer to extract the features, max-pooling layers to simplify the model, fully connected layers to make the prediction and output layer. Picture taken from [8].	15

2.6	Example of the feature maps after a convolution layer of 6 kernels. Source: [28].	15
2.7	Example of a max-pooling with 2×2 filter and stride 2. Source: [28]. . .	16
2.8	Conventional vs blind-spot network. A conventional network (a) looks at the whole input in order to make the prediction, so it more information to predict the outcome. In addition to this they have a clean picture that is expected as a result. A blind-spot network (b), on the other hand, does not look at the pixel to be predicted in order not to learn the identity since there is no objective result. Source [6].	17
2.9	Blind-spot masking scheme in <i>Noise2Void</i> : (a) a noisy training image; (b) a magnified image patch from (a), a random value (blue rectangle) is copied into the central pixel to create the blind-spot this modified image is the input during training; and (c) the target patch corresponding to (b). The original input is used as target. The loss is only calculated for the blind-spot pixels masked in (b). Source: [6].	18
2.10	Encoder-decoder scheme. The encoder input is encoded into latent space. Then, the decoder reconstructs from the information obtained by encoding the input. Source: https://towardsdatascience.com/understanding-variational-auto	
2.11	Problem of encoding points in latent space. Let's imagine that we get a powerful encoder that manages to encode all input on the real axis (one input one real value). If we choose any point in the latent space, what we get when we decode it may not contain any information to recreate a signal. Source: https://towardsdatascience.com/understanding-variational-autoencoders-va	
2.12	Latent space of a VAE. The aim is to create a "gradient" over the information in the latent space and the intermediate solutions, thus being able to generate new information. Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 .	21
2.13	Training scheme. From bottom to top: The input data, x , is encoded to a normal distribution $\mathcal{N}(\mu_{z x}, \Sigma_{z x})$. After the encoder have learn the distribution the decoder take the distribution to sample from that latent space z . This way a new output is generated at the end. Source: https://web.eecs.umich.edu/justincj/slides/eecs498/498FA2019lecture19.pd37	22

3.1	<i>Divnoising</i> Training/Inference scheme. Top, in the training phase the latent space is adjusted using the noise model, minimizing the loss value (see Equation 3.6. Bottom, to predict the clean image, the model generates various prediction from the learned distribution, once the predictions are made, the model interpolate the result with all the predictions. Source [29].	26
3.2	<i>Divnoising</i> inference. We can observe the results returned by the model. From left to right: input (noisy) image, MMSE estimation, three predictions of the 1000 generated, result from another estimation method (MAP), and the ground truth image (without noise). Source: [29].	27
3.3	Example of an executable notebook cell. In order to execute the cell, click on the arrow icon in the upper left corner.	28
3.4	Example of an executable notebook cell. In this case the code is visible to see how each module works and to be able to make modifications.	29
3.5	Example of an executable notebook cell. The information output of an executed cell is shown, in this case of a cell that determines whether or not a GPU is available for training the model.	29
3.6	File site of the notebook. if a Google Drive session is connected the files and folders will be appear and will be accessible to the notebook.	30
3.7	If the user do not have a database to test the notebook, check "Test_Data"option to download a standard database. Else, the user can write the path to their own dataset.	30
3.8	Definable parameters for the training of the network. The noise model can be learned form calibration images or can be bootstrapped from noisy images.	31
3.9	Definable parameters for the training of the network. The variables can be adjust to optimize the results for different datasets.	32
3.10	Execution of a training process. In this example, it will train for 500 epochs.	33
3.11	Trained model download cell. The model will be downloaded in a .zip to the local computer.	34
3.12	Cell indicating the path to the model that will be evaluated.	34

3.13	Evaluation charts, the output of the extract metrics cell.	35
3.14	Cell output showing the evaluation metrics. top left original image, top right model prediction, bottom left SSIM map, bottom right RSE map. . .	36
3.15	Prediction cell. Running this cell will save the predicted images in the desired path.	37
4.1	Dataset example. (a) Mouse nuclei image example. (b) Convallaria image example.	40
4.2	left, A single calibration image. right, averaged calibration image. This will be the GT to generate the probability distribution of the Noise Model.	41
4.3	Example Probability distribution $P(x s)$ at signal 10525. The noise model thats is going to use to train the Divnoising model.	42
4.4	Experiment 1 loss Chart.	47
4.5	Experiment 1 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map wuth NRMSE and PSNR values	48
4.6	Experiment 2 loss Chart.	49
4.7	Experiment 2 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map wuth NRMSE and PSNR values	49
4.8	Experiment 3 loss Chart.	50
4.9	Experiment 3 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map wuth NRMSE and PSNR values	50
4.10	Experiment 4 loss Chart.	51
4.11	Experiment 4 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map wuth NRMSE and PSNR values	52
4.12	Experiment 5 loss Chart.	52

4.13	Experiment 5 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	53
4.14	Experiment 6 loss Chart.	54
4.15	Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	54
4.16	Comparison between the worse prediction and the best prediction. Left, worse, prediction from experiment 1 . Right, best, prediction from experiment 5.	55
4.17	Experiment 6 loss Chart.	57
4.18	Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	58
4.19	Comparison between the worse prediction and the best prediction. Left, worse, prediction from experiment 2 . Right, best, prediction from experiment 6.	58
A.1	Gantt diagram displaying the task with their respective duration. Research: Task 1, revise and study in depth the vae models in order to be able to implement Divnoising in a notebook. Task 2, after getting to know how a vanilla VAE works, read about the new method to be implemented and see why the differences it has make it superior for this job. Task 3, to study and look at different denoising technologies in order to complete the thesis and to understand the origin of this problem and how it has been tackled during this time. To better understand the motivation behind it. Task 3, research methods to be able to create GT and choose an option to implement on the notebook.	64
B.1	Experiment 1 loss Chart.	65
B.2	Experiment 1 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	66

B.3	Experiment 2 loss Chart.	66
B.4	Experiment 2 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	67
B.5	Experiment 3 loss Chart.	67
B.6	Experiment 3 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	68
B.7	Experiment 4 loss Chart.	68
B.8	Experiment 4 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	69
B.9	Experiment 5 loss Chart.	69
B.10	Experiment 5 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	70
B.11	Experiment 6 loss Chart.	70
B.12	Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values	71

Table index

4.1	Experiments of Divnoising network with Mouse Nuclei dataset. The column No is used to denote the number of experiment. The column Ns method represents the noise model generator method used in the experiment. The column Eps indicates the number of epochs the model has trained in each experiment. The next 3 columns indicate the evaluation metrics, SSIM (see Section 4.3.1), NRMSE (see Section 4.3.2) and PSNR (see Section 4.3.2). The bold line highlights the experiment with the best results.	47
4.2	Experiments of Divnoising network with Convallaria dataset. The column No is used to denote the number of experiment. The column Ns method represents the noise model generator method used in the experiment. The column Eps indicates the number of epochs the model has trained in each experiment. The next 3 columns indicate the evaluation metrics, SSIM (see Section 4.3.1), NRMSE (see Section 4.3.2) and PSNR (see Section 4.3.2). The bold line highlights the experiment with the best results.	56

CHAPTER 1

Introduction

Biomedical image processing is a very broad field [9], it covers biomedical signal gathering, image forming, picture processing, and image display to medical or biological diagnosis based on features extracted from images. Some basic image processing techniques including outlining, deblurring, noise cleaning, filtering, search, classical analysis and texture analysis. This thesis will focus on denosing techniques and will describe various classical techniques before analyzing a state of the art technique in depth.

In order for these systems to be effective for biomedical applications fast and efficient image processing methods have to be developed. With that in mind, methods are created giving fast and reliable results. Since a late or wrong diagnosis can have fatal consequences. With that in mind, Divnoising is developed, a fast, efficient and easy to use noise removal method for professionals in fields other than computer science.

1.1 Motivation & Objectives

One of the biggest challenges in image processing is image denoising. Images are often taken in poor lighting and atmospheric conditions. Therefore, it is essential to have effective tools to solve or alleviate this problem.

Noise is usually generated during image acquisition, encoding, transmission and processing. The noise comes in many forms, Additive White Gaussian Noise (AWGN), impulse

noise (salt and pepper), quantisation noise, Poisson noise and speckle noise are most frequently discussed noises in the literature [7].



(a) Original



(b) AWGN



(c) Salt & pepper noise



(d) Speckle noise

Figure 1.1: Noise types: (a) original image, clean signal, (b) original signal with Additive white Gaussian noise (AWGN), (c) original signal with added salt pepper noise, and (d) original signal with added speckle noise

Due to the sensitivity of biological sample to the radiation damage, the low dose imaging conditions used for electron microscopy result in extremely noisy images. The processes of digitization, image alignment, and 3D reconstruction also introduce additional sources of noise in the final representation [20]. Innumerable investigations have been made to solve this problem, although it has not yet been completely solved, there have been great improvements in this field. It is a very important field of study because of its applications, especially in the fields of medicine and biology. It is of vital importance to be able to clearly differentiate the objects seen in the original signal. A misdiagnosis in these cases can be fatal, so having effective and fast denoising methods is of vital importance.

Along the same lines, microscopy is also fundamental in the medical area of study, which is the science of using physical systems to view small objects. Those systems are known as microscopes. Originally, microscopes were plainly optical devices, using finely ground lenses to expand the resolution of samples. More recently, the field of microscopy has started to use technologies such as electron beams or even physical probes to produce the signal. But, as we have said before, being such small objects, they are sensitive to radiation and extreme caution must be taken with the doses, and these small doses is what causes more noise than usual to be generated in biomedical microscope images, noise that has to be eliminated later by denoising techniques.

The objective of denoising is to reconstruct the original image by removing the added noise. This process plays an important role in a wide range of applications. Such as image restoration, visual tracking, image segmentation, image classification, etc. Noise often makes these functions difficult or even impossible. Obtaining the original image is crucial for good performance in these algorithms. Many algorithms have been proposed, each with its strengths and weaknesses. Even so, this problem remains open, especially when the images have been taken in very poor conditions where the noise is very high.

Particularly in the context of microscope imaging, noise comes from three sources [2]:

1. Dark noise: corresponds to electronic noise, generated by agitation of electrons. It follows a Poisson distribution and affects mainly to the background.
2. Photon noise: generated by inaccuracy in counting the number of photons, due to the nature of photon emission, this form of noise is inherent in all optics.
3. Readout noise: generated by the inaccuracy of the chips in transforming the image into digital form. It follows a normal distribution with zero mean.

As we can see, there are many difficulties in obtaining the original image taken by a microscope. But it is vital that techniques are developed to overcome these problems. In order to advance in other fields such as biology or medicine. It is important to be able to detect shapes in an image to be able to diagnose or study with clarity, noise greatly hinders these tasks.

1.2 Objectives of this project

As we will discuss below, noise removal techniques have evolved over the course of history [2]. Giving solid results. We see this progress especially with the rise of deep learning, whose results greatly improve the results obtained by traditional methods.

With this in mind, the objective of this work is to study and analyze a modern deep learning model designed for denoising, especially for denoising microscope images. One of the main objectives, is the generation of clean images generated only from noisy images taken directly. The architecture to be used is a convolutional variational autoencoder, Divnoising [30]. This model is used to deal with two major difficulties. First, in order not to have to generate the clean image from a single clean image or solution. In this case, several images are generated and then following some criteria the new image is interpolated. Second, the need to learn without labels, this network learns the noise distribution only with images taken directly. In this way, we have a totally unsupervised system that generates results that competes and sometimes outperforms supervised models.

Once the model has been studied, another objective is to implement the model in such a way that people unfamiliar with computers science and machine learning methods can make use of this powerful technique. Furthermore, it will be implemented in such a way that it does not consume any of the user's resources. That is to say, all the computational process will be done on an external server. At all times the objective is to facilitate the use of this tool to all users.

For these reasons, there is a project called [ZeroCostDL4Mic](#)¹ that is born with this philosophy. We will implement the Divnoising model in a Google colabotary notebook following the structure of the work of this project.

In summary, the objectives of this project are as follows:

¹[ZeroCostDL4Mic project](#)

1. Study and analysis of different noise removal systems. Especially deep learning models.
2. Study, analysis and use of the Divnoising convolutional architecture.
3. Implementation of the model in an environment that is easy to access and use for non-expert users.
4. Contribution to the ZeroCostDL4Mic project with the developed implementation.

1.3 Project outline

This report will be structured as follows. First, Section 2, we will talk about some techniques that have been proposed throughout history to deal with this problem, thus introducing State of the art methods, methods that give the best results nowadays. In the following Section 3 we will explain the model that will be analyzed in this work, that is, we will explain how Divnoising works and why it gives good results. In addition, in this section we will show how the notebook developed for the easy use of this powerful tool is organised. Once this model has been explained, the results of the experiments, Section 4 carried out to corroborate the efficiency of the model and the notebook developed will be shown. In addition to the results, this section will explain the datasets used to recreate the experiments, the evaluation metrics used and how the noise model necessary for the correct functioning of the model is generated. Finally, Section 5, we will discuss the conclusions reached after having studied and experimented with the model.

Related and previous work

The major challenge of the algorithms implemented for denoising is to remove as much of the noise as possible without losing the most significant details of the original image. Over time, the resolution of the devices has increased, so the number of image sensors per unit area increases. Therefore, cameras capture noise more often. We will look at the broad solutions that have been given to this problem and algorithms that have evolved to achieve satisfactory results for users.

No image restoration technique is universal [14], that is, for all types of noise models. A proper estimation of noise and knowledge about the best available toolset at hand for a particular noise type is mandatory for efficient implementation of this image restoration task.

2.1 Classical approach

As we have already mentioned, the aim of denoising is to obtain the original image from the acquired noisy image. To deal with this problem, different methods have been implemented over the years [7].

starting from simple methods such as filters, transformations or statistical methods to more complex methods as unsupervised models that learns from noisy signals to generate the clean images.

We can model noise in a simple way as follows:

$$f(x) = u(x) + n(x) \quad x \in X, X \subset Z^2$$

where $u(x)$ is the true signal and $n(x)$ denotes noise at location x .

2.1.1 Spatial domain filtering

Filtering in the spatial domain [35], has been used for a long time because of its low complexity and good results. Filters are ultimately nothing more than the inverse degradation model of the image. They are designed for a single type of degradation and noise model, although some have multiple uses.

The simplest type of filtering is averaging or mean or box filtering. This generates an output for each pixel as the mean of the neighborhood pixels of a given range. However, these types of filtering tend to generate an undesirable amount of smoothing on edges and loss of detail.

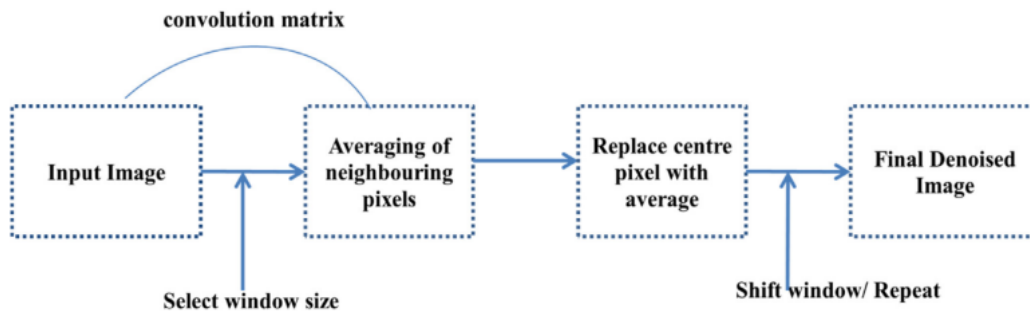


Figure 2.1: Basic methodology of averaging filter. First, a central pixel and the size of the window are selected. Then, the pixels belonging to that window are averaged and the value of the central pixel is replaced by that average. Finally, the center of the window is moved and the process for the whole image is repeated. Source: [7].

i) Mean Filters

This filter acts on an image by smoothing it. It reduces the intensity variations between the adjacent pixels. The mean filter is one of the simplest filter types. The filter window replaces the center pixel value with the average value of the near pixels [39]. Its methodology is depicted in Figure 2.1.

ii) Wiener Filter

The formulation of the Wiener filter is based upon the assumption that the imaging system is linear and stationary [15]. Furthermore, it is also assumed that the noise is additive with constant variance. With these assumptions, the filter minimizes the mean-square error between the original image and the restored image.

iii) Median Filter

The median filter follows the window principle too. The median of the window is calculated and then the value of the central pixel is replaced with this value.

iv) Lee's sigma filter

An enhanced version of Lee's sigma filter [22] is derived for filtering of images affected by multiplicative noise with speckle statistics. A new edge reserving filter which is called the mean and median hybrid (MMH) filter is developed to achieve all kinds of noise removal, as well as edge preservation. A hybrid filter that consists of a nonlinear filter and a fuzzy weighted linear filter is derived to reduce the mixed noise. The adopted the first part uses the statistics techniques are used to remove the large magnitude impulsive noise then the second part uses a weighted average linear filter to remove additive Gaussian noise and small ripple impulsive noise. Three variants are combined in a trimmed mean filter by fuzzy set to get better noise smoothing results [34].

2.1.2 Transform domain filtering

In contrast with spatial domain filtering methods, transform domain filtering methods first transform the given noisy image to another domain, and then apply a denoising procedure on the transformed image according to the different characteristics of the image and its noise (larger coefficients denote the high frequency part, i.e., the details or edges of the image, smaller coefficients denote the noise). The transform domain filtering methods can be subdivided according to the chosen basis transform functions, which may be data adaptive or non-data adaptive [24].

Lately, there has been a lot of research on these methods, especially the wavelet transforms, a simple method that offers very good results especially in image denoising. It involves applying the wavelet transform to the original data by thresholding the wavelet coefficient and inverting the wavelet transform afterwards (see Figure 2.2).

Here, the threshold plays an important role in the denoising process. Finding an optimum threshold is a tedious task [33]. Smaller threshold value leaves higher number of coeffi-

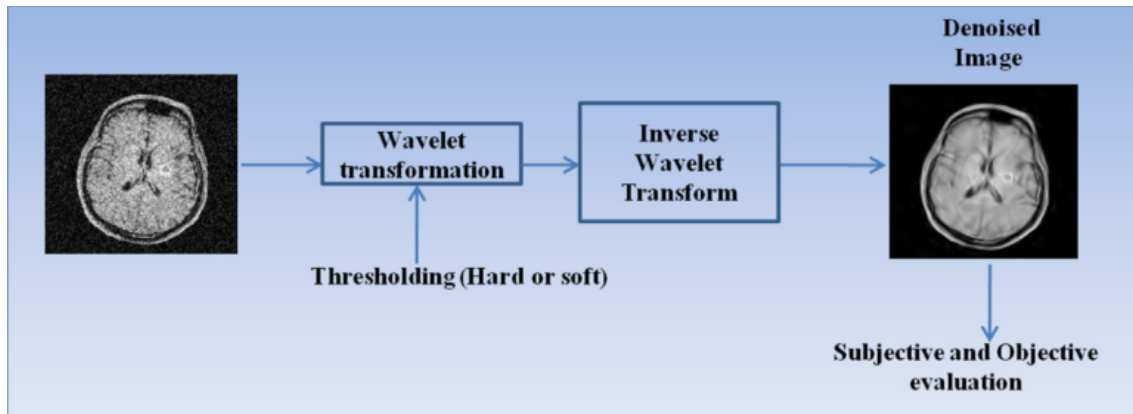


Figure 2.2: Schematic of the process followed by the wavelet transformation filter for denoising. The original signal is taken and transformed into wavelet space. The filter is applied in this space (soft or hard) and the inverse is done to return to the original signal space. Source: [7].

cients associated with noise information. On the other hand, larger threshold values will shrink the signal feature, over-smoothing or blurring the image.

2.1.3 Methods in other domains

Apart from spatial filters and domain transformations, many other methods for image denoising have been investigated. As well as methods based on statistics and random fields. These methods aim at constructing spatially homogeneous models adaptable to varying levels of signal with the help of parametrization of the variables which are self-random in nature.

For these methods, there are two effective techniques. Namely, the principal component analysis (PCA), which treats only the data information given by the second-order statistics, and the independent component analysis (ICA) which comes as an extension to PCA to give better performance, by living up to high order statistics (the case of the most natural images). The main idea of both statistical techniques, PCA and ICA, is to use an orthogonal decomposition to separate linearly as much as possible the correlated data into independent sub-sets. In the non-adaptative approaches there are many methods due to the variety of basic functions such as, wavelets, wave atoms, curvelets, contourlet, wedgelets, and bandelets, which transform the image to the frequency domain. From those, the oldest, the most popular, and the dominant one ("wavelet"), is highlighted in the following sections. Recently, in the transform domain, a new efficient method called block-matching and 3D filtering (BM3D) and developed by Dabov *et al.* (see [subsection 2.2.1](#)) use a spar-

se 3D transform by grouping similar 2D-blocks in the image into a 3D-array (grouped). In this case, the image is represented in the transform domain by many 3D-groups. Then, the spectrum is shrunken to separate the noise from other features. This method has shown great efficiency in removing noise compared to several techniques, but in the case of high level of noise, it gives less performance. For that, a bounded BM3D method was proposed based on the basis BM3D to exceed this limitation. Furthermore, many techniques were derived from the BM3D such as in [4], his paper proposes a bounded BM3D scheme which focuses on the optimal choice of shrinkage operator, and the two algorithms in [13] CD-BM3D and iterative CD-BM3D used the complex domain [3].

