eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

BACHELOR DEGREE IN COMPUTER ENGINEERING

COMPUTER SCIENCE

Thesis

Easy-to-use deep learning based super-resolution in microscopy images

Author
Ainhoa Serrano Guerrero

informatika fakultatea
facultad de informática

2021

Abstract

The main objective of this work is to present one of the most recent deep neural networks to solve the super-resolution task, named DFCAN, as well as the study of different methods that solve the same problem. Throughout the document, different approaches are explained and compared, emphasising the state-of-the-art methods.

This work also contains different experiments done with the DFCAN using different datasets. Finally, to complete the thesis, there is a guide for an easy-to-use Jupyter notebook, created with the aim of being available for anyone, specifically designed for people who do not have expertise in programming and deep learning.

Contents

List of Figures

Table index

CHAPTER 1

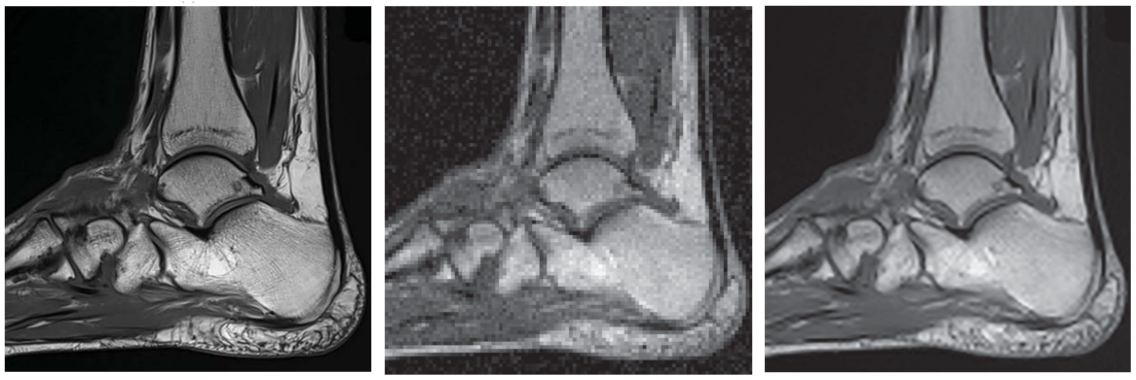
Introduction

1.1 Image super-resolution

In the last decades, imaging techniques have undergone a rapid development due to the technology progress, and as a result, the resolution has also improved. Image resolution refers to how much detail is portrayed in an image and it is usually measured in pixels.

This development has resulted in an increase of demands to generate high resolution images from existing low-resolution images. The fact is that there are some applications that need this resolution enhancement, such as security surveillance, where the image resolution is inferior in order to ensure the stability of the recording devices in the long term. Moreover, considering the high cost of some devices that obtain high resolution images, purchasing them is not an option.

Therefore, resolution enhancement is still needed, and with the aim of addressing that problem, the super-resolution (SR) task has been a research focus in the computer vision field, which consists of reconstructing a high-resolution (HR) image from one or more low-resolution (LR) images. As previously mentioned, a high resolution image has more details than a low resolution one, hence it can be quite useful in certain areas such as security surveillance, medical imaging and satellite imaging.



(a) Original HR image (b) Corrupted LR image (c) SR result

Figure 1.1: Example of a super-resolution result for a magnetic resonance imaging (MRI) image [1]

More particularly, in the medical field, images can be quite determining in diagnosis and research. Medical imaging can play a significant role to determine the presence of many diseases and analysis of experimental results, so enlarging medical images can help medical experts to elevate diagnosis accuracy in pathology research (see Figure 1.1). For example, in an X-ray, the detail needs to be rather high in order to detect possible fractures, or even tumors, therefore obtaining images with poor resolution is out of question. Due to this fact, doctors rely on expensive medical imaging systems, and it is of the utmost importance to discover inexpensive and effective ways to obtain high resolution images without resorting to sophisticated devices.

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 high-resolution samples. Compared to macroscale photographs, microscopy images demand greater accuracy, therefore the SR algorithms must be more precise for the predictions to be acceptable.

[1]Figure taken from: [4]

1.1.1 Objectives of this project

Over the years, the methods aimed to solve the task of super-resolution have evolved obtaining solid results. Especially, the study of deep learning methods are currently at its peak, given the fact that the results they obtain surpass previous methods that will be later explained in this work (see Section 2.2).