In this area, Hossein Rabbani proposed a novel noise reduction algorithm [31] where the noisy captured 3-D data are first transformed by a discrete complex wavelet transform (DCWT). Using a nonlinear function, the data is modeled as the sum of the clean data plus additive Gaussian or Rayleigh noise. A mixture of bivariate Laplacian probability density functions is used for the clean data in the transformed domain.

2.2 State of the art

The denoising techniques have evolved over the years, and not only that, new and more powerful tools such as neural networks have emerged. In this section, we will explain some of the denoising tools that nowadays provide the best results, such as block-matching or neural networks like convolutional neural networks (CNNs) or variational autoencoders (VAEs), which despite needing a large amount of data to learn, once trained they produce very good results.

2.2.1 Block-matching and 3D filtering (BM3D)

As mentioned in the previous section, BM3D [6] is an algorithm based on an enhanced sparse representation in transform domain. This algorithm assembles similar 2-D fragments of the image into a 3-D array called groups. In our work, we will use this tool to generate a pseudo-ground truth, in case there are no calibration images from the microscope to use as reference. We will go more in depth on this topic when we analyze the model used in the project.

The transform-domain denoising methods typically assume that the true signal can be well approximated by a linear combination of some few elements. With this purpose in mind,

what they propose is grouping. They call grouping to putting together similar fragments of d dimensions into a $d + 1$ dimensional structure called a group. The idea of this is to be able to use a higher dimensional filtering of each group, thus being able to better exploit the similarities (correlation, affinity, etc.) between the grouped fragments in order to estimate the true signal. This approach is called *collaborative filtering*.

This grouping can be done in many ways [11]: K-means clustering, self-organizing maps, fuzzy clustering, and others. The similarity between two fragments is usually done as the inverse of some distance, i.e. the more similar two fragments are, the smaller this distance will be.

To create the different clusters in BM3D, they use what they call grouping by matching. In the techniques we have mentioned, the objective is to create different clusters, where each object belongs to a single cluster. That is, disjoint classes are created, which causes that not all objects in each class are treated in the same way. Those that are closer to the centroid will be better represented than those that are further away from the centroid. Furthermore, creating clusters is often a time-consuming process since recursive procedures are used, which is quite computationally demanding. For these reasons, a simpler method is used, grouping similar fragments by matching. Matching is a method for grouping fragments similar to the reference fragment. This makes the same fragment belong to the group of two different references. The fragment whose similarity (distance) is greater than a certain threshold belongs to the group of that reference. Any signal can be used as a reference, thus building several groups. The threshold that is set could be considered as the diameter of the group.

Block-matching (BM) is a particular matching approach, it is used to find similar blocks. In Figure 2.3, we see a few reference blocks and the ones that they match.

Once we have the groups formed, we have to estimate the value of the reference. This could be done by averaging all the fragments of the group. But, this is only an efficient technique when all the fragments of the group are identical and in nature this is not the case. Therefore, another way of estimation has to be found. To solve this problem, the collaborative filtering by shrinkage in transform domain is presented. For this purpose, first a linear transformation of dimension $d + 1$ is applied. Then, we shrink (e.g. by soft and hard thresholding) the transform coefficients to attenuate the noise. Finally, the linear transform is inverted, to produce estimates of all grouped fragments (see Figure 2.4.

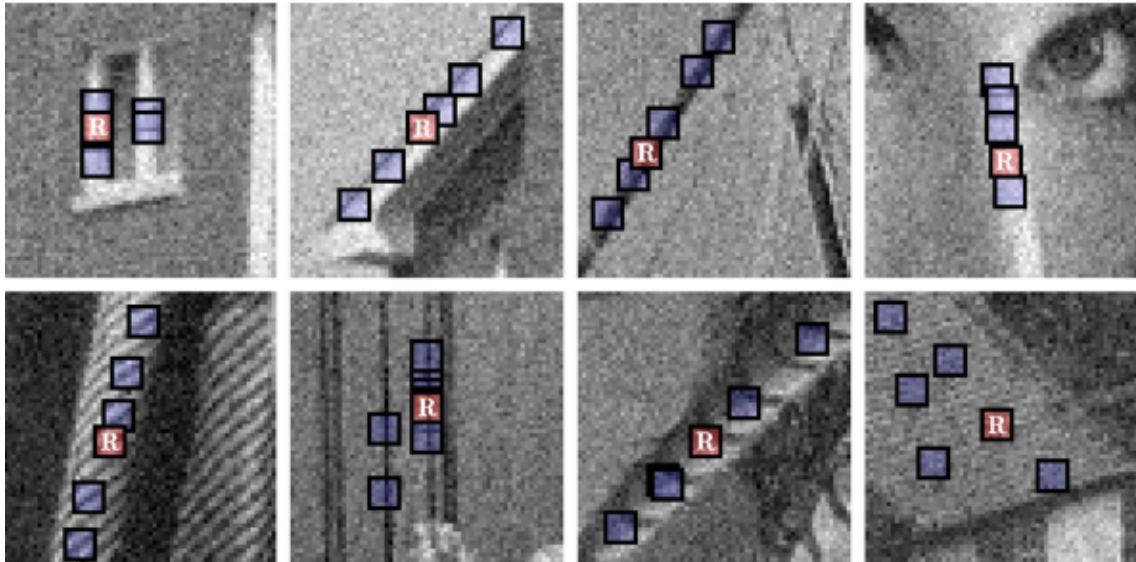


Figure 2.3: Illustration of grouping blocks from noisy natural images corrupted by white Gaussian noise with standard deviation 15 and zero mean. Each fragment shows a reference block marked with “R” and a few of the blocks matched to it. Source: [6].

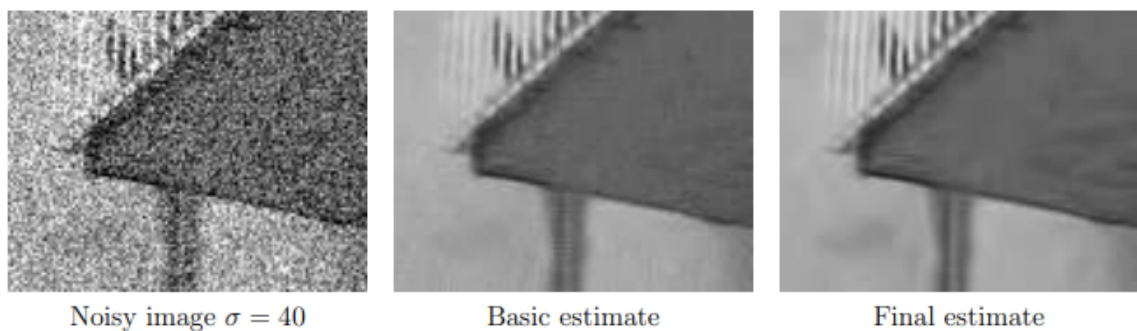


Figure 2.4: BM3D example results. Results for different values of σ for each step of the algorithm are shown. In particular: a noisy image with a high variance (left), a basic estimation in an intermediate phase of the algorithm (center), and the final result (right). Source: [21].

2.2.2 Deep Learning

Deep learning techniques have emerged as powerful solutions in all fields of computer vision, and the field of denoising is no exception [36]. Many deep learning models have been presented that offer state-of-the-art results.

First of all, there are several types of machine (and thus deep) learning methods. In general terms, machine learning methods consist of supervised, semisupervised and unsupervised learning methods [36].

Supervised learning methods use a label to know whether the obtained result is accurate. This way, the parameters can be updated and the network learns. For example, with the following denoising model:

$$y = x + \mu$$

where x , y and μ represent the given clean image, noisy image and AWGN of standard deviation σ , respectively. From the equation above and Bayesian knowledge, it can be seen that the learning of parameters of the denoising model relies on pair $\{x_k, y_k\}_{k=1}^N$, where x_k and y_k denote the k th clean image and noisy image, respectively. Also, N is the number of noisy images. This processing can be expressed as $x_k = f(y_k, \theta, m)$, where θ is the parameters and m denotes the given noise level.

Brief overview of CNNs

CNNs were mainly developed for use in image processing and computer vision. This is why it works especially well with images, such as noise removal. The main reason why this type of network is used in imaging is because of the number of connections. For example, with a color picture of only 32×32 , if we want to connect it to a single neuron in the hidden layer, we would have $32 \times 32 \times 3$ weights. Therefore, if we intend to develop a deep network, it would be unfeasible.

CNNs generally consist of three main types of neural layers [8]: convolutional layers, pooling layers and fully connected layers. Each layer plays a different role in the process (see Figure 2.5).

Instead of using layers of neurons, we have layers of kernels of dimension n , which are used to convolve the whole image and the feature maps that are created after convolving

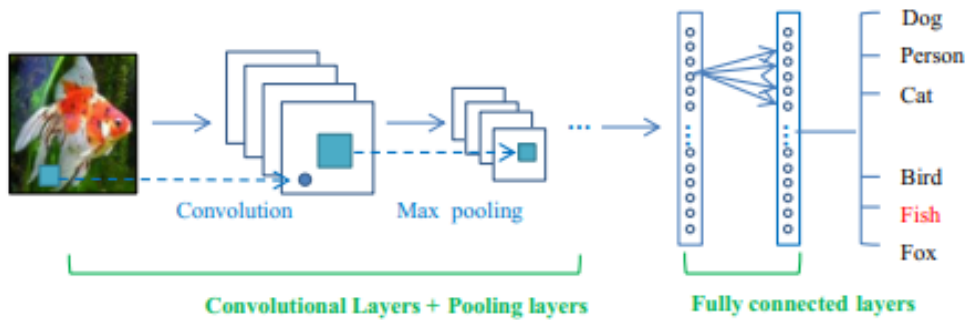


Figure 2.5: Different layers of a CNN. From left to right: input layer with color image, convolutional layer to extract the features, max-pooling layers to simplify the model, fully connected layers to make the prediction and output layer. Picture taken from [8].

the original one (see Figure 2.6). Convolution has many advantages: the mechanism of weight sharing in the same feature map reduces the number of parameters, the local connectivity learns the correlations between neighboring pixels, and it provides invariance to the location of the object.

Different convolution kernels can be concatenated to generate different feature maps. The more kernels we apply, the more the channels of the image will grow (remember that an RGB image has size $H \times 3$, including 3 channels).

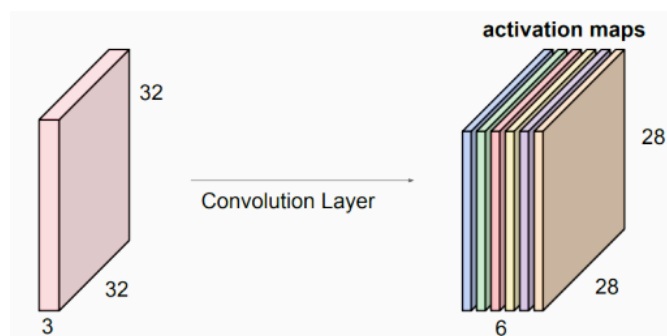


Figure 2.6: Example of the feature maps after a convolution layer of 6 kernels. Source: [28].

The idea of pooling is down-sampling in order to reduce the complexity of the further layers [10]. It would be something like decreasing the resolution, thus preventing over-fitting. Pooling does not affect the number of channels in that layer and can be done in different ways. The most common ways are max pooling or average pooling. The image is partitioned into n sub-regions and either the maximum value (see Figure 2.6) of the sub-region or the average value is chosen depending on the type of pooling.

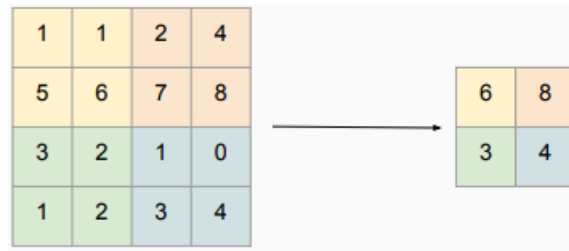


Figure 2.7: Example of a max-pooling with 2×2 filter and stride 2. Source: [28].

The fully connected layer works similarly to the neurons in a traditional neural network. Each node is directly connected to all nodes in the previous and next layer. This layer is usually where the most parameters are concentrated (90% of the total parameters of the model) and the most difficult to train. The function of the fully connected layer is decision making. The training of these networks is identical to that of traditional networks with a forward pass (prediction) and a backward pass (learning).

Denoising networks

In this case, we are interested in having our network fed one (noisy) image and return another (clean) image. The image denoising task must be formulated as a learning problem to train the convolutional network. Since we assume access to clean, noise-free images, we implicitly specify the desired image processing task by integrating a noise process into the training procedure. Specifically, we assume a noise process $n(x)$ operating on an image x_i drawn from a distribution of natural images X . If we consider the full convolutional network to be some function [12] F_ϕ with free parameters ϕ , then the parameter estimation problem is to minimize the reconstruction error of the images subjected to the noise process: $\min_\phi \sum_i (x_i - F_\phi(n(x_i)))^2$

As mentioned above, this type of network requires a pair of images. Normally, the noise is generated artificially from the noise-free image. In practical cases, it is difficult to obtain noise-free images. In order to generate a model that is able to learn only from images with noise, *Noise2Noise* [23] was implemented. However, sometimes it is not possible to obtain pairs of noisy images, and to solve this problem and go one step further, *Noise2Void* [18] was created, a CNN that learns from single noisy images only.

In this *Noise2Void*, it is proposed to derive the two parts of the training, the input and the target, from a single noisy image x^i . If the model were to learn from a complete image

patch, it would learn the identity. In particular, it would simply take the central pixel of the input and replicate it in the prediction (see Figure 2.8a).

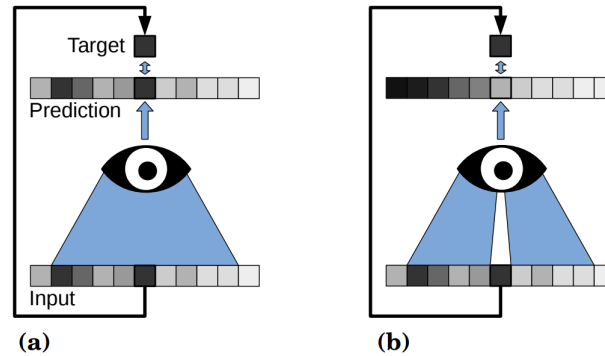


Figure 2.8: Conventional vs blind-spot network. A conventional network (a) looks at the whole input in order to make the prediction, so it more information to predict the outcome. In addition to this they have a clean picture that is expected as a result. A blind-spot network (b), on the other hand, does not look at the pixel to be predicted in order not to learn the identity since there is no objective result. Source [6].

To train without a reference (clean) image we do the following. We assume that the receptive field $x_{RF(i)}$ of this network has a blind-spot in the center (see Figure 2.8b). The CNN prediction s_i is affected by the surrounding pixels except for the pixel x_i at that location. This blind-spot network has less information than a conventional CNN would have, so it is expected to have a slightly reduced accuracy. However, since only one pixel is subtracted from the receptive field, reasonable results are expected. The main advantage of using this blind-spot architecture is the impossibility to learn the identity. Since we assume the noise to be pixel-wise independent given the signal, the neighboring pixels have no information about the noise that may be in pixel x^i . Even if the original value cannot be obtained, since the signal is assumed to contain statistical dependencies, the network can still estimate the signal s_i by looking only at the surroundings of pixel x_i (see Figure 2.9).

Variational Autoencoders (VAEs)

As we have just seen, the most recent denoising methods implement architectures that are able to learn from noisy images alone. As we have said, most of the time the pictures that are accessible are noisy images. Therefore, the way forward in the field of denoising is this, training with noisy images. The main approach of this project, *Divnoising*, shares this vision, although unlike *Noise2Void*, it does not use a CNN-based architecture, but a variational autoencoder (VAE) based one.

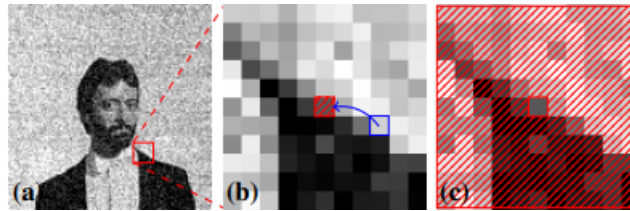


Figure 2.9: Blind-spot masking scheme in *Noise2Void*: (a) a noisy training image; (b) a magnified image patch from (a), a random value (blue rectangle) is copied into the central pixel to create the blind-spot this modified image is the input during training; and (c) the target patch corresponding to (b). The original input is used as target. The loss is only calculated for the blind-spot pixels masked in (b). Source: [6].

Unlike CNNs, which are discriminative models (they only generate one solution), VAEs are generative models. They seek to learn the probability distribution in order to create new solutions. Before explaining how a VAE works, there are some terms that are useful to know beforehand.

VAE is an autoencoder whose encoding distribution is regularized during the training in order to ensure that its latent space has the property that allows to generate new data [32].

First, we will explain what the encoder is. The encoder is the process that generates new features from old features. The job of the decoder would be to retrieve the old features from the ones created by the encoder. The features created by the encoder (from the initial space) are also called latent space. When encoding the initial space, information can be lost, which means that it is often impossible to recover all the information in the decoding process (See Figure 2.10).

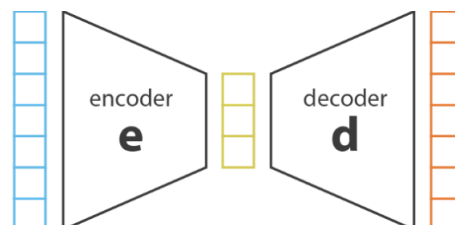


Figure 2.10: Encoder-decoder scheme. The encoder input is encoded into latent space. Then, the decoder reconstructs from the information obtained by encoding the input. Source: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>.

As we have mentioned, the decoder generates a new space, but we do not know which

space is the optimal one in order not to lose information when generating it with our encoder. The VAEs are neural networks, so the main idea is to make both encoder and decoder two different networks: one to generate the latent space and the other to return that information to the initial space. These two neural networks seek to reduce the reconstruction error, which can be done via gradient descent over the parameters of the joint network, parameters that we will see later.

If we only have a decoder and an encoder, we cannot generate new information. The only thing we will do is compress and decompress information that we already know and, in many cases, this information will no longer be useful due to the loss of information when going into the decoder and trying to reconstruct it again. It is true that we can take a random point from the latent space and pass it through the decoder to generate new data. However, there is another factor to take into account: the regularity of the latent space. This is a complicated issue that depends a lot on the distribution of the information in the initial space, the dimensions of the latent space and the architecture of the encoder. Therefore, it is very difficult to ensure a priori that the encoder will organize the information in a way that is compliant with the idea of generating information in this way.

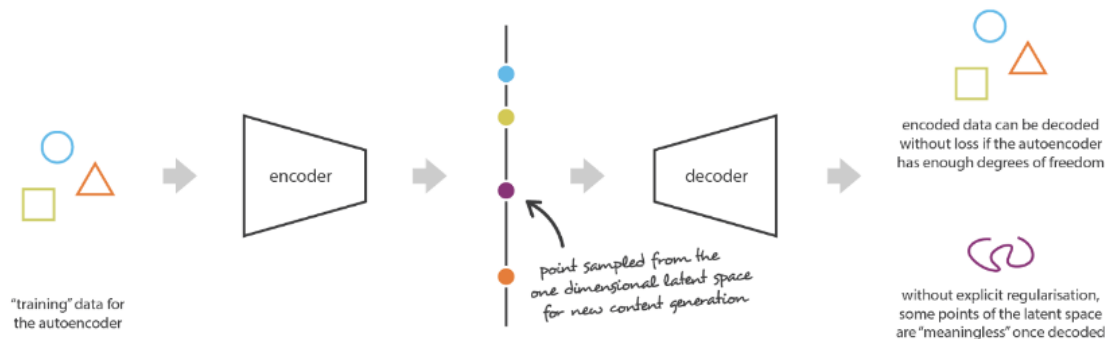


Figure 2.11: Problem of encoding points in latent space. Let's imagine that we get a powerful encoder that manages to encode all input on the real axis (one input one real value). If we choose any point in the latent space, what we get when we decode it may not contain any information to recreate a signal. Source: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>.

In Figure 2.11, there is a simple example of what would happen if we try to generate information in this way, choosing a random point in the latent space. Let's supposed we have an encoder (to put a simple and illustrative case) that converts the given information to the real axis (each data represented as a real value). We could encode and decode without any loss (overfitting). If we choose a point (the purple point, new information) from that real space, what it would generate has no value. This happens because the autoencoder

is trained to have as little loss as possible, without taking into account the latent space organization. To avoid this problem, a regularization is added to the process.

To regularize the autoencoder, VAEs are created. They work in the same way as an autoencoder, but with one difference: instead of encoding the input at a single point in the latent space, it encodes that information as a distribution over the latent space. Therefore, the model would be trained as follows. First, the input is encoded as a distribution over the latent space. Second, a point in the latent space is sampled with that probability. Third, the sampled point is decoded and the reconstruction error can be computed. Finally, the reconstruction error is backpropagated through the network.