Taking this into account, the main goal of this project is the study of modern deep learning methods for the super-resolution task, particularly for microscopy images. More specifically, one of the goals is to use a deep neural network that generates high resolution images from low resolution ones. The architecture that is going to be tested in order to achieve this is the Deep Fourier Channel Attention Network (DFCAN) [5], a network inspired by the spatial domain channel attention mechanism employed in the deep residual channel attention network (RCAN) [6]. This network has been very recently proposed in an article published in Nature Methods [5].

As deep learning methods have not been thoroughly explored for optical microscopy, the architecture will be subjected to testing using microscopy images. The test dataset that we will used is composed by LR-SR image pairs, created using multi-modality structured illumination microscopy (SIM). This dataset is publicly available with the name of BioSR [2].

Secondly, it is important to mention that although the investigation and testing of super-resolution methods are important, they are limited to the computer science community since the creation and use of them require a minimum knowledge in the field. This limitation can entail a deceleration in its development since less testing means less information. For example, for the medical imaging, the testing and feedback of the experts can lead to the improvement of these methods. Besides, these achievements can loose significance if the interested people can not make use of them. Therefore, with the goal of closing this gap, the platform ZeroCostDL4Mic [7] was created, which provides access to user-friendly Jupyter Notebooks for popular pre-existing networks. These cover a range of important image analysis tasks (e.g., segmentation, denoising, restoration, label-free prediction).

For this reason, the DFCAN architecture will be implemented in a notebook for the ZeroCostDL4Mic project. The platform offers 3 types of implemented networks:

1. **Fully supported:** considered mature and considerably tested by the Zero-

[2]Dataset available in: BioSR dataset

CostDL4Mic team.

2. **Under beta-testing:** an early prototype of networks which may not be stable yet.

3. **Contributed:** networks following the ZeroCostDL4Mic guidelines and contributed by community members. The ZeroCostDL4Mic team does not maintain these networks, but they have been tested so that they offer a similar workflow experience and quality control.

Therefore, the final notebook for this project falls under the *Contributed* category and thus, it will follow their guidelines and will be submitted to them.

To sum up, the main objectives of this work will be the following:

1. Familiarisation and study of state-of-the-art techniques for the single image super-resolution (SISR) task.

2. Implementation of a modern network for super-resolution in microscopy images, more specifically, the DFCAN network.

3. Adaptation of the implemented network to the ZeroCostDL4Mic scheme.

4. Evaluation on real data.

CHAPTER 2

Background

2.1 Technical background

The most commonly used image sensors nowadays are the charge-coupled devices (CCDs) and the complementary metal oxide sensors (CMOSs) [8], two different technologies for capturing images digitally by converting light into electric charge and processing it into electronic signals.

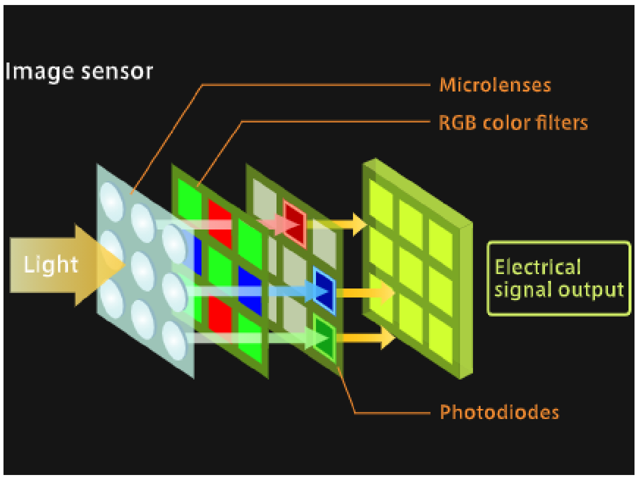


Figure 2.1: Processing of light into electrical signals [1]. Firstly, the microlenses collect the light which is then passed to the following layer, the colour filters. These filters are indispensable because the image sensor are unable to reproduce colours by themselves. Lastly, the filters pass the information to the photodiodes, which generate the electrical signal output.

The spatial resolution, the number of pixels utilised in construction of a digital image, is restrained by the CCD array and the optical lens, so one of the options to increase that resolution is to decrease the pixel size. However, there is a problem with that approach: as the pixel size decreases, the amount of light decreases too, and thus the image quality is degraded by shot noise, a property of the light field. Another solution to increase the spatial resolution is to increase the focal length, but this approach results in an enlargement of cameras which inflates its price. Despite CCDs being superior in terms of image resolution, sensor sensitivity, noise suppression and technology development, CMOSs have been more studied recently, due to the high cost of manufacturing CCD based cameras.

All in all, the limitations of the hardware technology have made it vital to study algorithms to accomplish the task of super-resolution.

2.2 Image Super-resolution methods

As previously stated, image super-resolution refers to the task of obtaining high resolution images from low resolution ones. However, this task can be categorised into different groups depending on elements like the number of images used to obtain the HR image.

According to the number of input LR images, the SR can be classified into single image super-resolution (SISR) and multi-image super-resolution (MISR), although the former one is significantly more used considering that it is more efficient. MISR methods use patch recurrence across images to obtain correlation. However, the computational cost of these methods is especially high. On the other hand, in the SISR problem, the LR image is usually a low-pass filtered and downsampled version of an HR image, therefore, this is considered an ill-posed problem due to the fact that one LR image can correspond to several possible HR images.

[1]Figure taken from: The Nanotech Museum website

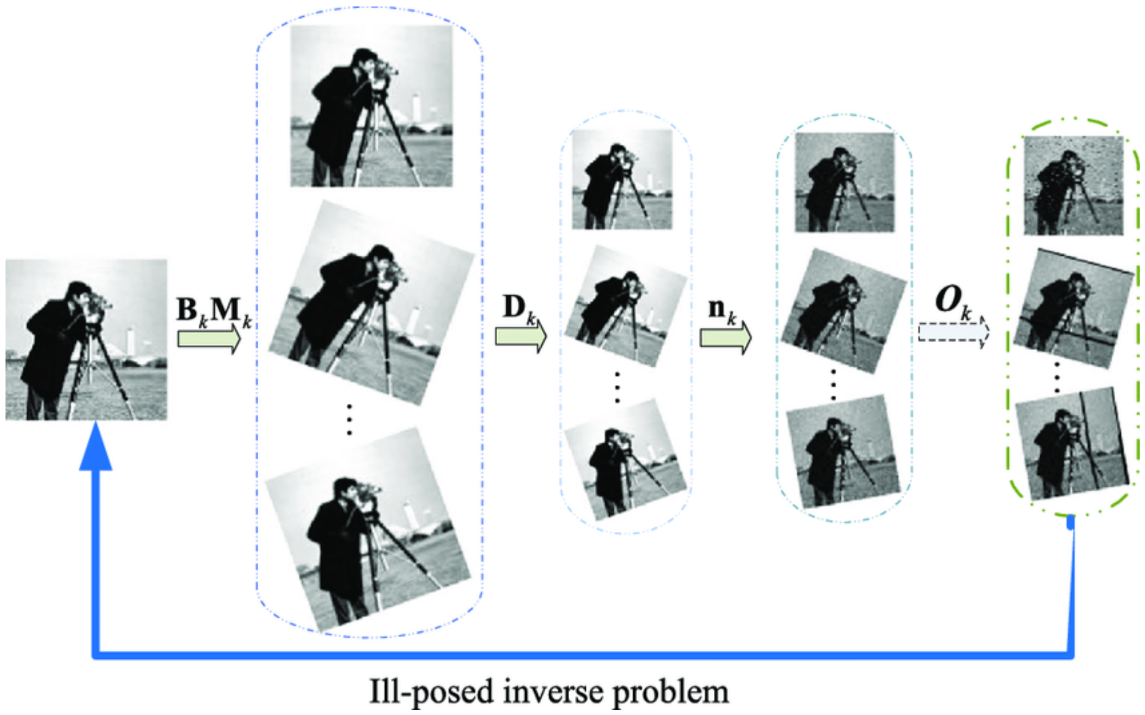


Figure 2.2: The super-resolution imaging model. B_k , M_k and D_k indicate the blur matrix, warp matrix and down-sampling matrix, respectively. n_k represents the additive noise, while O_k is the operator cropping the observable pixels from y_k. The first image represents the high resolution (HR) image, which is then modified by different methods obtaining a low resolution (LR) image. As it can be seen, variations in the methods produce different low resolution images, meaning that obtaining a HR image from a LR one is an ill-posed problem, since a HR image can correspond to different LR images and vice versa. [2]