The distribution in the latent space is a normal distribution, so that it can be trained to return the mean and the covariance matrix describing that Gaussian.

$$\mathcal{N}(\mu, \sigma^2)$$

The reason why the input is encoded as a distribution with variance is that in this way, we force the encoder to form a distribution close to the standard distribution. The regularization term in the loss formula is expressed as the Kulback-Leibler divergence (KL divergence) between the returned distribution and the standard Gaussian.

This regularization (see Figure 2.12) aims to make the generative process possible and it can be simplified into two properties: continuity (two nearby points in the latent space cannot give totally different results) and completeness (for a chosen distribution, a sampled point should give a "meaningful" content). Both the covariance matrix and the mean that the encoder returns have to be regularized. As mentioned before, this is done by forcing the distribution returned by the encoder to resemble a normal distribution. With this regularization term, we get the distributions to "overlap" and be centered, in order to achieve the properties mentioned above. This, as with any regularization term, makes the reconstruction error larger. However, the trade off between reconstruction error and KL divergence can be adjusted.

We have seen how autoencoders work, so we can define the encoder this way, $p(z|x)$, and the decoder on the other hand, $p(x|z)$, being z the encoded information in the latent space. The regularization of the latent space appears in the encoder representation, encoded representations in the latent space are indeed assumed to follow the prior distribution $p(z)$. By Bayes' theorem we know that

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du} \quad (2.1)$$

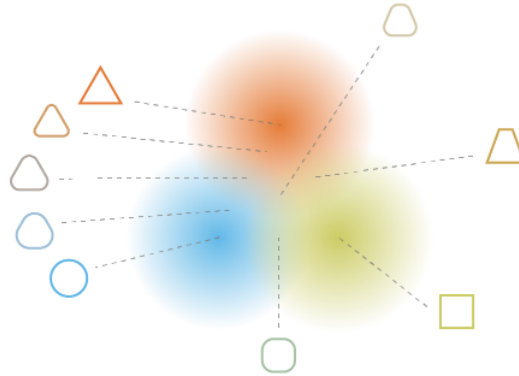


Figure 2.12: Latent space of a VAE. The aim is to create a "gradient" over the information in the latent space and the intermediate solutions, thus being able to generate new information. Source: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>.

That distribution is intractable. So we want to approximate $p(z|x)$ using another distribution $q(z)$. If we make the assumption that $q(z)$ is a standard Gaussian distribution (tractable) and that $p(x|z)$ is a Gaussian distribution defined by a function f whose covariance matrix has the form of an identity matrix multiplied by a constant c , we have that:

$$p(z) \equiv \mathcal{N}(0, I)$$

$$p(x|z) \equiv \mathcal{N}(f(z), cI), \quad c > 0 \quad (2.2)$$

we want to minimize $KL(q(z)||p(z|x))$

$$KL(q(z)||p(z|x)) = -\sum_z q(z) \log \frac{p(z|x)}{q(z)} \quad (2.3)$$

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x, z)}{p(x)} \quad (2.4)$$

$$KL(q(z)||p(z|x)) = -\sum_z q(z) \log \frac{p(z, x)}{q(z)} + \underbrace{\sum_z q(z) \log p(x)}_{\log p(x)} \quad (2.5)$$

$$\underbrace{\log p(x)}_{\text{constant}} = \underbrace{KL(q(z)||p(z|x))}_{\text{Minimize this}} + \underbrace{\sum_z q(z) \log \frac{p(x, z)}{q(z)}}_{\mathcal{L}} \quad (2.6)$$

\mathcal{L} is usually called Evidence Lower-Bound(ELBO) The optimization objective of the VAEs, is the ELBO [17]. The ELBO is derived through Jensen's inequality [26]. The key idea is to minimize KL by maximizing \mathcal{L} as $p(x)$ is a known constant value. As we have said the KL calculation is intractable so the only way we have to minimize it is to maximize \mathcal{L} .

In summary:

$$\log p(x) \geq \mathcal{L} = \underbrace{E_{z \sim q(z)} \log p(x|z)}_{\text{Data reconstruction}} - \underbrace{KL(q(z)||p(z))}_{\text{regularization}} \quad (2.7)$$

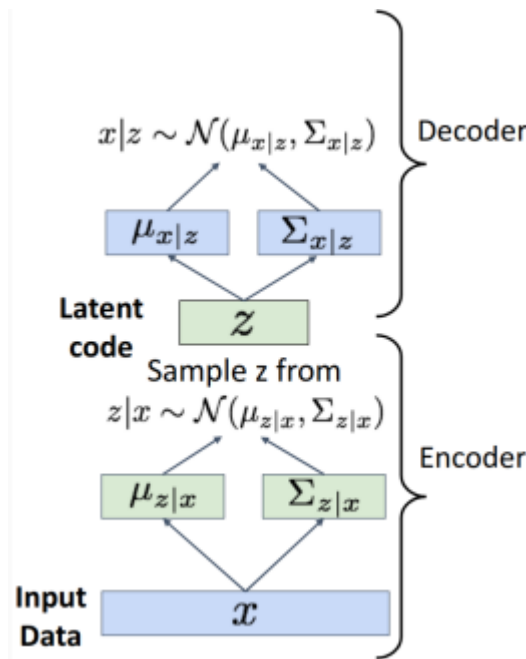


Figure 2.13: Training scheme. From bottom to top: The input data, x , is encoded to a normal distribution $\mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$. After the encoder have learn the distribution the decoder take the distribution to sample from that latent space z . This way a new output is generated at the end. Source: <https://web.eecs.umich.edu/justincj/slides/eecs498/498FA2019lecture19.pd37>.

Taking all this into account, the train process would take the following form: we enter the data through the encoder to obtain the latent space distribution. Then the encoder should match $p(z)$ (KL see Equation 2.3). Then, the model has to sample z from the encoder output, and then pass that sample through the decoder. The resulting data should be similar to the original data (ELBO see Equation 2.7).

Once we have trained the model, we can modify data and generate new one.

With a trained VAE model we can generate new information. As an input we put the information we want to manipulate. Once we have the latent space distribution, we take the z -sample we have generated, $z \sim \mathcal{N}(\mu_{z|x}, \sigma_{z|x}^2)$. When we have sampled z we can modify some dimension of z and then pass the modified z through the decoder. Finally, all you have to do is sample new data $x \sim (\mu_{x|z}, \sigma_{x|z}^2)$.

Methodology

3.1 The *Divnoising* model

First, unsupervised content-aware image restoration (CARE) methods [19] emerged. They can, enabled by sensible assumptions about the statistics of imaging noise, learn a mapping from noisy to clean images, without ever seeing clean data during training. Some of these methods additionally include a probabilistic model of the imaging noise [29] to further improve their performance. Note that such *denoisers* can directly be trained on a given body of noisy images.

As mentioned above, the methodology chosen in this project is *Divnoising* [29].

Divnoising is build on the VAE setup but interprets it from a denoising-specific perspective. It supposed that the images have been created from a clean signal via a known noise model. They replace the generic normal distribution (see Equation 2.2), with a known noise model

$$PNM(x|s) = \prod_i^N PNM(x_i|s_i) \quad (3.1)$$

We get

$$p_{\theta}(x|z) = PNM(x|s) = \prod_i^N PNM(x_i|s_i) \quad (3.2)$$

with the decoder now predicting the signal

$$g_{\theta}(z) = s \quad (3.3)$$

With $p(z)$ and the noise model, the decoder now describes a full joint model for all three variables:

$$p_{\theta}(z, x, s) = PNM(x|s)p_{\theta}(s|z)p(z), \quad (3.4)$$

For a given z^k , the decoder describes a distribution over noisy images ($p(x|z)$). The corresponding clean signal s^k , is deterministically defined. Hence, $p_{\theta}(s|z)$ is a Dirac distribution [38] centered at $g_{\theta}(z)$.

The noise model is usually thought to factorize as a product of pixels, implying that the corruption, given the underlying signal, is occurring independently in each pixel as

$$p(x|s) = \prod_i^N PNM(x_i|s_i) \quad (3.5)$$

The noise model ($PNM(x_i|s_i)$) can either be measured with paired calibration images, or bootstrapped from noisy data. In the current project, the option to bootstrap the noise model has been implemented with BM3D. The implementation gives you the option to use calibration images, but, if you don't have them available you can always use just the dirty images and through BM3D bootstrap the noise model. We will talk about this later.

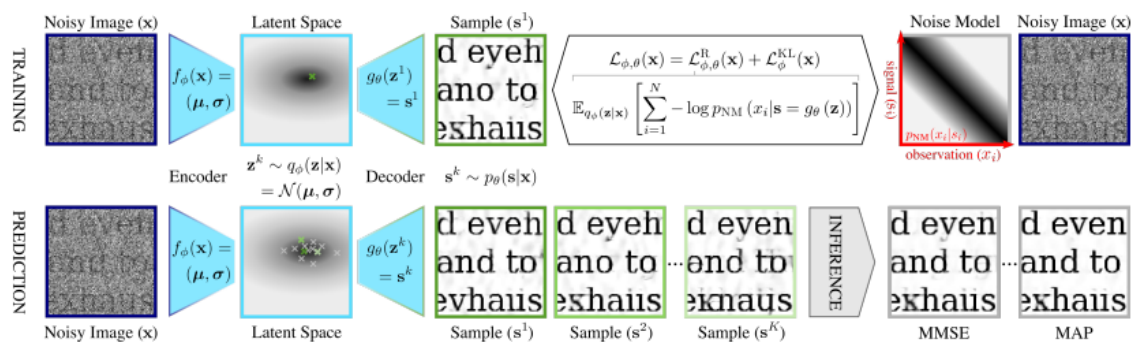


Figure 3.1: *Divnoising* Training/Inference scheme. Top, in the training phase the latent space is adjusted using the noise model, minimizing the loss value (see Equation 3.6). Bottom, to predict the clean image, the model generates various prediction from the learned distribution, once the predictions are made, the model interpolate the result with all the predictions. Source [29].

This is said to be true for Poisson noise and camera readout noise. We will refer to the probability $PNM(x_i|s_i)$ of observing a particular noisy value x_i at a pixel i given clean signal s_i .

Having the previous equation into account (see Equation 3.5), the reconstruction loss becomes

$$\mathcal{L}_{\Phi\theta}^R(x) = E_{q_{\Phi}(z|x)} \left[\sum_{i=1}^N -\log p(x_i|s = g_{\theta}(Z)) \right]. \quad (3.6)$$

Apart from this modification, the method used by the traditional VAE for training is followed (see Section 2.2.2). The only thing that is modified is the distribution that the training data model learns. Therefore, it can be assumed that it complies with the properties of the vanilla VAE. That is, the model describes the distribution of the training data while the latent space approximates a distribution that generates results.

The *Divnoising* model can be used to generate images from $p_{\theta}(x)$, but what we are interested in is to use it for denoising. Therefore, we aim to obtain $p(s|x)$, that is, the distribution of the possible clean images s given a noisy observation x . With a well-trained model, it is possible to obtain samples s^k from an approximate posterior, feeding the noisy image x into the encoder, drawing samples $z^k \sim q_{\Phi}(z|x)$ and decoding the samples via the decoder to get $s^k = g_{\theta}(z^k)$. Given a set of posterior samples s^K for noisy images x , there are different consensus estimates to be able to infer. For example, approximate the minimum mean square error (MMSE) estimate by averaging the samples s^k . Another option is to find the maximum a posteriori (MAP). Once we have this distribution, to get the clean images the the mean shift algorithm [5] with a decreasing bandwidth is followed.

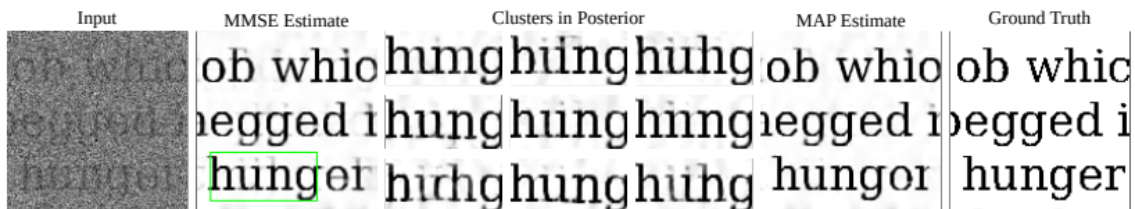


Figure 3.2: *Divnoising* inference. We can observe the results returned by the model. From left to right: input (noisy) image, MMSE estimation, three predictions of the 1000 generated, result from another estimation method (MAP), and the ground truth image (without noise). Source: [29].

3.2 ZeroCostDL4Mic notebook

Finally, the last task for this thesis was the creation of a user-friendly notebook in order to create a tool accessible to everyone. In this way, an interface is created to be able to use this tool even without any knowledge of programming or deep learning. An interface, in which the users enter the images to be denoised and after a simple process they get clean images. In other words, to make the network usable for everyone, always following the philosophy and aesthetics of the notebooks developed by the ZeroCostDL4Mic project [37].

Remember that all the processes that are run are executed on a server external to the user's computer, making it even more accessible as it does not require computational power. Normally it takes time and power to train a network. This simplifies this problem and the user does not have to worry about his computer equipment.

To begin with, there are two types of cells in notebooks. Text cells and executable cells. The text cells are merely informative, while the executable cells have an icon in the upper left corner of the cell that must be clicked to run the program.

1. Complete the Colab session

▶ Play the cell to connect your Google Drive to Colab

- Click on the URL.
- Sign in your Google Account.
- Copy the authorization code.
- Enter the authorization code.
- Click on "Files" site on the right. Refresh the site. Your Google Drive folder should now be available here as "drive".

Figure 3.3: Example of an executable notebook cell. In order to execute the cell, click on the arrow icon in the upper left corner.

These executable cells are composed of code and text. Normally the code is not hidden and is visible to everyone. In this case, it is decided to hide it as it is not revealing to the end user. However, if you want to see the code behind the text in these cells, you can do so by selecting "show code". For example, in the Figure 3.3 the code is not visible. But, as we have said, when selecting "show code" the code appears and it would look like in the Figure 3.4.

```

▶ #@markdown ##Play the cell to connect your Google Drive to Colab

#@markdown * Click on the URL.

#@markdown * Sign in your Google Account.

#@markdown * Copy the authorization code.

#@markdown * Enter the authorization code.

#@markdown * Click on "Files" site on the right. Refresh the site. Your Google Drive folder sl

# mount user's Google Drive to Google Colab.
from google.colab import drive
drive.mount('/content/gdrive')

```

Figure 3.4: Example of an executable notebook cell. In this case the code is visible to see how each module works and to be able to make modifications.

Once these cells are executed, they usually have an output, but not always. For example, the Figure 3.3 does not show any output. But, if you run the cell that determines whether or not you have a GPU for training, you will see the entire output as in Figure 3.5. If a graphics card is available, the graphics card information is printed.

```

📄 You have GPU access
Sun Jun 19 19:48:54 2022
+-----+
| NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2      |
+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M. |
+-----+-----+
|    0   Tesla T4            Off      | 00000000:00:04:0 Off |                    0 |
| N/A   56C    P0      29W / 70W   | 264MiB / 15109MiB |         0%      Default |
|                                           N/A |
+-----+-----+

+-----+
| Processes: |
| GPU   GI    CI          PID   Type   Process name                      GPU Memory |
|      ID    ID              |                 |           Usage |
+-----+-----+
Tensorflow version is 2.8.2

```

Figure 3.5: Example of an executable notebook cell. The information output of an executed cell is shown, in this case of a cell that determines whether or not a GPU is available for training the model.

After executing that first cell, the next executable cell gives the possibility of connecting the session to Google Drive. This allows the notebook to access all the files saved in the users Google Drive account, so that if the user intends to use a dataset for that session it can be easily accessed. To log in to google to access the information stored in drive, the cell shown in the figure below is executed (Figure 3.3). Once played the cell a new browser window will appear, select the google account and select "allow".

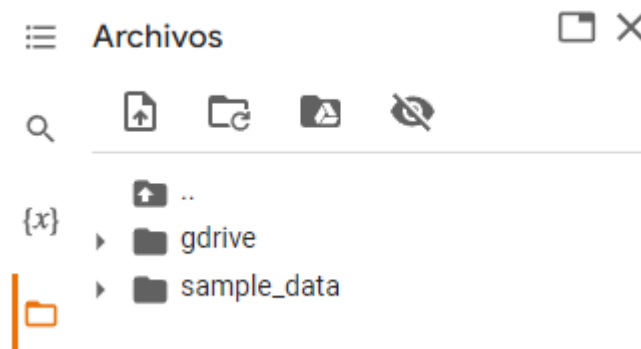


Figure 3.6: File site of the notebook. if a Google Drive session is connected the files and folders will be appear and will be accessible to the notebook.

However, if the user does not have a dataset with which to test the network, there is a default option that downloads one of the two datasets mentioned above (see Section 4.1), the mouse nuclei dataset. The images can be downloaded in section 3 of the booklet.

Paths for training data

`Test_Data`: If checked demo images will be downloaded

`Images_source`: This is the path where you have the images you want to use to train the model. The images will be all in a single .tif file.

`dataName`: This is the name under which the generated files will be saved.

▶ Noisy Images Path

`Test_Data`:

`Images_source`: " data/mouse_nucleid/example2_digital_offset300.tif

`dataName`: " mouse_nucleid

[Mostrar código](#)

Figure 3.7: If the user do not have a database to test the notebook, check "Test_Data"option to download a standard database. Else, the user can write the path to their own dataset.

The next step in the notebook is to indicate the parameters to decide which type of noise

model will be used to train the network. The user can either select a trained noise model or learn it from scratch. The noise model can be learned from noisy signals (bootstrapped) or can be learned using microscopy calibration images. If training with noisy images is selected, there is a parameter to choose the percentage of images that will be used to train the model. Training time increases with the number of images. Using calibration images is less time consuming, although it also takes some time to train the noise model.

▶ Noise model options:

Trained Noise model:

Trained_Noise_Model:

Noise_Model_Source: " data/mouse_nucleid/NoiseModel/GMMNoiseModel_nuclei_3_2_calibration.npz"

Training a Model:

percentage_images_for_training:

Create a model from noisy images:

Create_From_Images:

Use_Calibration_Images:

Create a model from calibration images:

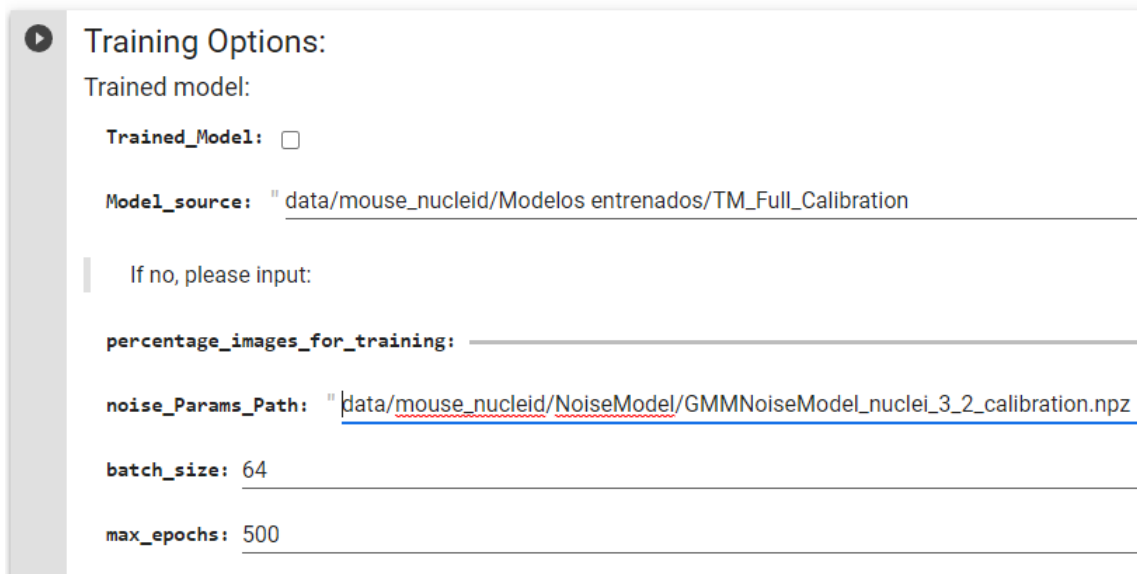
If yes, please input:

Calibration_source: " sdata/mouse_nucleid/CalibrationImages/edgeoftheslide_300offset.tif"

[Mostrar código](#)

Figure 3.8: Definable parameters for the training of the network. The noise model can be learned from calibration images or can be bootstrapped from noisy images.

The following step in the notebook is to indicate the parameters with which the network is going to be trained. Each parameter is defined in a text cell shown in Figure 3.9. The parameters can be calibrated to obtain the best time-quality results. As shown in the Section 4, sometimes training longer does not give much better results. If the user has a previously trained model, there is the possibility to enter the path of this model and by selecting the option "Trained_Model", the step of training the model is saved. This is the most expensive step of the whole denoising process.



▶ Training Options:

Trained model:

Trained_Model:

Model_source: " data/mouse_nucleid/Modelos entrenados/TM_Full_Calibration

If no, please input:

percentage_images_for_training: _____

noise_Params_Path: " data/mouse_nucleid/NoiseModel/GMMNoiseModel_nuclei_3_2_calibration.npz

batch_size: 64

max_epochs: 500

Figure 3.9: Definable parameters for the training of the network. The variables can be adjusted to optimize the results for different datasets.