To date, SISR algorithms are primarily divided into three categories: interpolation-based methods, reconstruction-based methods and learning-based methods. Interpolation - based SISR methods, such as Lanczos resampling [9] and bicubic interpolation [10], are very speedy and straightforward, but they lack accuracy. Reconstruction-based SR methods often use complex prior knowledge to restrict the possible solution space with the outcome of generating sharp details. However, as the scale factor increases, the performance of many reconstruction based methods diminish, and thus the methods become more time-consuming.

Learning-based SISR methods, or example based methods, are highly relevant considering their fast computation and remarkable performance. These methods usually employ machine learning algorithms to analyse relationships between the LR and its corresponding HR image from training examples [11] [12].

[2]Figure taken from: [8]

Up to now, deep learning based models have been highly considered to solve this task, as they have proved to obtain excellent results compared to the aforementioned methods [13]. In a nutshell, deep learning is a subfield of machine learning that uses multilayered neural networks with the purpose of learning diverse representations of data.

2.3 Single image super-resolution problem formulation

Given a LR image Y downsampled from the corresponding HR image X, they can be related by the following degradation model,

$$Y = D(X, \alpha) \tag{2.1}$$

where α is the parameter by which X is degraded, and D the degradation process. The goal of SISR is to solve the equation so that it obtains a good approximation of the HR image by reversing the degradation process shown in Equation (2.1), which can be represented as follows,

$$\hat{X} = R(Y, \beta) \tag{2.2}$$

where R is the SISR reconstruction function, β its parameter and $\hat{X}$ the estimation of the HR image.

In theory, the SR problem is the inverse of the degradation process, so the former one depends on the later. One of the most used degradation process is blurring, downsampling and the addition of noise, where most common noise addition is the white Gaussian noise and the blurring is usually done by convolving the high resolution image with a Gaussian kernel. The process can be represented as

$$Y = SBX + N \tag{2.3}$$

where S is the downsampling, B the blurring and N the noise addition.

However, regardless of how sophisticated the degradation process is, the real world is more complex and therefore, there are countless factors that the artificial degradation process does not take into account. In consequence, the reconstructions of SR real-world images do not have as good results as predictions of the downsampled images may have.

2.4 Evaluation of single image super-resolution algorithms

As the SISR algorithms have been increasingly developed in recent years, the methods by which these algorithms' performance is evaluated have also been a focus in research. Depending on whether the human is involved in said evaluation, the methods can be divided into two categories: subjective evaluation and objective evaluation. The evaluator of the subjective evaluation is human while the objective evaluation uses a mathematical model to evaluate. Currently, the most used objective evaluation methods include structural similarity index (SSIM), peak-signal-to-noise-ratio (PSNR), root mean squared error (RMSE) and perceptual index (PI).

2.4.1 Structural similarity Index (SSIM)

The SSIM metric, first introduced in a 2004 IEEE paper [14], measures the perceptual difference between two similar images. However, this metric can not deduce which one is better, so it has to be indicated which one is the original and which one has been processed. This aims to replicate humans visual perception system, which is highly capable of identifying structural information from a scene and thus identifying the differences between the information extracted from a reference and a sample scene.

The metric gives a value between -1 and 1. The higher the score, the more similar both images are. It is common to adjust the value range to [0,1].

The SSIM consists of three features from an image: brightness, contrast and structure. Firstly, brightness is measured by averaging all the pixel values. It is denoted by μ :

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.4}$$

where x is the input image. The brightness comparison function $l(x, y)$ is then a function of μ_x and μ_y.

Secondly, the contrast is measured by taking the standard deviation of all the pixel values. It is denoted by σ, as shown in the following equation:

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \tag{2.5}$$

The contrast comparison $c(x, y)$ is then the comparison of σ_x and σ_y.

Lastly, the signal is normalised by its own standard deviation and the structure comparison $s(x', y')$ is calculated on these normalised signals:

$$x' = \frac{(x - \mu_x)}{\sigma_x}, y' = \frac{(y - \mu_y)}{\sigma_y} \tag{2.6}$$

Finally, the SSIM function is defined as presented in:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{2.7}$$

where $l(x, y)$ compares the brightness, $c(x, y)$ compares the contrast and $s(x', y')$ compares the structure.

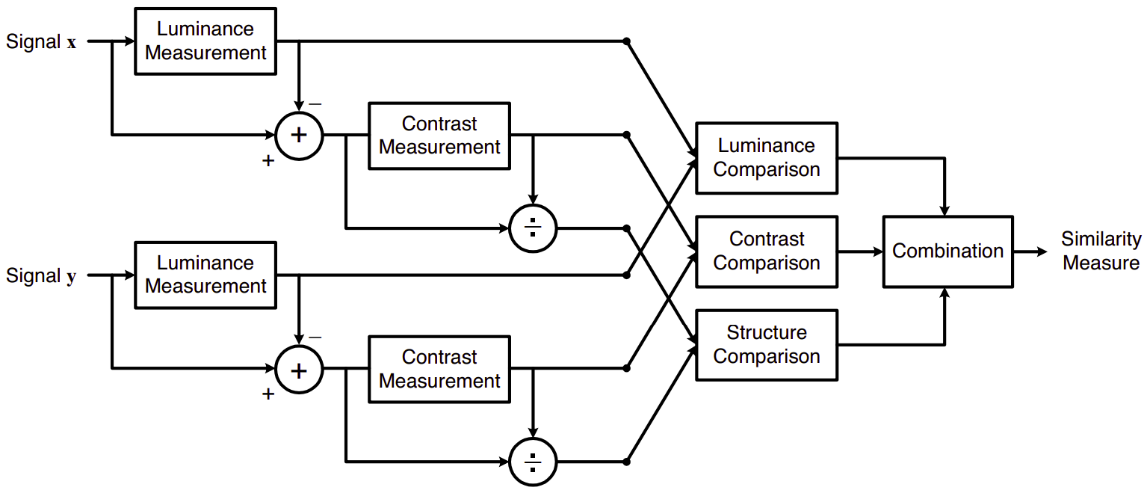


Figure 2.3: Diagram of the structural similarity (SSIM) measurement system. Firstly, the luminance (or brightness) is measured for both images, which are then used in the luminance comparison function ($l(x, y)$). Secondly, the contrast is measured for both images and the results are used in the contrast comparison function ($c(x, y)$). Lastly, the signals are normalised using the previously calculated luminances and contrasts and they are used to make the structure comparison. Finally, the three comparison functions are combined to create the SSIM function. [3]

[3]Figure taken from: [14]

It is important to mention that applying this metric regionally can obtain a more accurate evaluation than applying it globally. In other words, it is better to apply it in small sections of the image and taking the mean overall, than applying it all over the image. This is because the feature statistics of an image are usually unevenly distributed in the pixel space and, furthermore, it replicates more truthfully the characteristics of the human visual system, as human eyes tend to focus on a region of an image rather than on the whole image.

2.4.2 Multi-scale structural similarity (MS-SSIM)

This metric, developed by Wang et al. [15], is a slightly improved version of the previously mentioned SSIM (see Section 2.4.1). It is based on the premise that in the real world, the sampling density of the image signal, the distance from the image plane to the observer, and the perceptual capability of the observer's visual system are elementary components that influence the perceivability of image details, and therefore, in practice the subjective evaluation of a given image varies when these factors vary.

With the aim of addressing this matter, the MS-SSIM simulates different spatial resolutions by iterative downsampling and weighting the luminance, contrast and structure components of the SSIM at different scales, which has proven to be more accurate than SSIM for some circumstances [16].