If it is not the case that you have a trained model, the training process will start. It lasts up to the epochs that have been assigned to it in the parameters. Otherwise, if the loss of the model does not decrease in many epochs, it patiently stops the training process and returns the model with the lowest loss that has been achieved in the whole process.

4.1. Download the new model (optional)

Run the cell below to download a copy of the trained model to use it in the future. In order not to have to train a model again

Download Trained Model

[Mostrar código](#)

Figure 3.11: Trained model download cell. The model will be downloaded in a .zip to the local computer.

After training and saving the model, it is time to evaluate the model. In this notebook different metrics are used to evaluate the quality of the results returned by the model (see Section 4).

```
▶ Extract the model losses  
  
model_p: " data/convallaria/Modelos entrenados/TM_50_bootstrap  
  
Mostrar código
```

Figure 3.12: Cell indicating the path to the model that will be evaluated.

Once we have a trained model, either a newly trained model or a previously trained model, we have to indicate the path as shown in 3.12. This extracts the values needed for the subsequent calculation of the loss graphs and the structural similarity (SSIM) and root squared error (RSE). the next executable cell displays two charts: the training loss and validation loss vs. epoch number in linear scale and the training loss and validation loss vs. epoch number in log scale (see Figure 3.13).

▶ Play the cell to show a plot of training errors vs. epoch number

[Mostrar código](#)

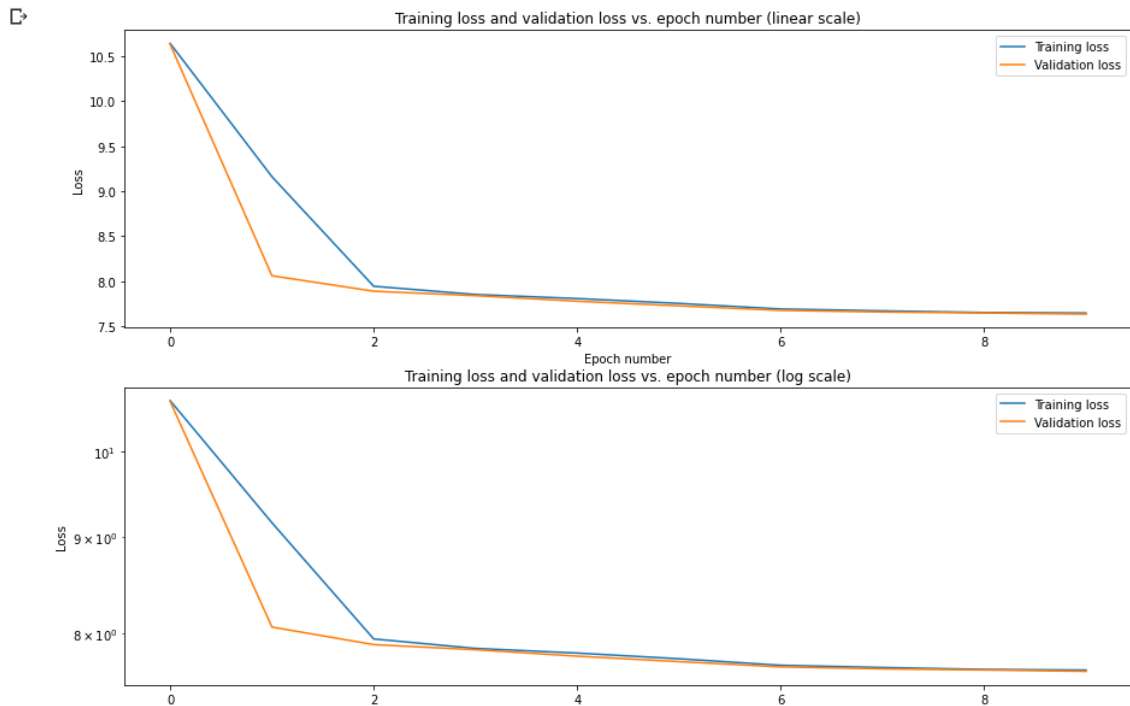


Figure 3.13: Evaluation charts, the output of the extract metrics cell.

In addition to the graphs, RSE and SSIM metrics can also be generated (see Figure 3.14). These metrics help to evaluate the performance of the model. The original image and the predicted image are also displayed. Thus giving the opportunity to make a subjective assessment by the human eye.

Generate Metrics

[Mostrar código](#)

```
Running QC on: example2_digital_offset300.tif
'\n#Root Squared Error between GT and Prediction\nplt.subplot(3,3,9)\nplt.axis('off')\nplt.tick_params(\n    axis='both',    # che
d\n    bottom=False,    # ticks along the bottom edge are off\n    top=False,    # ticks along the top edge are off\n    left=False,    # ticks along the left edge are off\n    right=False,    # ticks along the right edge are off\n    labelbottom=False,\n    labelleft=False)\n\nimg_RSE_GTvsPrediction = plt.imshow(img_RSE_GTvsPrediction[0], cmap = cmap,
rediction',fontsize=15)\nplt.xlabel('NRMSE: '+str(round(NRMSE_GTvsPrediction,3))+', PSNR: '+str(round(PSNR_GTvsPrediction,3)),fontsize=15)
```

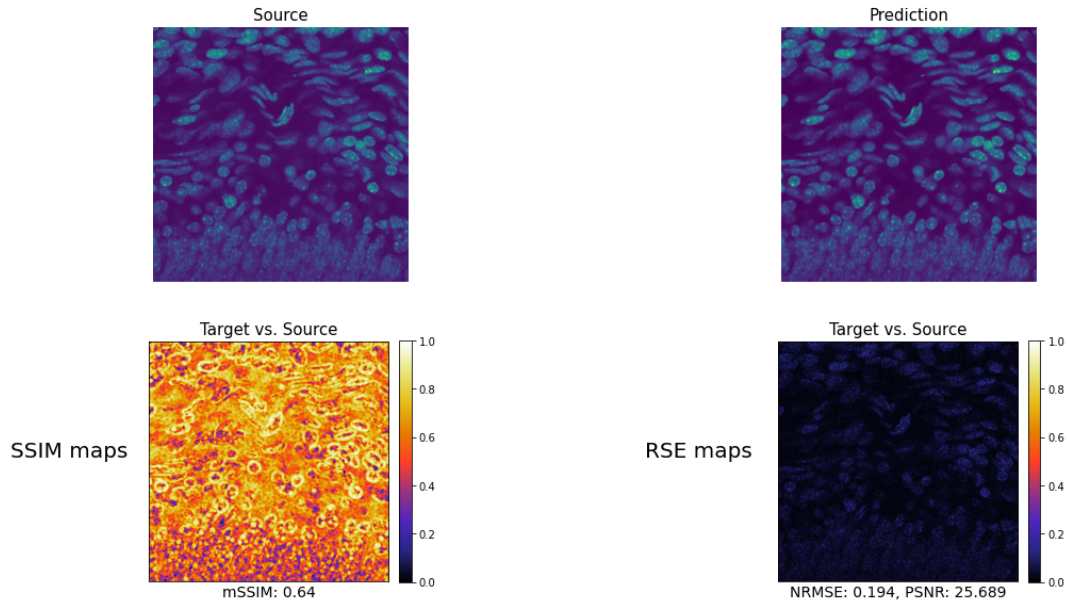


Figure 3.14: Cell output showing the evaluation metrics. top left original image, top right model prediction, bottom left SSIM map, bottom right RSE map.

Finally, there is the cell that extracts the results. If the cell in the Figure 3.15 is executed, the noisy images will be processed by the network and the results predicted by the network will be stored in the path chosen by the user. The user can choose the number of samples he wants to use to make the inference and produce the final noise-free image as explained in the Figure 3.2.

▶ Prediction options:

`num_samples:` 10

`export_results_path:` "./denoised_results"

`fraction_samples_to_export:` 0

`export_mmse:`

`tta:`

[Mostrar código](#)

Figure 3.15: Prediction cell. Running this cell will save the predicted images in the desired path.

Experiments

The aim of this chapter is to see the functionality of the Divnoising network and its performance under different scenarios and to show the results it produces. A set of images (see dataset) with different training parameters will be processed to see the difference in performance under different conditions. To make this comparison we will use different metrics (see evaluation) that we will explain later

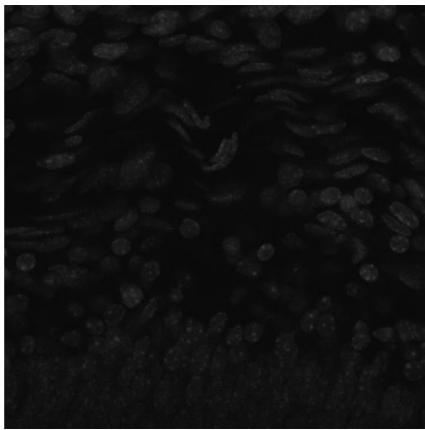
4.1 Dataset description

We use public microscopy datasets (see Figure 4.1) which show realistic levels of noise, introduced by the respective optical imaging setups. The FU-PN2V Convallaria¹ [19] data, consists of 100 noisy calibration images (intended to generate a noise model), and 100 images of size 1024×1024 showing a noisy Convallaria section. The FU-PN2V Mouse nuclei² [29] data is composed of 500 noisy calibration images and 200 noisy images of size 512×512 showing labeled cell nuclei. The FU-PN2V Mouse actin (Prakash et al., 2020) data from the same source consists of 100 noisy calibration images and 100 noisy images of size 1024×1024 of the same sample, but labeled for the protein actin.

For this experiment we will only use the unlabeled images and they will all be 512×512 . The 1024×1024 images will be discarded. By using more than one dataset we will be able

¹convallaria

²Mouse nuclei



(a) MouseNuclei



(b) Convallaria

Figure 4.1: Dataset example. (a) Mouse nuclei image example. (b) Convallaria image example.

to observe better the real performance of the network. To verify that it does not work well only with a specific type of image, it is useful for use with any type of image.

4.2 Noise Model

Image restoration is the task of estimating a clean signal $s = (s_1, \dots, s_N)$ from a corrupted observation $x = (x_1, \dots, x_N)$, where s_i and x_i refer to the respective pixel intensities. The corrupted x is thought to be drawn from a probability distribution $P_{NM}(x|s)$, which we call the observation likelihood or the noise model. Contrary to existing methods, Divnoising is designed to capture the inherent uncertainty of the denoising problem by learning a suitable posterior distribution. Formally, the posterior we are interested in is $p(s|x) \propto p(x|s)p(s)$ and depends on two components: the prior distribution $p(s)$ of the signal as well as the observation likelihood $p_{NM}(x|s)$ introduced above. While the prior is a highly complex distribution, the likelihood $p(x|s)$ of a given imaging system (camera/microscope) can be described analytically.

4.2.1 Calibration Images

We will use pairs of noisy calibration observations x_i and clean signal s_i (created by averaging these noisy, calibration images) to estimate the conditional distribution $p(x_i|s_i)$

The noise model is a characteristic of the camera or microscope and not of the sample.

The calibration images can be anything which is static and imaged multiple times in succession. Thus, the edge of slide works as well. We can either bin the noisy - GT pairs (obtained from noisy calibration images) as a 2-D histogram or fit a GMM (Gaussian Mixture model) distribution to obtain a smooth, parametric description of the noise model.

Using the raw pixels x_i , and our averaged GT s_i , we are now learning a GMM based noise model. It describes the distribution $p(x_i|s_i)$ for each s_i .

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

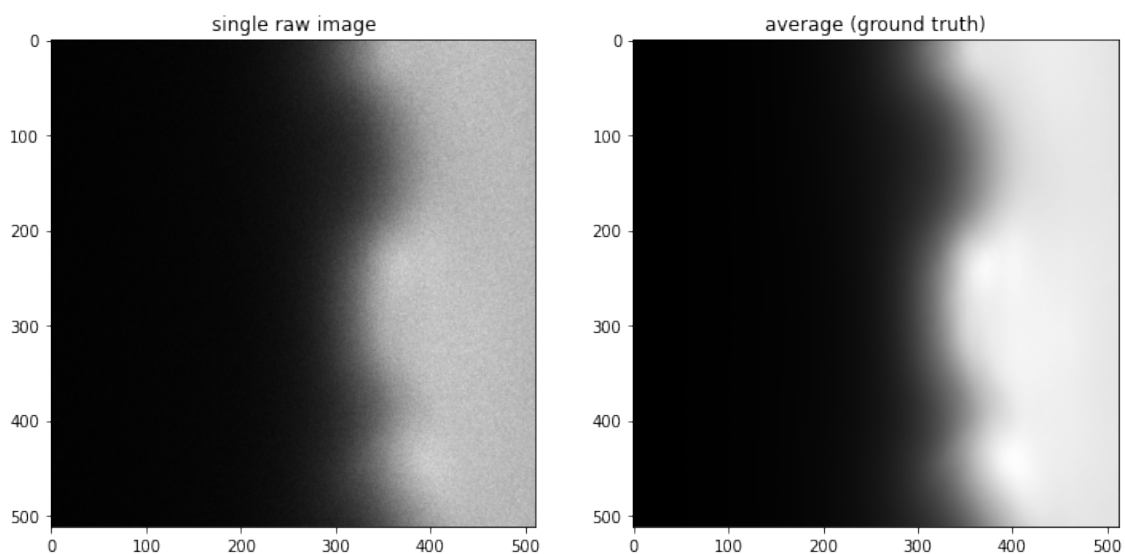


Figure 4.2: left, A single calibration image. right, averaged calibration image. This will be the GT to generate the probability distribution of the Noise Model.

Using the raw pixels x_i , and our averaged GT s_i , we are now learning a GMM based noise model. It describes the distribution $p(x_i|s_i)$ for each s_i .

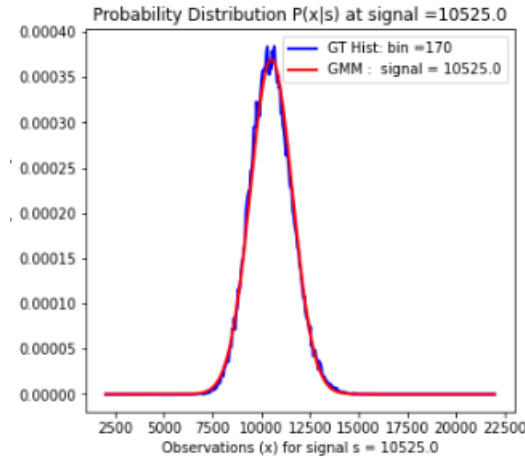


Figure 4.3: Example Probability distribution $P(x|s)$ at signal 10525. The noise model that is going to use to train the Divnoising model.

4.2.2 Bootstrap

Here we assume that we do not have access to calibration data to create a noise model for training DivNoising. In this case, we use an approach called Bootstrapping to create a noise model from noisy data itself. The idea is that we will first use the unsupervised denoising method BM3D to obtain denoised images corresponding to our noisy data. Then we will treat the denoised images as pseudo GT corresponding to the noisy data and use the pair of noisy images and corresponding BM3D denoised images to learn a noise model.

As will be explained in the Section 3, the opportunity to create a noise model simply from dirty images is given. Because, usually no calibration images are available to generate the noise model. It has been decided to use BM3D for its results and speed. Although any denoising technique that generates a GT in order to create the GMM would work.

DivNoising when using bootstrapped noise model generally gives better results compared to Noise2Void denoising. Also, unlike Noise2Void, we additionally obtain diverse denoised samples corresponding to any noisy image unlike Noise2Void.

As described above, we will use the denoising results obtained by BM3D and treat them as pseudo GT corresponding to our noisy data. Following this, we will use the pair of noisy images and corresponding BM3D denoised images to learn a noise model. You can use any other denoising method as well and treat their denoised result as pseudo GT to learn a noise model for DivNoising training.

The result is a noise model that will help us in making predictions with Divnoising. The process from this point on is the same as if we do it with calibration images. Having the GT images and the dirty images. A GMM based noise model is created.

4.3 Evaluation metrics

In order to evaluate the results of this work, two different techniques will be used, in addition to the subjective visual assessment. The objective methods are SSIM (structural similarity) and RSE (root Squared Error) map.

4.3.1 SSIM (structural similarity) map

The SSIM metric is used to evaluate whether two images contain the same structures. It is a normalized metric and an SSIM of 1 indicates a perfect similarity between two images. Therefore for SSIM, the closer to 1, the better. The SSIM maps are constructed by calculating the SSIM metric in each pixel by considering the surrounding structural similarity in the neighbourhood of that pixel (currently defined as window of 11 pixels and with Gaussian weighting of 1.5 pixel standard deviation)

The SSIM [27] index consists of three sub-indices: the luminance index, contrast index and structure index. The luminance reflects the intensity of the object recorded as the pixel value in the image. The contrast reflects the difference in luminance or extent of the luminance variation. The structure reflects the Pearson correlation of the luminance between two images. If we have image X and image Y, the luminance, contrast, and structure comparison functions of each point in images can be defined, respectively, as:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \quad (4.1)$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad (4.2)$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \quad (4.3)$$

where μ_x and μ_y , σ_x and σ_y , and σ_{xy} are the local means, standard deviations and cross-

covariance of image X and image Y, respectively. C1, C2 and C3 are the regularization constants that have very small values to avoid the extreme small denominator. μ_x , σ_x and σ_{xy} are computed by:

$$\mu_x = \frac{1}{N} \sum_{i=1}^N x_i \quad (4.4)$$

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \quad (4.5)$$

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y) \quad (4.6)$$

Where i is the index of the points into a local area, N is the total number of points in the same local area. The local area is a group of neighbors, the shape of the neighborhood is variable of the filter type. Finally, the SSIM index combines three sub functions and has the following form:

$$SSIM(x, y) = [l(x, y)]^\alpha [c(x, y)]^\beta [s(x, y)]^\gamma \quad (4.7)$$

4.3.2 RSE (Root Squared Error) map

This is a display of the root of the squared difference between the normalized predicted and target or the source and the target. In this case, a smaller RSE is better. A perfect agreement between target and prediction will lead to an RSE map showing zeros everywhere (dark).

NRMSE

NRMSE[25] gives the average difference between all pixels in the images compared to each other. Good agreement yields low NRMSE scores. Is defined as follows:

$$NRMSE = \frac{\sum (S_i - O_i)^2}{\sum_i^2} \quad (4.8)$$

where O_i are observed values and S_i are simulated values. A small value of NRMSE identifies a numerical simulation in good agreement with the field observations.

PSNR

The term peak signal-to-noise ratio (PSNR) [1] is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. Because many signals have a very wide dynamic range, (ratio between the largest and smallest possible values of a changeable quantity) the PSNR is usually expressed in terms of the logarithmic decibel scale. The higher the score the better the agreement.

The dimensions of the correct image matrix and the dimensions of the degraded image matrix must be identical. The mathematical representation of the PSNR is as follows:

$$PSNR = 20 \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right) \quad (4.9)$$

Where MSE(Mean Squared Error) is:

$$MSE = \frac{1}{mn} \sum_0^{m-1} \sum_0^{n-1} \|f(i, j) - g(i, j)\|^2 \quad (4.10)$$

Where f represents the matrix data of our original image, g represents the matrix data of our degraded image, and m and n represents matrix rows and columns.

4.4 Training

The training process was done in a notebook making use of the services provided by Google Colab. Some of the parameters varied with the purpose of searching the best hyperparameters, but several are inherent to the network:

- Optimizer : Adam [16]. It is defined as an extension to Stochastic gradient descent method.
- Loss Function : The one explained in Section 3, in Equation 3.6.
- Patience : Patience is set to 100 epochs.

When the network goes a long time without a significant improvement (threshold), there is a mechanism in place for the model to stop training without iterating over all the epochs

that were indicated at the beginning of the training. The function has a patience parameter, as we have said before it is a high value, 100 epochs, but it is a value that we cannot change because it is implemented directly in the model.

The following section shows the results of several experiments, which we will perform. In these experiments, we want to observe the behavior of the model when two parameters are changed. The first one is the number of epochs we allow the model to iterate and the other is the source of the model noise. We will see if it affects having microscope calibration images or if bootstrapping from noisy images give similar results.

4.5 Results

In this section the different experiments will be presented. Experiments will be done training 6 different models, changing the number of epochs they have to train and the noise model provided for training. These experiments will be repeated with both datasets (see Section 4.1) to see the behavior of the model with different data and to be able to make a better analysis of the behavior of the model.

4.5.1 Mouse nuclei results

The first experiments were done with the Mouse Nuclei dataset, a dataset composed of mouse nucleus images. The Table 4.1 shows the results obtained, results that we will analyze in more detail later.

EXPERIMENT 1

As we can see in the figure below, the loss tended to decrease even more. If he had been allowed to train with more epochs the loss would have decreased and the results would have been better. We can also observe that in the SSIM map (see Figure 4.5) there are many dark blue areas, indicating that in that area it differs quite a lot from the image you want to obtain. In this case, it would indicate the presence of a lot of noise in the lower part of the image that the model has not been able to eliminate in this experiment.