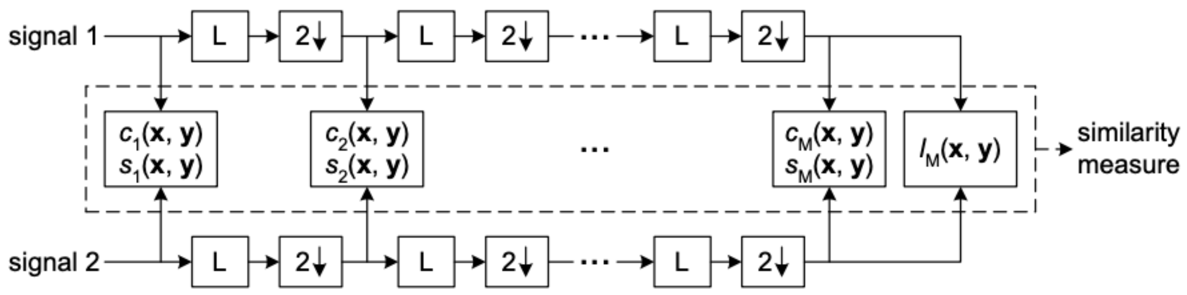


Figure 2.4: Multi-scale structural similarity measurement system. L: low-pass filtering; 2 ↓: downsampling by 2. The MS-SSIM system iteratively downsamples and calculates the luminance, contrast and structure of both signals (images). [4]

[4]Figure taken from: [15]

2.4.3 Peak-signal-to-noise-ratio (PSNR)

The peak-signal-to-noise-ratio [17] represents the ratio between the maximum possible value of a signal and the power of distorting noise that affects the quality of its representation. In the specific case of images, that value is 255. It is represented by the following formula:

$$PSNR = 20 * log\left(\frac{MAX(f)}{\sqrt{MSE}}\right) \tag{2.8}$$

where the MSE is:

$$MSE = \frac{1}{MN}\sum\nolimits_{n=1}^{M}\sum\nolimits_{m=1}^{N}\left[\widehat{g}(n,m) - g(n,m)\right]^2 \tag{2.9}$$

where $\widehat{g}$ and g refer to the resulting image and original image respectively, and their sizes are $M(height) \times N(width)$. MAX is the maximum value of the pixels, as said before, 255.

The larger the PSNR value, the smaller the difference between the predicted image and the original image, which means the better the image quality. Nevertheless, as opposed to SSIM, this method is based on global statistics of the pixels, so the human eyes' characteristics are not taken into account.

2.4.4 Root mean squared error (RMSE)

The root mean squared error has also been a frequently used image quality metric. This metric is scale-dependent accuracy measure, which means that it is very sensitive to large or small errors in measurements, therefore it cannot be used to make comparisons between series using different scales. It is represented as follows:

$$RMSE = \sqrt{MSE} \tag{2.10}$$

where the MSE is as defined in Equation (2.9)

2.5 Convolutional neural networks

Before explaining the state-of-the-art methods for the super-resolution task, it is important to describe the Convolutional neural networks (CNN). The CNN is a Deep Learning algorithm that was developed for use in image processing and computer vision. It is inspired by the functioning of the animal visual cortex, which can identify objects and characteristics in the images it sees. Research performed from the 1950s to the 1980s established that vision is processed through a sequence of layers. Each neuron in the first layer takes input from a small region of the visual field (its receptive field). Different neurons are specialised to detect particular local patterns or features, such as vertical or horizontal lines. In the second layer, cells take input from cells in the first layer, combining their signals to detect more complicated patterns over a larger receptive field. Hence, each layer can be considered as a representation of the first input image.

CNNs emulate this design, by taking an image as an input and feeding it to every layer of the model. Hence, CNNs are similar to the Multilayer perceptron (MLP), but with a slight difference. MLP use fully connected layers, meaning that every element of the output depends on every element of the input. On the other hand, CNNs use convolutional layers that take advantage of spatial locality. Each output element corresponds to a small region of the image, and only depends on the input values in that region, which can reduce the number of parameters of the model. Moreover, CNNs assume that the parameters are the same for every local region of the image, so if a layer uses one set of parameters to detect vertical lines in a local region, it uses the same parameters to detect vertical lines in every region. Consequently, the number of parameters for the layer is independent of the size of the image. In addition, this network has to learn a convolutional kernel which has the function of defining how output features are computed from a local region of the image.

On the other hand, each layer is composed by neurons, which are mathematical functions that calculate the weighted sum of multiple inputs and output an activation value , with the aim of highlighting the relevant features of the image. The first layer of the CNN detects basic features such as horizontal, vertical, and diagonal edges. The second one extracts more complex features, for instance corners. Therefore, each layer detects more complex features than the previous one, so when a layer reaches certain degree of complexity it can detect higher-level features such as objects, eyes or faces.