No.	Ns method	#Eps	SSIM \uparrow	NRMSE \downarrow	PSNR \uparrow
1	Bootstrap	10	0.578	0.208	24.419
2	Calibration	10	0.594	0.204	24.723
3	Bootstrap	50	0.624	0.198	25.38
4	Calibration	50	0.639	0.194	25.738
5	Bootstrap	500	0.644	0.193	25.858
6	Calibration	244	0.641	0.194	25.719

Table 4.1: Experiments of Divnoising network with Mouse Nuclei dataset. The column No is used to denote the number of experiment. The column Ns method represents the noise model generator method used in the experiment. The column Eps indicates the number of epochs the model has trained in each experiment. The next 3 columns indicate the evaluation metrics, SSIM (see Section 4.3.1), NRMSE (see Section 4.3.2) and PSNR (see Section 4.3.2). The bold line highlights the experiment with the best results.

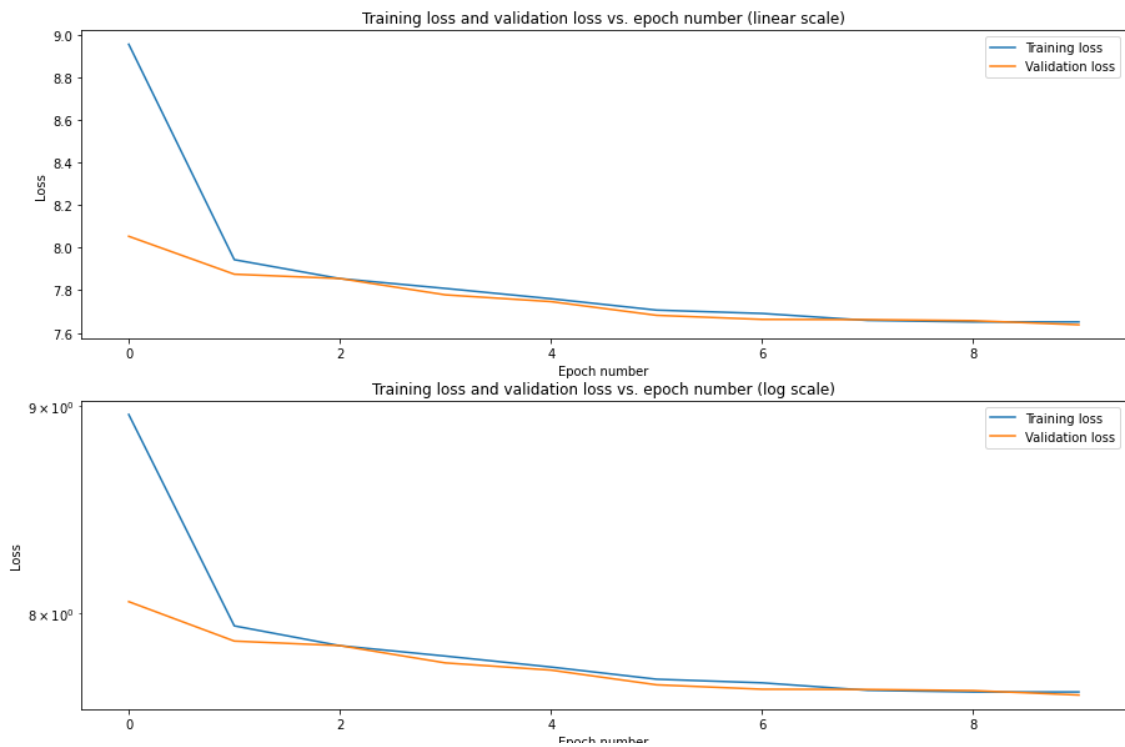


Figure 4.4: Experiment 1 loss Chart.

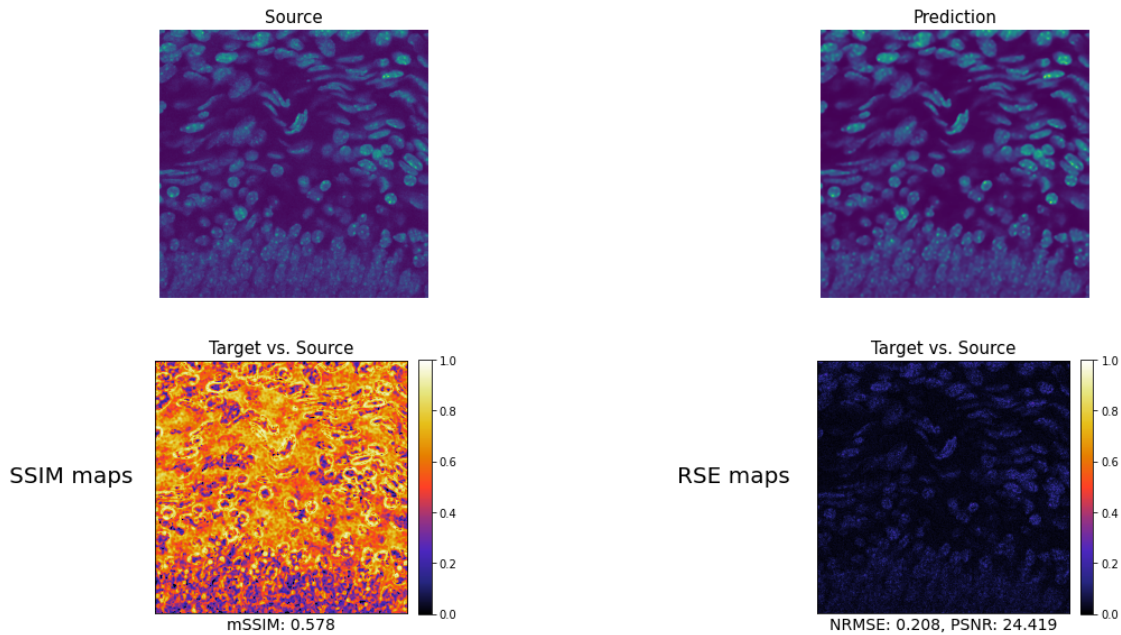


Figure 4.5: Experiment 1 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

EXPERIMENT 2

The results of this second experiment are very similar to the results of the first experiment. What changed from one experiment to the other is the noise model used to train the model. Both models have had 10 epochs to learn how to remove the noise from the images. As we can see in the graphs of the Figure 4.6, the trend of the loss is the same. In other words, there is still room for improvement. The results are slightly better but the difference is not remarkable. If we compare the maps of experiment 1 (see Figure 4.5) and experiment 2 (see Figure 4.7), the difference is imperceptible.

EXPERIMENT 3

Continuing with Experiment 3, As seen in experiments 1 and 2, the model has the capacity to keep learning and make better predictions. In this experiment it is allowed to train up to 50 epochs, which results in much better results as can be seen in the metrics of the 4.1 Chart. Regarding the loss, we can see in the Graph 4.8 how around epoch 30 the decrease in loss slows down and at epoch 40 there is hardly any improvement. We can then intuit what will happen with the following experiments. Even so, we can see in the metrics of the Figure 4.9 that the results improves visibly compared to the 10 epochs of the previous experiments.

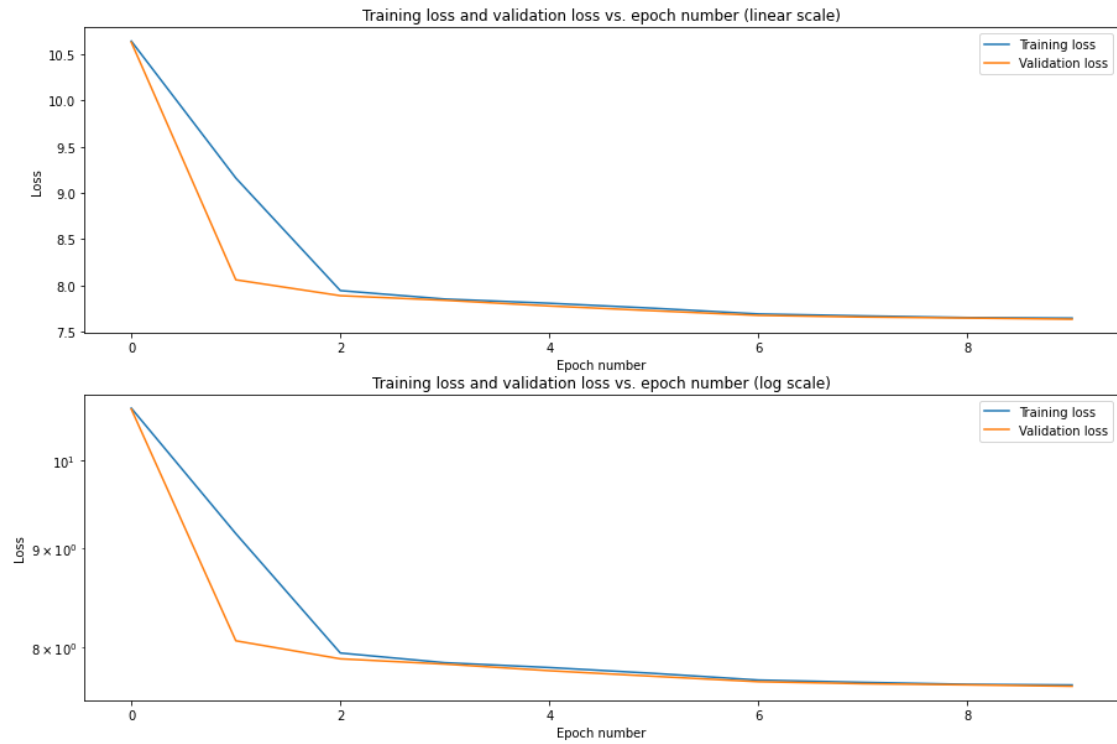


Figure 4.6: Experiment 2 loss Chart.

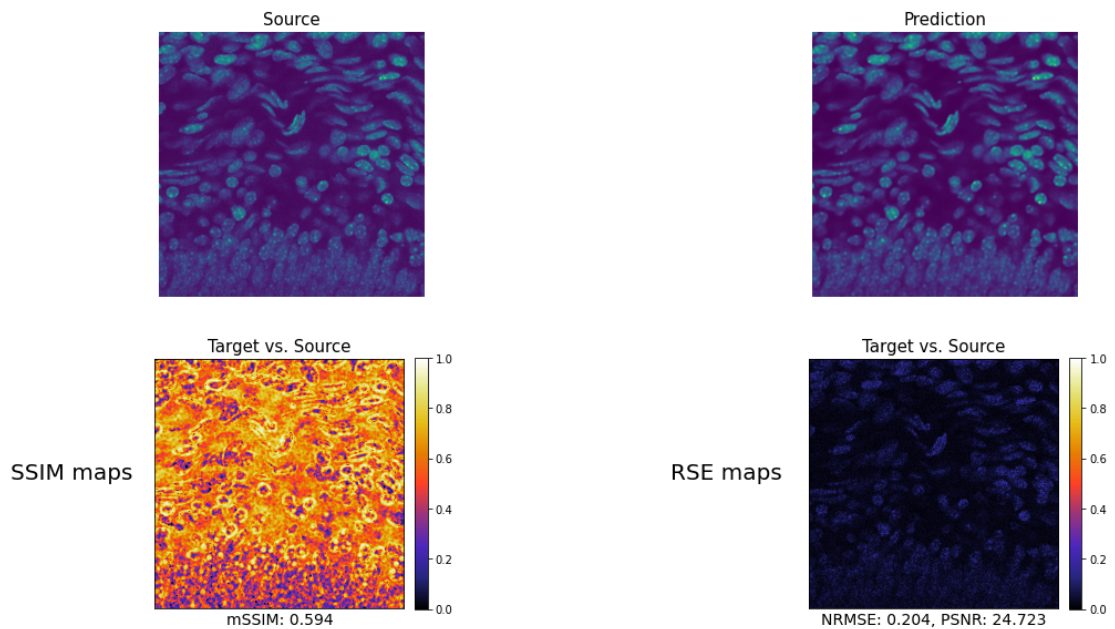


Figure 4.7: Experiment 2 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

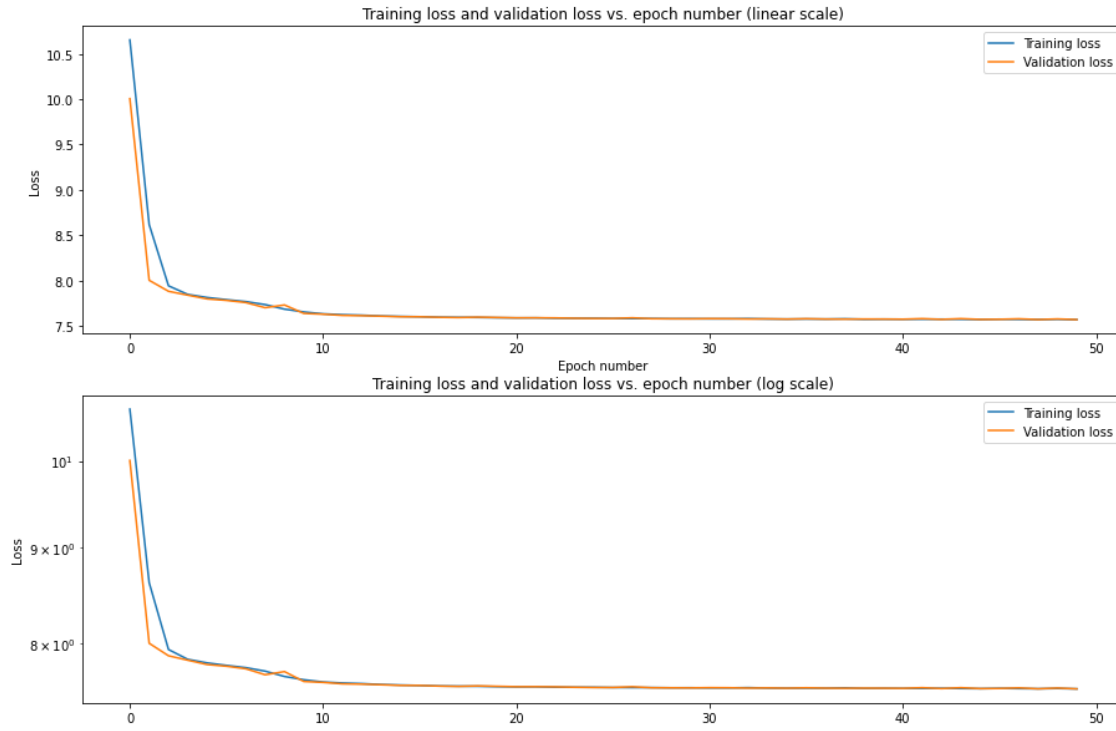


Figure 4.8: Experiment 3 loss Chart.

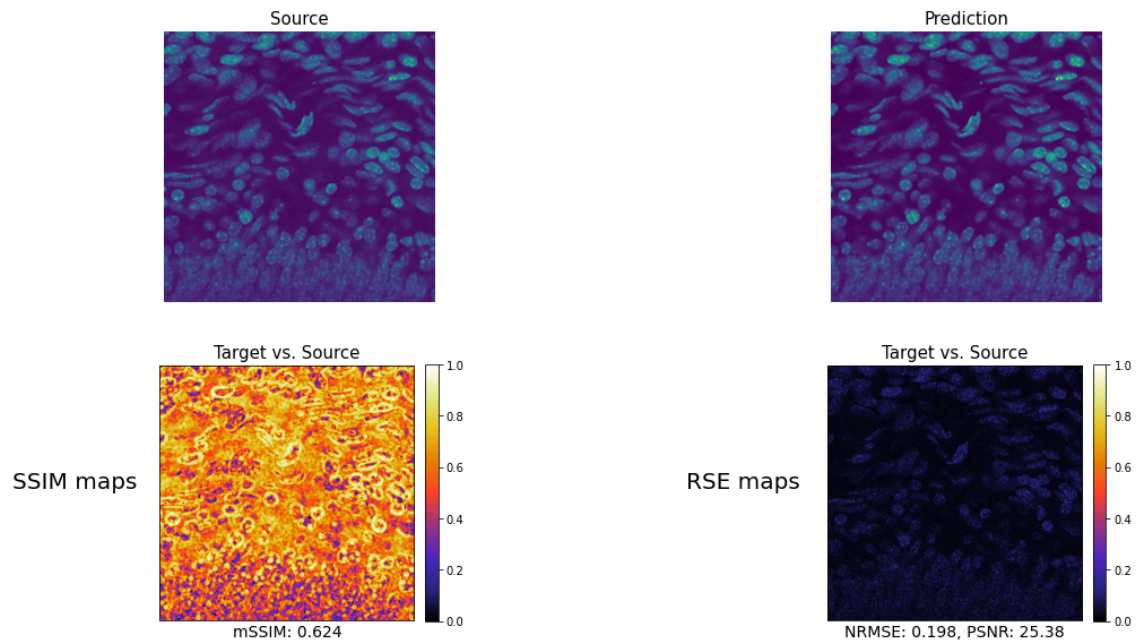


Figure 4.9: Experiment 3 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

EXPERIMENT 4

In experiment 4, the trend of experiments 1 and 2 is repeated. That is, the results provided by the model are identical using the noise model trained with calibration images or if is bootstrapped from noisy images instead. Looking at the Figure 4.10 we see that the same thing happens with the loss, it stagnates and the improvement is very small.

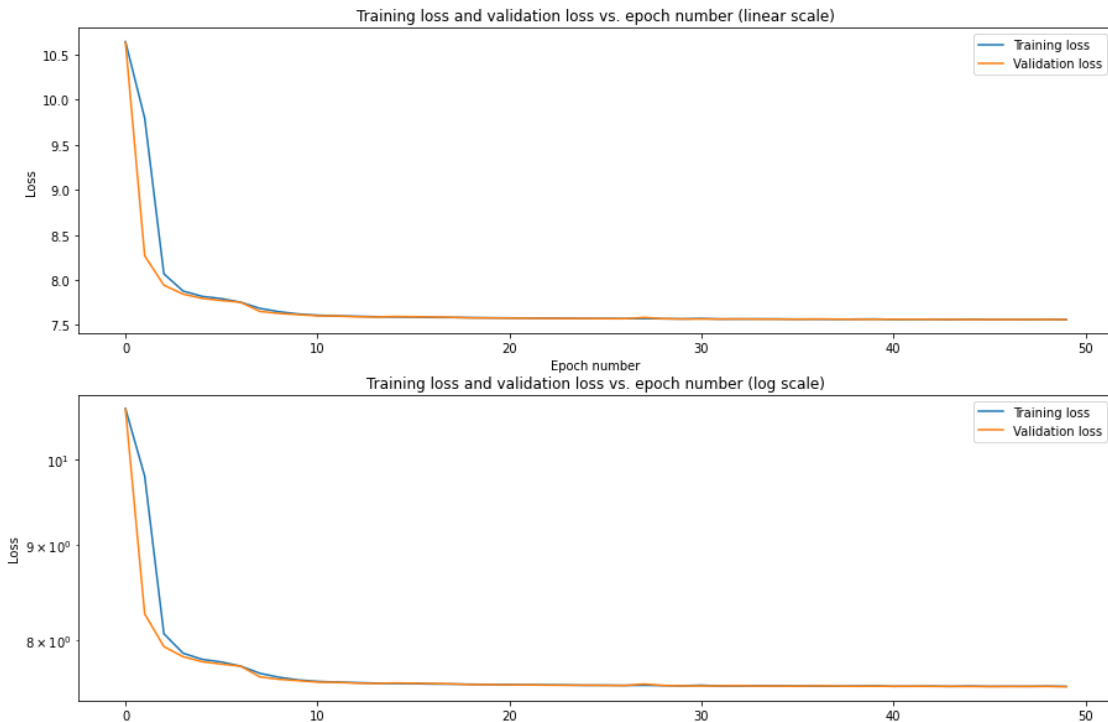


Figure 4.10: Experiment 4 loss Chart.

EXPERIMENT 5

For Experiment 5, the idea was to check if training until the model no longer improved gave good enough results to justify the training time, about 6 times more than training with 50 epochs. As we can see in the Table 4.1, this experiment is the one that gives the best results. Even so, it should be noted that the improvement is very small compared to the results obtained in experiments 3 and 4. As expected from the previous experiments, the loss decreases rapidly at the beginning but then stabilizes and the improvement is minimal. This can be seen in the Graph 4.12 and in the Table 4.1, as in spite of the considerable increase on time the metrics are almost the same.

EXPERIMENT 6

In this last experiment number 6, the method of learning the noise model is by using

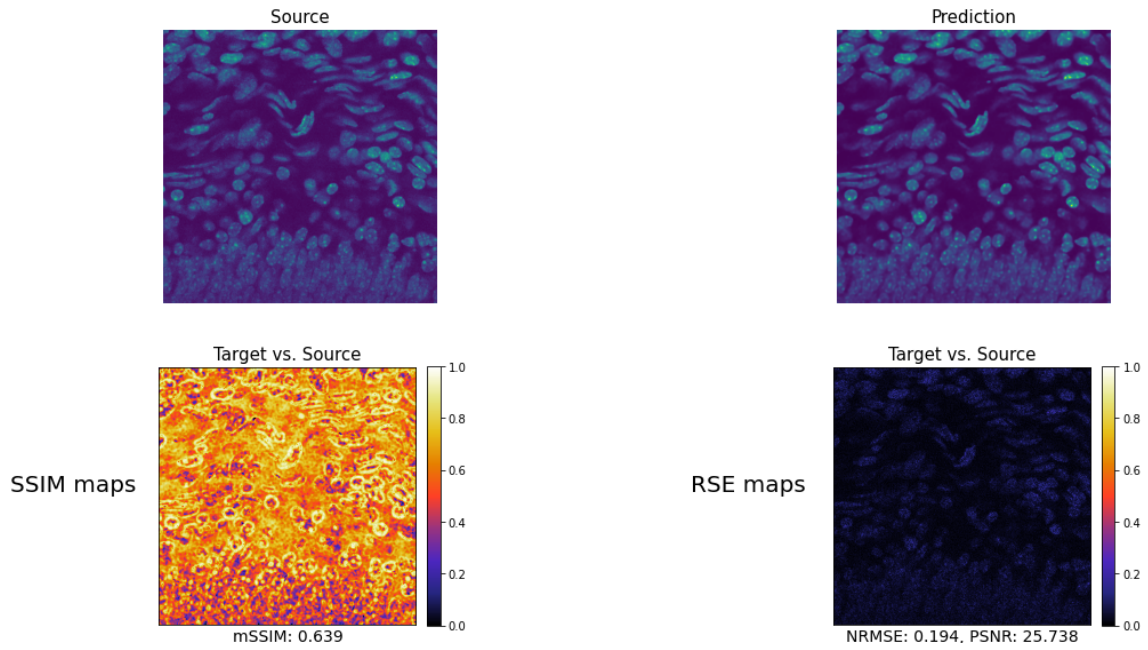


Figure 4.11: Experiment 4 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

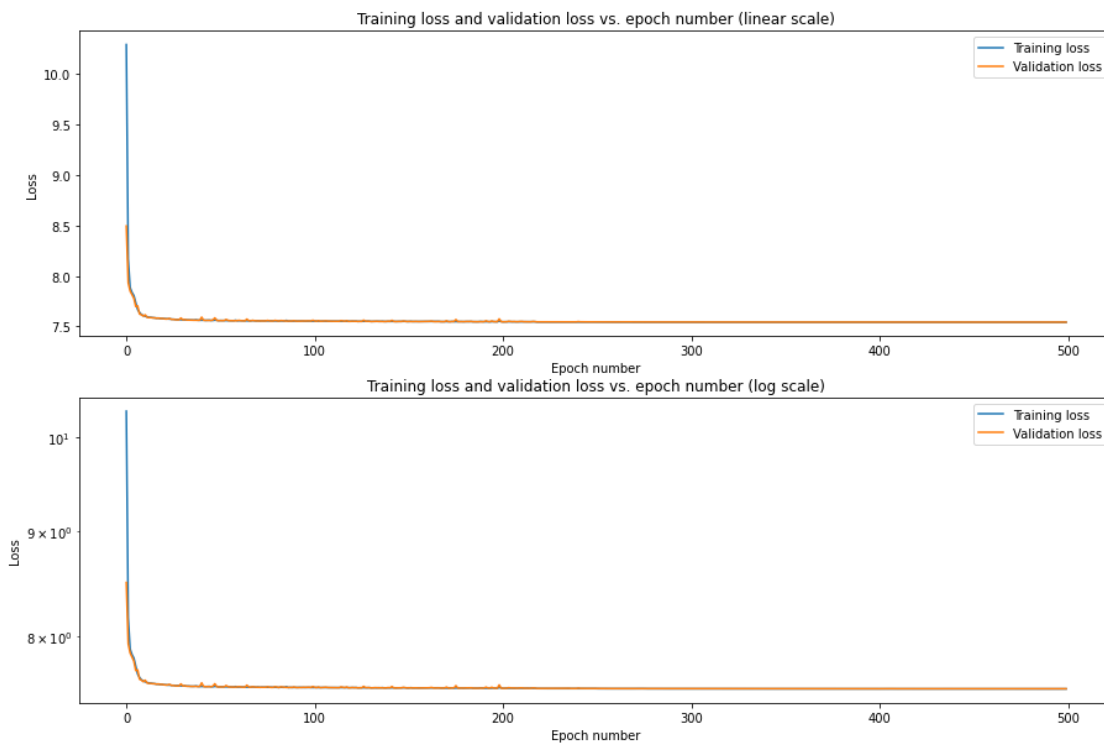


Figure 4.12: Experiment 5 loss Chart.

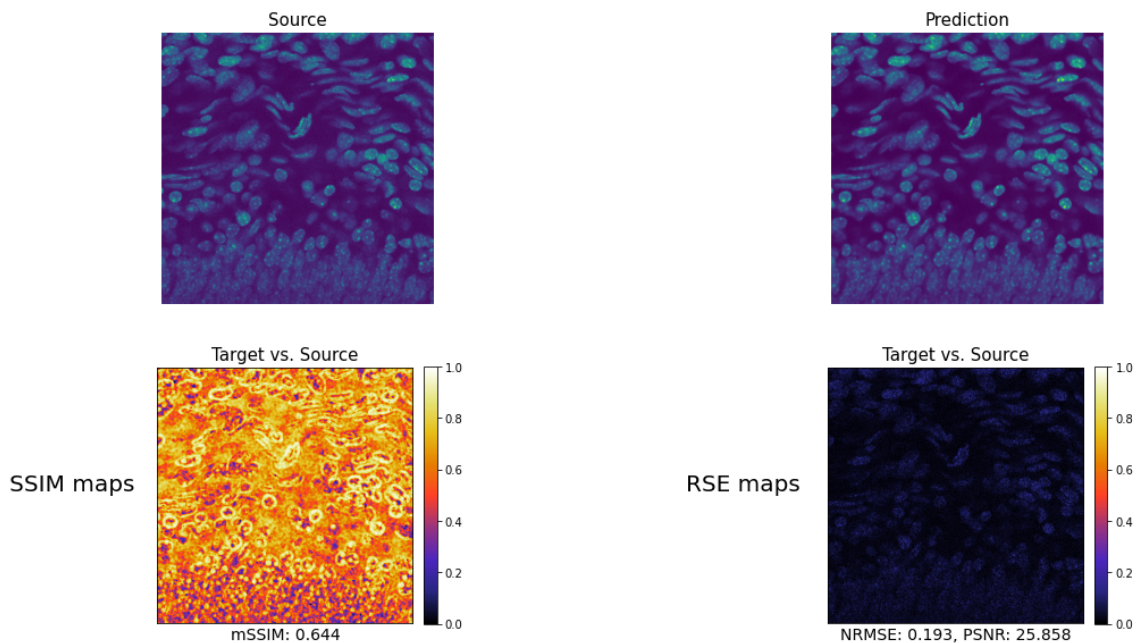


Figure 4.13: Experiment 5 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

calibration images as in experiments 2 and 4. Unlike the first two experiments performed with this technique, it gives results slightly inferior to bootstrapping. However, it is worth noting that the training process instead of finishing with 500 iterations, it finished earlier because of patience at 244 iterations. The result is almost the same as using the other method (see Table 4.1). We can say that even if we train this model more, it will not give better results than it does now.

To summarise the experiments with the first database, one could say that the model gives good results. Even so, there is a moment, around 40 epochs, where the model hardly improves. So it does not make much sense to train it much longer, as the computational time needed is not justified by the results to be obtained. Next, a figure with the worst performing instance (experiment 1) next to the best performing instance (experiment 5) (see Figure 4.19) are shown.

It can be seen in the Figure 4.19, that the instance of experiment 1 blurs the images a lot and information is lost especially at the edges and in areas of high intensity. On the contrary, the prediction of the experiment 5 instance preserves the image information much better. By not blurring the edges and areas where the contrast of values is high, the difference is better perceived. In the case of experiment 1, the inside of the cell is perceived

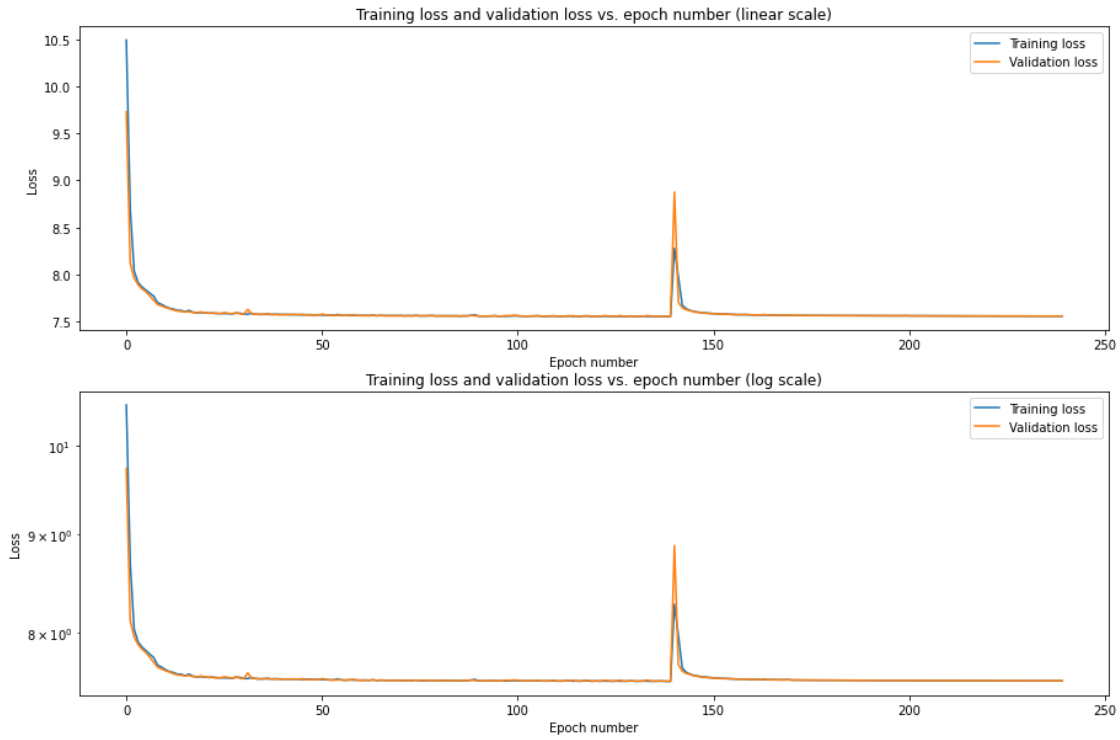


Figure 4.14: Experiment 6 loss Chart.

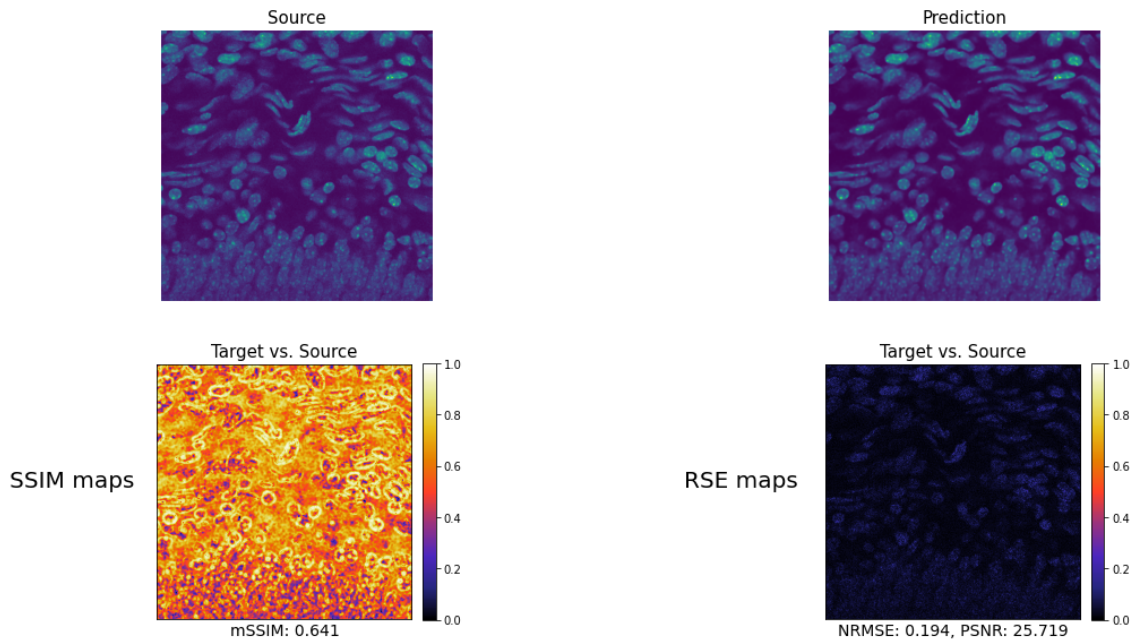


Figure 4.15: Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

as very uniform when in fact it is not. The other instance is responsible for maintaining these differences, and the shapes inside the cell, which is where the information is most complex, are better distinguished.

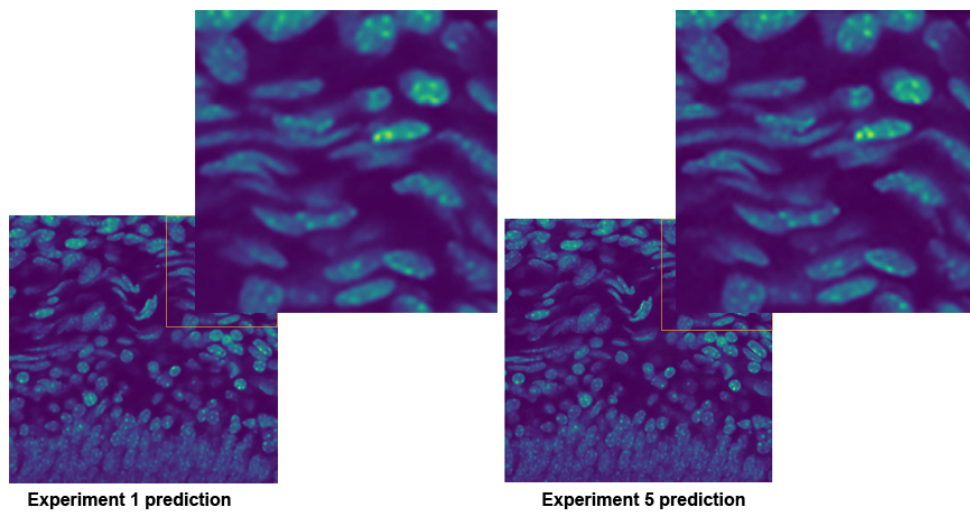


Figure 4.16: Comparison between the worse prediction and the best prediction. Left, worse, prediction from experiment 1 . Right, best, prediction from experiment 5.

4.5.2 Convallaria results

The second round of experiments were done with the convallaria dataset, a dataset composed of Convallaria majalis cell images. The Table 4.2 shows the results obtained, results that we will analyze in more detail later.

No.	Ns method	#Eps	SSIM \uparrow	NRMSE \downarrow	PSNR \uparrow
1	Bootstrap	10	0.717	0.163	27.191
2	Calibration	10	0.754	0.165	27.845
3	Bootstrap	50	0.717	0.163	27.191
4	Calibration	50	0.767	0.162	28.158
5	Bootstrap	200	0.767	0.162	28.158
6	Calibration	200	0.767	0.162	28.158

Table 4.2: Experiments of Divnoising network with Convallaria dataset. The column No is used to denote the number of experiment. The column Ns method represents the noise model generator method used in the experiment. The column Eps indicates the number of epochs the model has trained in each experiment. The next 3 columns indicate the evaluation metrics, SSIM (see Section 4.3.1), NRMSE (see Section 4.3.2) and PSNR (see Section 4.3.2). The bold line highlights the experiment with the best results.

For this second experiment the maximum epochs have been decreased from 500 to 200 because the improvement is almost null but the computation time increased a lot and caused problems with the services offered by Google Colab. However, as expected, this change had no impact on the results obtained.

As can be seen in the table, the behaviour is identical with both datasets. For this reason, in this section we will only present a graph with the loss of the best result (experiment 6), the results offered by this experiment and, as in the previous section, a comparison between the best and the worst prediction. All the results can be found in Appendix B.

It is worth noting that the results obtained with this dataset are better as observed in all metrics. This may be due to the complexity of the images and the noise in them. The images of the Mouse Nuclei dataset have more complexity (more edges and shapes) than this second dataset. For this reason the results obtained are somewhat worse.

Even so, as we have mentioned before, the behaviour of the model is the same in both cases. Around epoch 40, the improvement is almost nil, and all the training from then on hardly gives any results. You can see this in that with 50 epochs or 200 epochs the result is exactly the same. The model cannot learn any more.

Training the noise model with calibration images gives slightly better results, but it is shown that if these are not available, training the noise model with bootstrap gives almost as good results. Therefore, it is not decisive to have calibration images to be able to use this tool, and they are not always available. This makes this tool much more versatile and easily accessible for everybody.

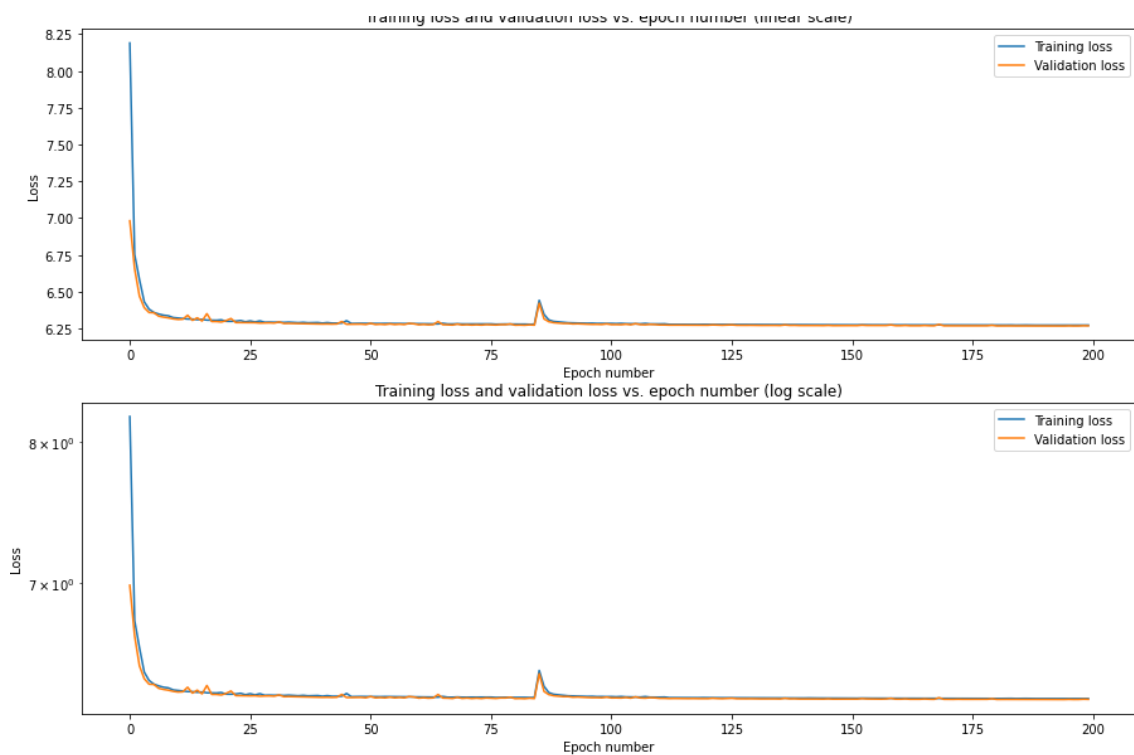


Figure 4.17: Experiment 6 loss Chart.

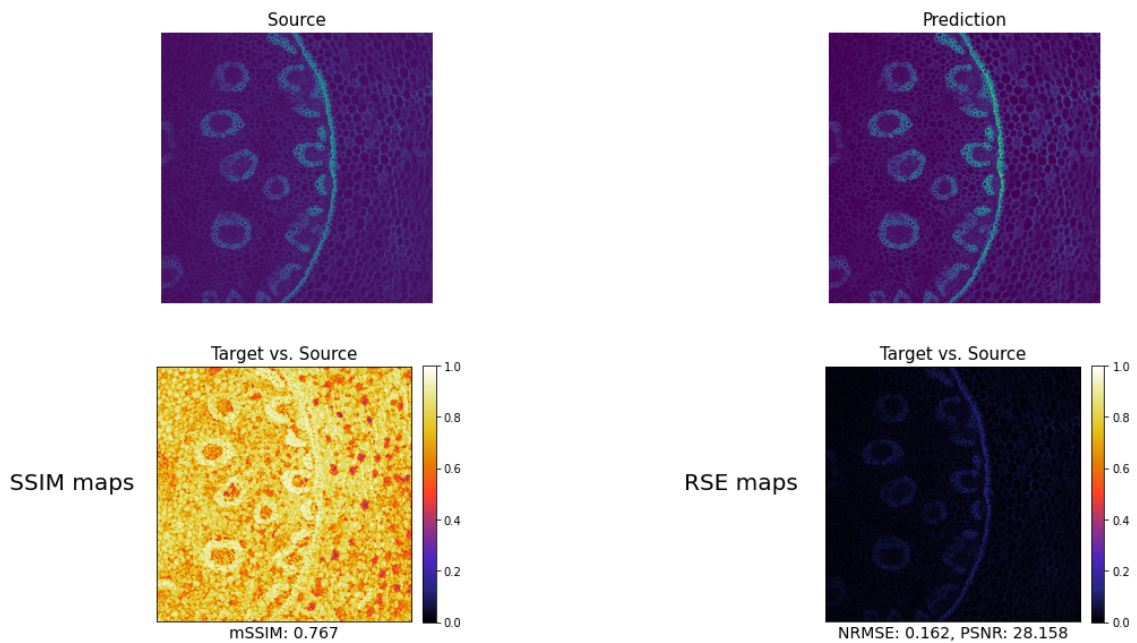


Figure 4.18: Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

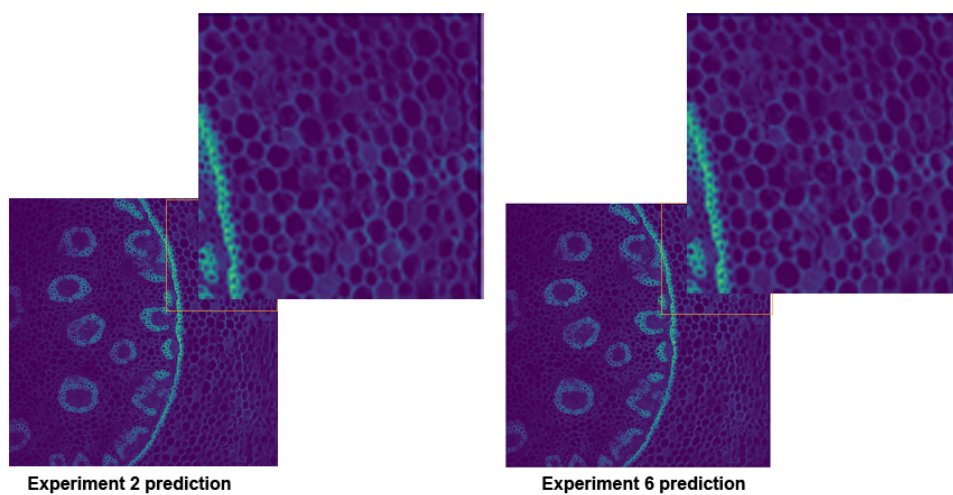


Figure 4.19: Comparison between the worse prediction and the best prediction. Left, worse, prediction from experiment 2. Right, best, prediction from experiment 6.

Conclusions

As mentioned earlier in Chapter 1, one of the goals of this project was to solve the job of noise removal. This goal has been achieved in the Chapters 2 and 3 by investigating and different denoising methods and their performance.

The second objective was to implement one of the most recent networks to tackle the denoising task. The chosen network has been *Divnoising*. A network that improves the traditional VAE for this type of work by adding a noise model. Moreover, this implementation has been developed following the ZeroCostDL4Mic team guidelines, whose goal is to make deep learning methods available to people who do not have expertise in the field of deep learning or programming. The resulting notebook is an easy-to-use notebook available to everyone.

Furthermore, the *Divnoising* model has been tested as explained in Section 4 . First it is tested with the MouseNuclei dataset. The best result, as can be seen in the Table 4.1, is given by experiment 5. It was expected that with calibration images the result would be better than with the other noise model learning method. Even so, the difference is almost null, both methods give similar results.

The model was then tested with the second dataset, Convallaria, and the best results were obtained in experiment 4. In this case, as expected, the best results are by giving the model the noise model learned from calibration images. Even so, as in the previous case, the difference is almost nil. Moreover, it shows that it is not necessary to train it for many periods because the experiment 4, 5 and 6 give the same result.

Summarising the experiments, two issues become clear. First, after 50 epochs, the improvement is very small, but the time it takes to train the model increases a lot. Second, the method chosen to learn the noise model is not so relevant when training the model. The results obtained by both methods are very similar, although using calibration images the results are slightly better.

All in all, it is important to point out that even though the predicted images are not as good as the original signal, the results can be of great use for the biologists, since the improvement from a noisy image before and after processing is significant. Additionally, the availability of the network to everyone can result in a thorough testing, which can lead to the investigation of new methods that can improve the current results.

Appendixes

CHAPTER A

Gantt Diagram

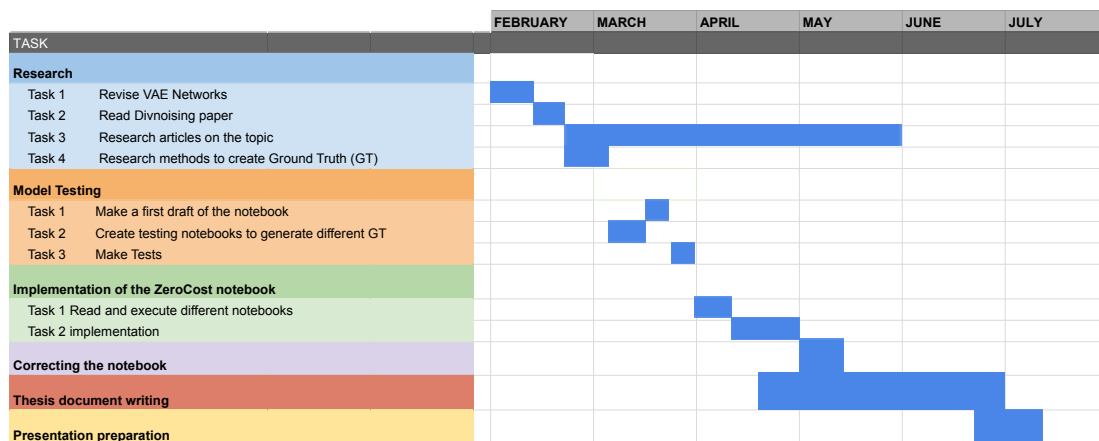


Figure A.1: Gantt diagram displaying the task with their respective duration. Research: Task 1, revise and study in depth the vae models in order to be able to implement Denoising in a notebook. Task 2, after getting to know how a vanilla VAE works, read about the new method to be implemented and see why the differences it has make it superior for this job. Task 3, to study and look at different denoising technologies in order to complete the thesis and to understand the origin of this problem and how it has been tackled during this time. To better understand the motivation behind it. Task 3, research methods to be able to create GT and choose an option to implement on the notebook.

CHAPTER B

Experiment 2 results

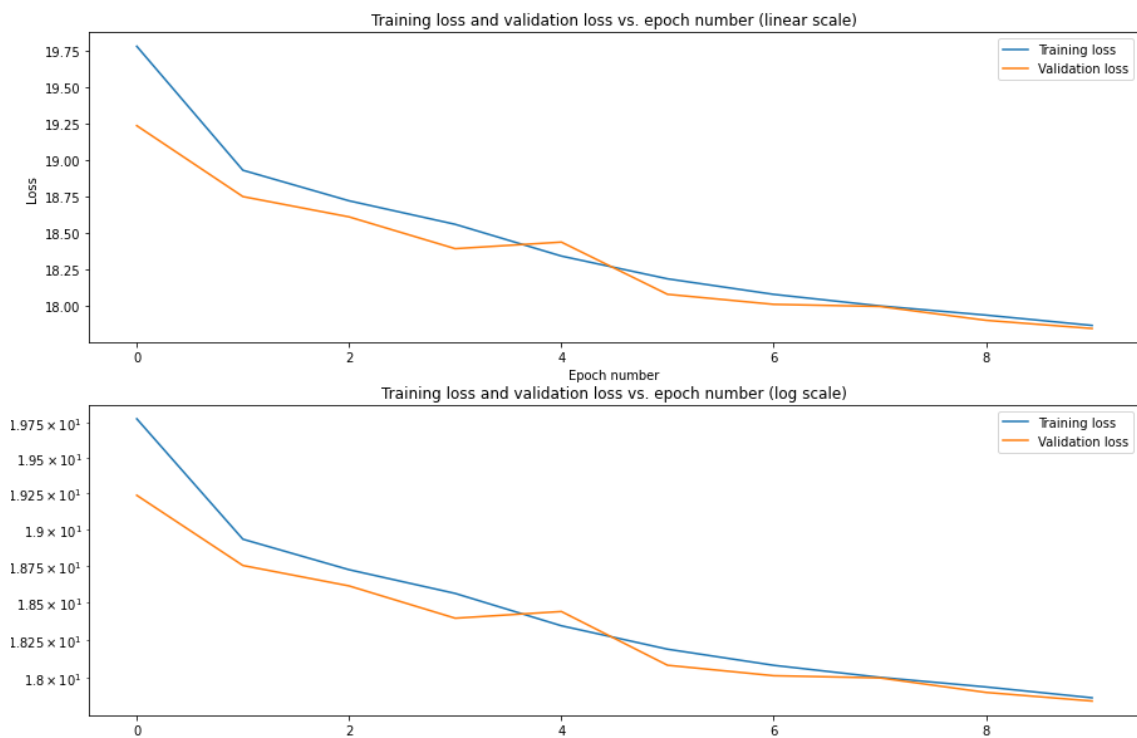


Figure B.1: Experiment 1 loss Chart.

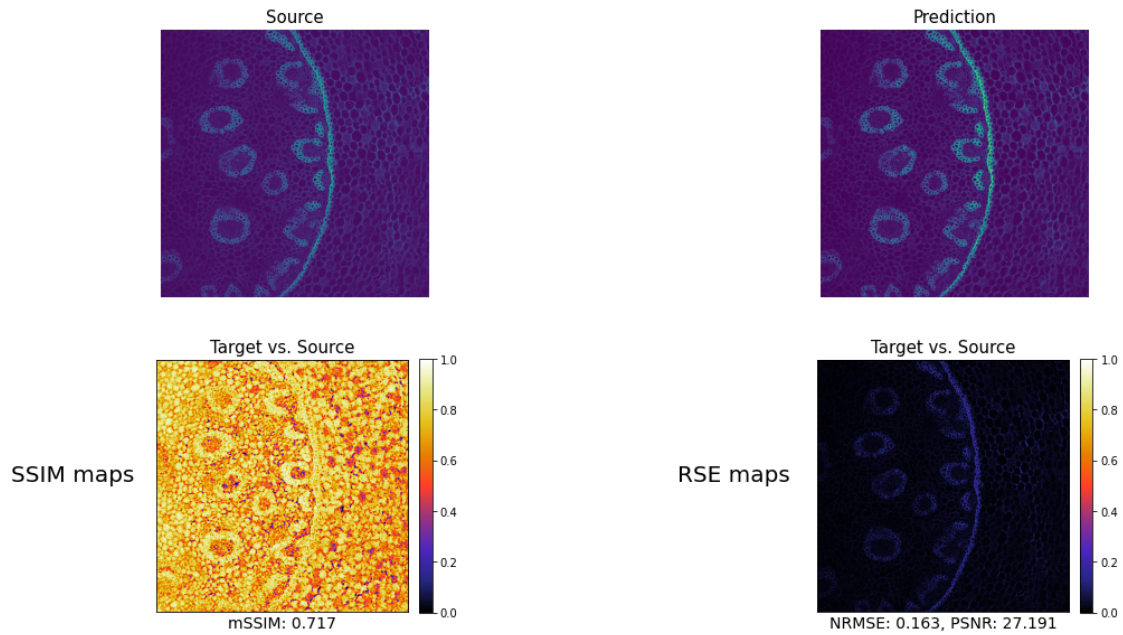


Figure B.2: Experiment 1 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

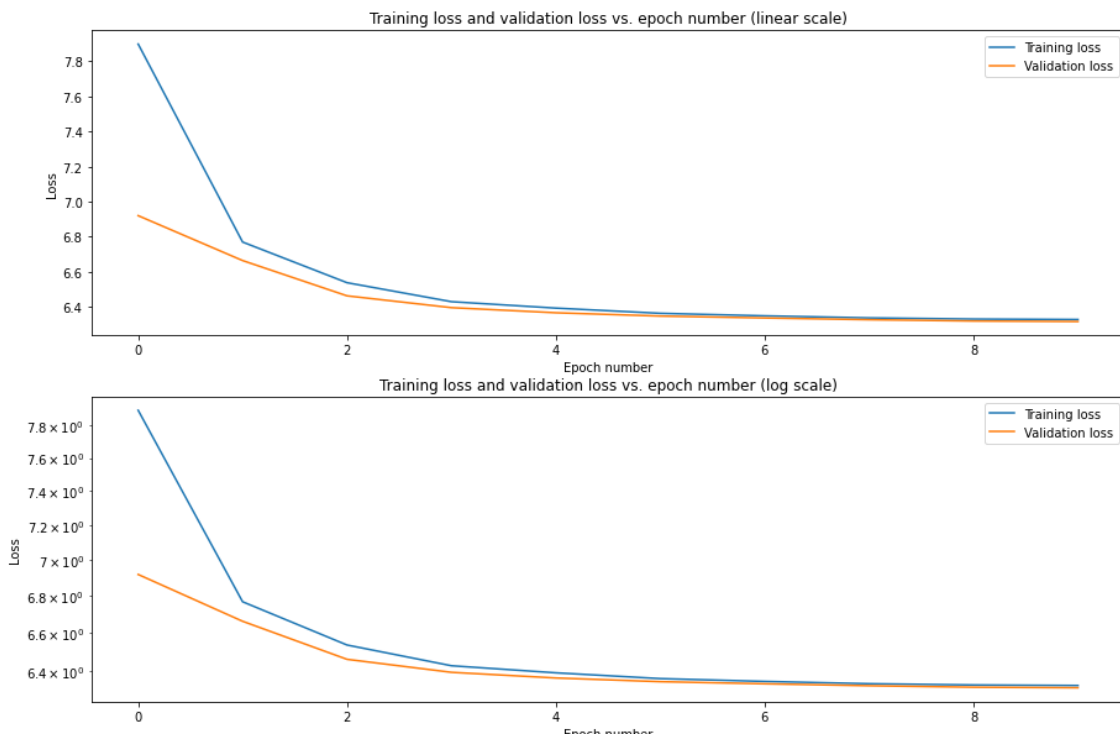


Figure B.3: Experiment 2 loss Chart.

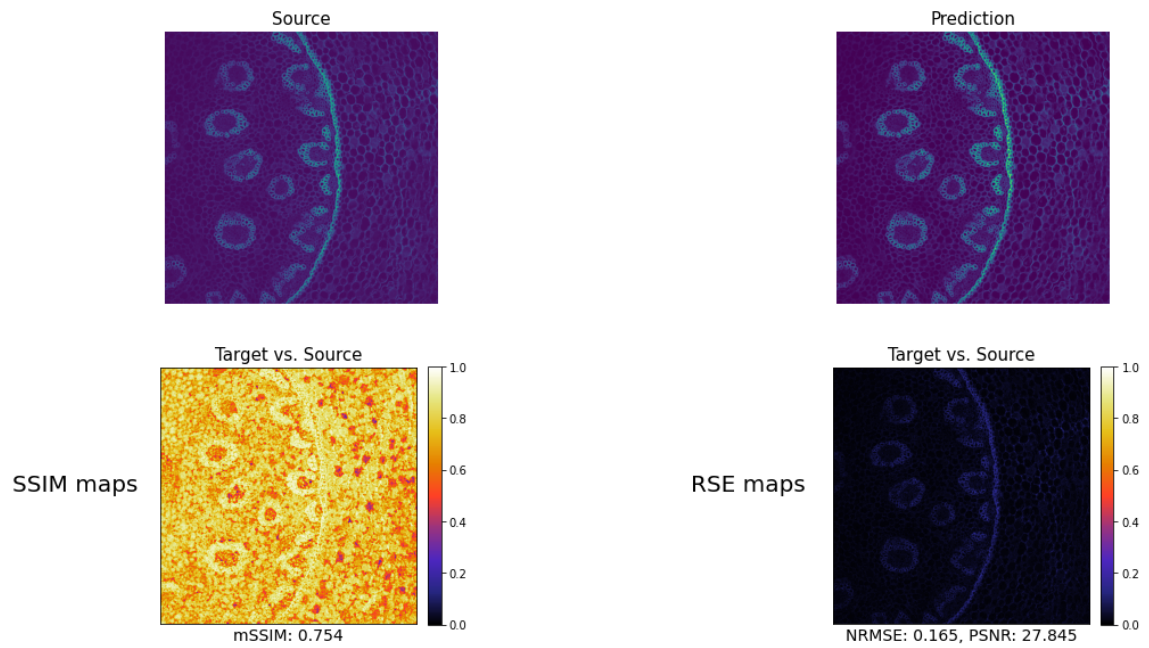


Figure B.4: Experiment 2 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

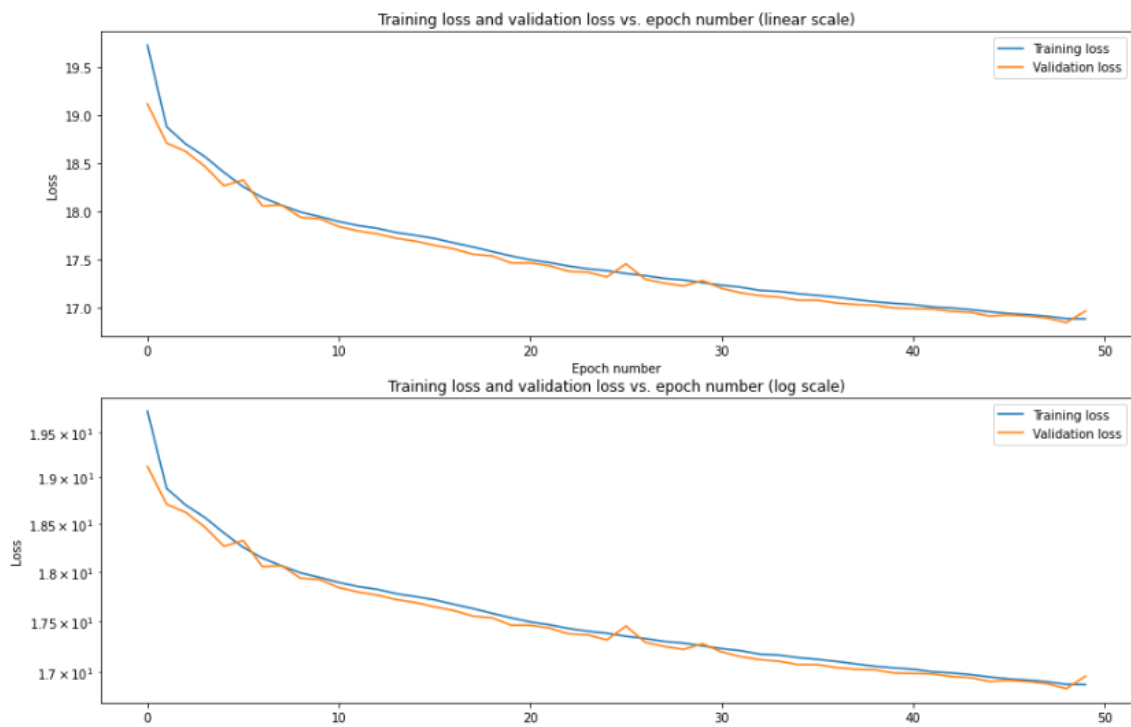


Figure B.5: Experiment 3 loss Chart.

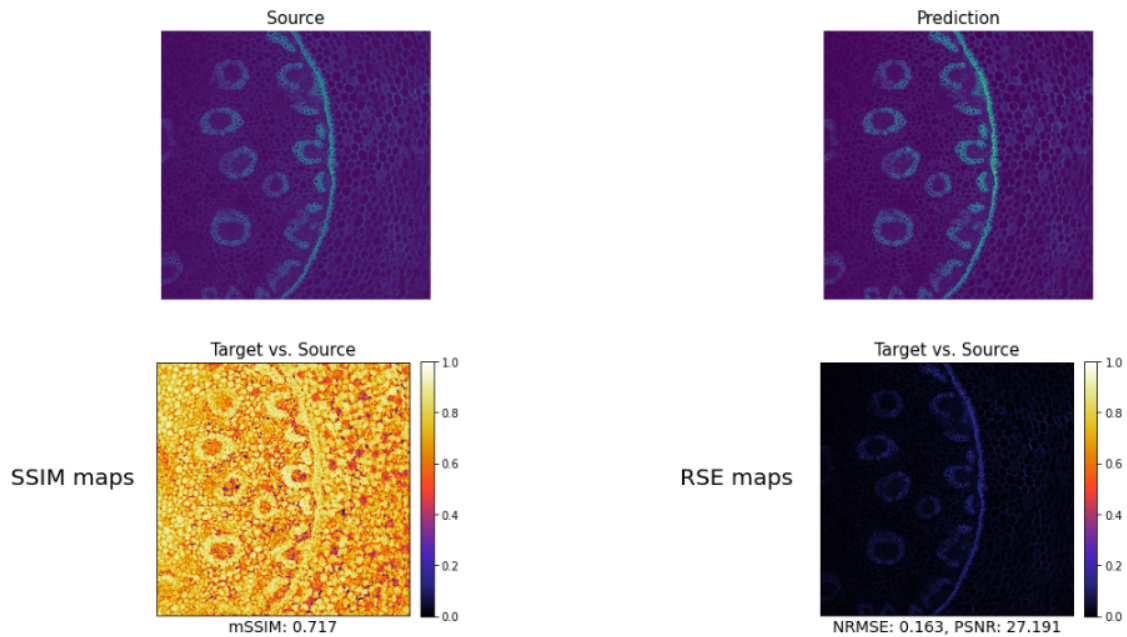


Figure B.6: Experiment 3 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

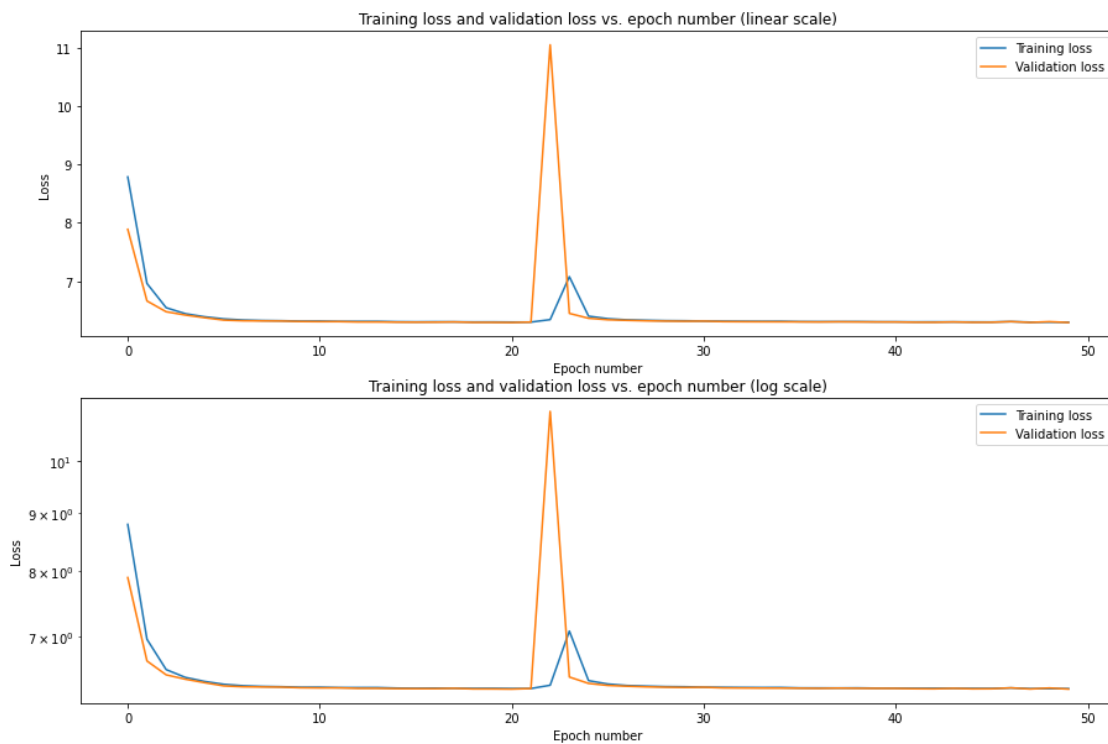


Figure B.7: Experiment 4 loss Chart.

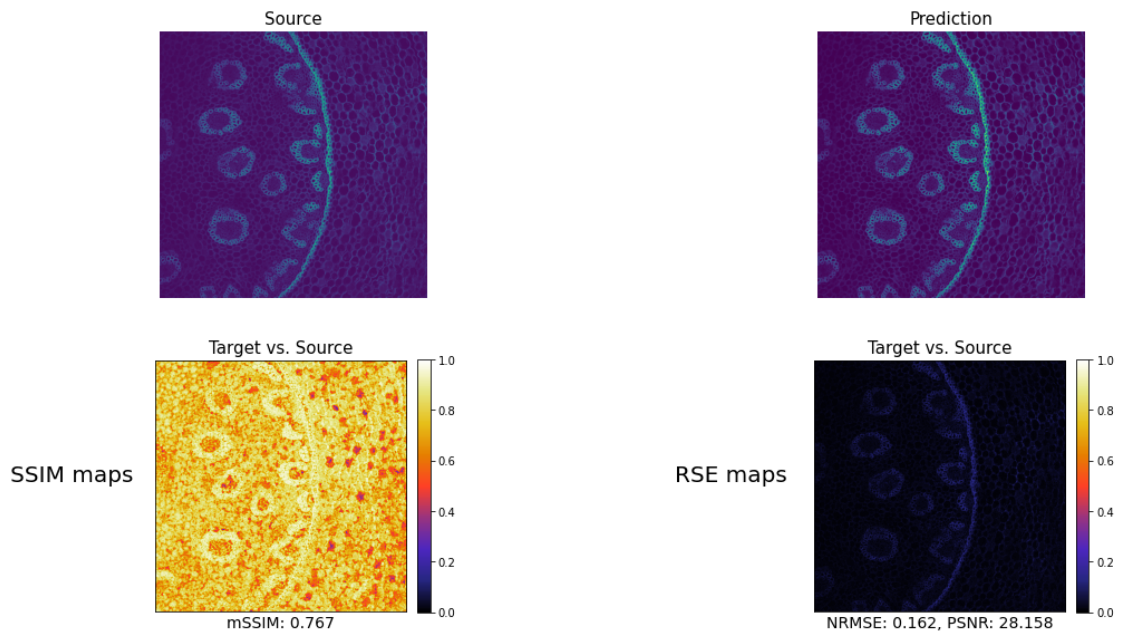


Figure B.8: Experiment 4 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

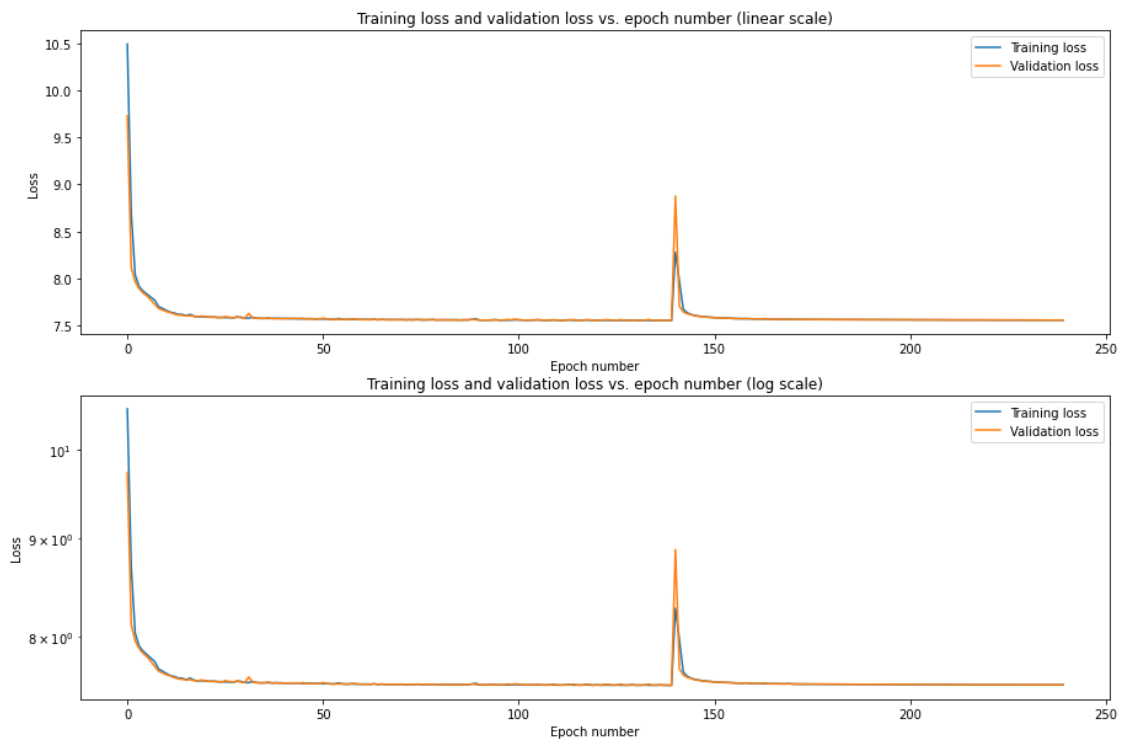


Figure B.9: Experiment 5 loss Chart.

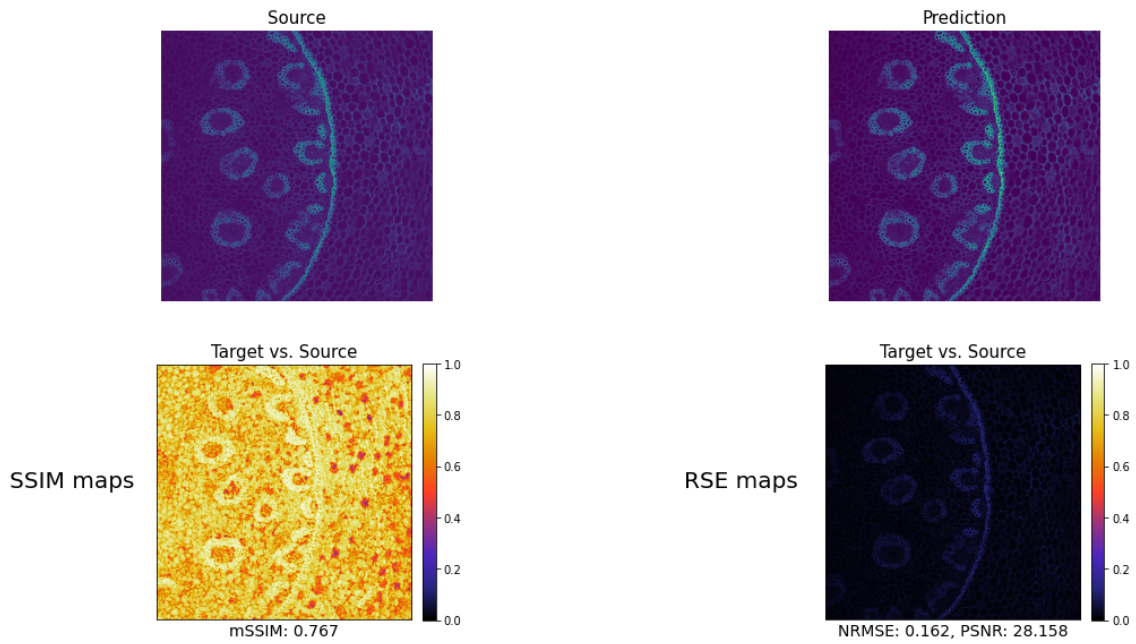


Figure B.10: Experiment 5 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

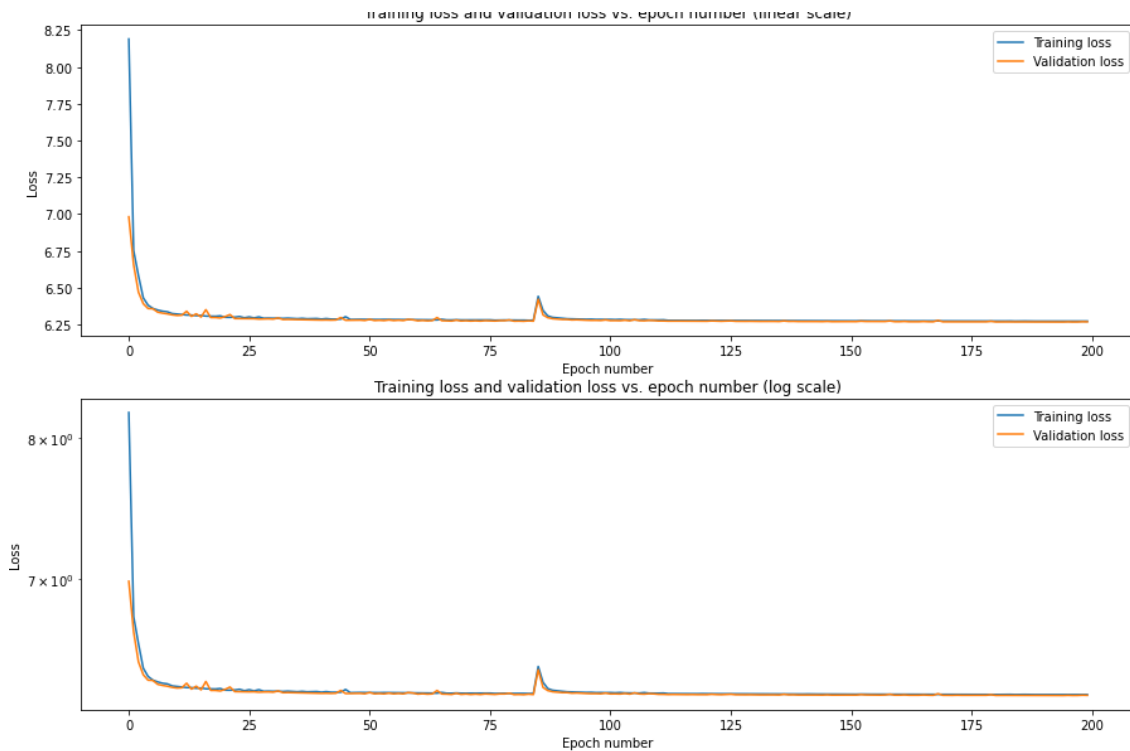


Figure B.11: Experiment 6 loss Chart.

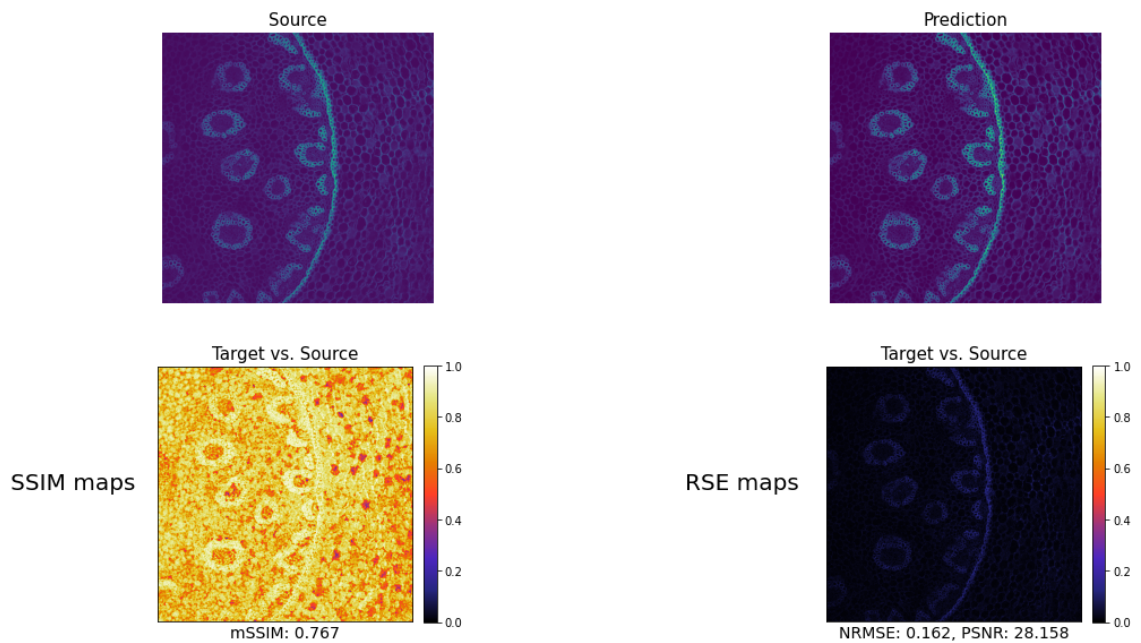


Figure B.12: Experiment 6 metrics. Top left, the original image. Top Right, The prediction given by the model. Bottom left, SSIM map with SSIM value. Bottom Right, RSE map with NRMSE and PSNR values

Bibliography

- [1] (2020). Peak signal-to-noise ratio as an image quality metric.
- [2] Boyat, A. K. and Joshi, B. K. (2015). A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489*.
- [3] Charmouti, B., Junoh, A. K., Mashor, M. Y., Ghazali, N., Wahab, M. A., Muhammad, W. Z. A. W., Yahya, Z., and Beroual, A. (2019). An overview of the fundamental approaches that yield several image denoising techniques. *Telkomnika*, 17(6).
- [4] Chen, Q. and Wu, D. (2010). Image denoising by bounded block matching and 3d filtering. *Signal Processing*, 90(9):2778–2783.
- [5] Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799.
- [6] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. (2007). Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095.
- [7] Goyal, B., Dogra, A., Agrawal, S., Sohi, B., and Sharma, A. (2020). Image denoising review: From classical to state-of-the-art approaches. *Information Fusion*, 55:220–244.
- [8] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., and Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48.
- [9] Huang, H. (1981). Biomedical image processing. *Critical reviews in bioengineering*, 5(3):185–271.
- [10] Ilesanmi, A. E. and Ilesanmi, T. O. (2021). Methods for image denoising using convolutional neural network: a review. *Complex & Intelligent Systems*, 7(5):2179–2198.

-
- [11] Jain, A. K., Murty, M.Ñ., and Flynn, P. J. (1999). Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323.
- [12] Jain, V. and Seung, S. (2008). Natural image denoising with convolutional networks. *Advances in neural information processing systems*, 21.
- [13] Katkovnik, V. and Egiazarian, K. (2017). Sparse phase imaging based on complex domain nonlocal bm3d techniques. *Digital Signal Processing*, 63:72–85.
- [14] Kaur, J., Kaur, M., Kaur, P., and Kaur, M. (2012). Comparative analysis of image denoising techniques. *International journal of Emerging Technology and Advanced engineering*, 2(6):296–298.
- [15] King, M. A., Doherty, P. W., Schwinger, R. B., and Penney, B. C. (1983). A wiener filter for nuclear medicine images. *Medical physics*, 10(6):876–880.
- [16] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [17] Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.
- [18] Krull, A., Buchholz, T.-O., and Jug, F. (2019). Noise2void - learning denoising from single noisy images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [19] Krull, A., Vičar, T., Prakash, M., Lalit, M., and Jug, F. (2020). Probabilistic noise2void: Unsupervised content-aware denoising. *Frontiers in Computer Science*, 2:5.
- [20] Kujawa, S. and Krahl, D. (1992). Performance of a low-noise ccd camera adapted to a transmission electron microscope. *Ultramicroscopy*, 46(1-4):395–403.
- [21] Lebrun, M. (2012). An analysis and implementation of the bm3d image denoising method. *Image Processing On Line*, 2012:175–213.
- [22] Lee, J.-S. (1983). Digital image smoothing and the sigma filter. *Computer vision, graphics, and image processing*, 24(2):255–269.
- [23] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., and Aila, T. (2018). Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*.

- [24] Li, X., Hu, Y., Gao, X., Tao, D., and Ning, B. (2010). A multi-frame image super-resolution method. *Signal Processing*, 90(2):405–414.
- [25] Mentaschi, L., Besio, G., Cassola, F., and Mazzino, A. (2013). Why nrmse is not completely reliable for forecast/hindcast model test performances. In *Geophysical Research Abstracts*, volume 15.
- [26] Pap, E. and Štrboja, M. (2010). Generalization of the Jensen inequality for pseudo-integral. *Information Sciences*, 180(4):543–548.
- [27] Peng, J., Shi, C., Laugeman, E., Hu, W., Zhang, Z., Mutic, S., and Cai, B. (2020). Implementation of the structural similarity (ssim) index as a quantitative evaluation tool for dose distribution error detection. *Medical physics*, 47(4):1907–1919.
- [28] Pinaya, W. H. L., Vieira, S., Garcia-Dias, R., and Mechelli, A. (2020). Convolutional neural networks. In *Machine learning*, pages 173–191. Elsevier.
- [29] Prakash, M., Krull, A., and Jug, F. (2020). Fully unsupervised diversity denoising with convolutional variational autoencoders. *arXiv preprint arXiv:2006.06072*.
- [30] Prakash, M., Krull, A., and Jug, F. (2021). Fully unsupervised diversity denoising with convolutional variational autoencoders. In *International Conference on Learning Representations*.
- [31] Rabbani, H., Nezafat, R., and Gazor, S. (2009). Wavelet-domain medical image denoising using bivariate Laplacian mixture model. *IEEE transactions on biomedical engineering*, 56(12):2826–2837.
- [32] Rocca, J. (2019). Understanding variational autoencoders (VAEs) building, step by step, the reasoning that leads to VAEs.
- [33] Roy, V. (2013). Spatial and transform domain filtering method for image de-noising: A review. *International Journal of Modern Education & Computer Science*, 5(7).
- [34] Sairam, R. M., Sharma, S., and Gupta, K. (2013). Study of denoising method of images—a review. *Journal of Engineering Science and Technology Review*, 8(5):41–48.
- [35] Sharma, A. and Singh, J. (2013). Image denoising using spatial domain filters: A quantitative study. In *2013 6th International Congress on Image and Signal Processing (CISP)*, volume 1, pages 293–298. IEEE.

-
- [36] Tian, C., Fei, L., Zheng, W., Xu, Y., Zuo, W., and Lin, C.-W. (2020). Deep learning on image denoising: An overview. *Neural Networks*, 131:251–275.
- [37] von Chamier, L., Laine, R. F., Jukkala, J., Spahn, C., Krentzel, D., Nehme, E., Lerche, M., Hernández-Pérez, S., Mattila, P. K., Karinou, E., et al. (2021). Democratizing deep learning for microscopy with zerocostdl4mic. *Nature communications*, 12(1):1–18.
- [38] Wildberger, K., Lang, P., Zeller, R., and Dederichs, P. (1995). Fermi-dirac distribution in ab initio green’s-function calculations. *Physical Review B*, 52(15):11502.
- [39] Windyga, P. S. (2001). Fast impulsive noise removal. *IEEE transactions on image processing*, 10(1):173–179.