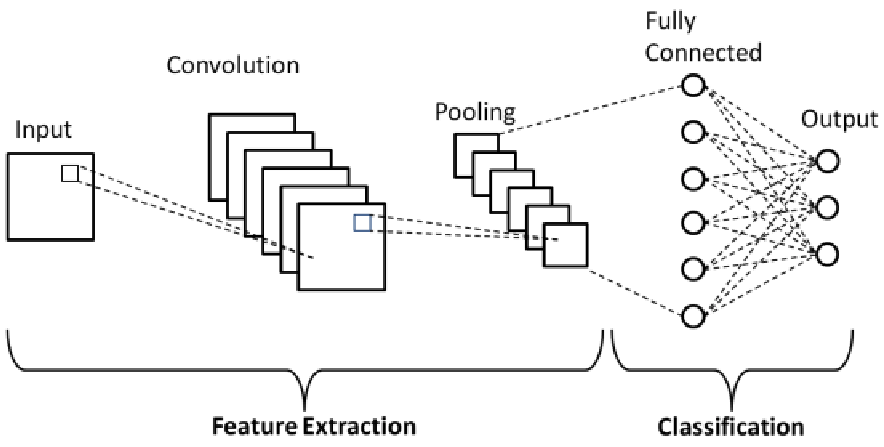


Figure 2.5: Example diagram of a basic convolutional neural network (CNN) architecture. CNNs combine both feature extraction and classification. The network consists of five different layers: input, convolution, pooling, fully-connected, and output. The input layer specifies a fixed size for the input images. The image is then convolved with multiple learned kernels using shared weights. Next, the pooling layers reduce the size of the image while trying to maintain the contained information. These two layers constitute the feature extraction part. Afterwards, the extracted features are weighted and combined in the fully-connected layers, which represents the classification part of the CNN. Finally, there is one output neuron for each category in the output layer. [5]

[5]Figure taken from: [18]

CHAPTER 3

State of the art

3.1 Deep learning based super-resolution

In recent times, DL-based SISR algorithms have demonstrated great superiority to aforementioned methods (see Section 2.2). The first deep learning architecture to solve the SISR task was the super-resolution convolutional neural network (SRCNN), presented by Dong et al. [19] This architecture is a three layer convolutional neural network (CNN), where the filter sizes of each layer are $64 \times 1 \times 9 \times 9$, $32 \times 64 \times 5 \times 54$ and $1 \times 32 \times 5 \times 5$.

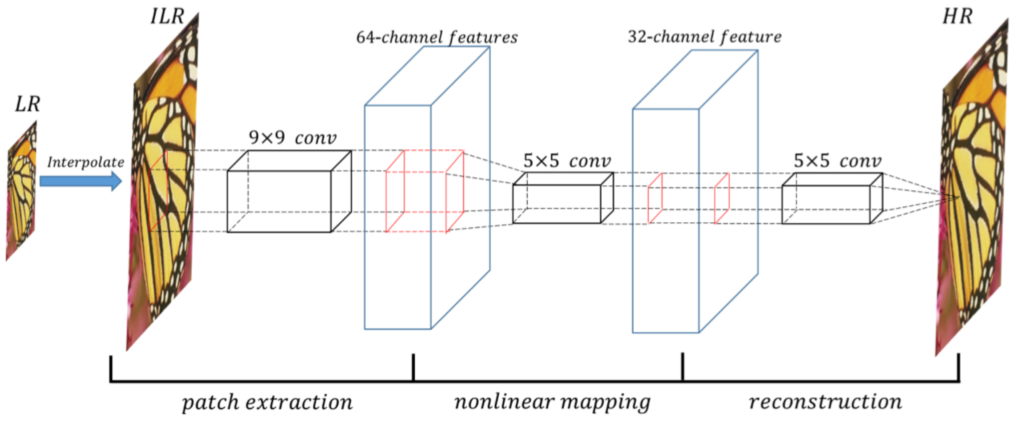


Figure 3.1: Diagram of the SRCNN architecture. The LR image is fed into the model going through three layers, each of which has a different task: patch/feature extraction, nonlinear mapping and reconstruction. [1]

The model's objective is to find an optimal model to predict SR from unobserved examples, having previously trained on a training set of LR-HR images. The SRCNN comprises the following steps:

1. **Preprocessing**: Upscale the LR image to desired HR image using bicubic interpolation.

2. **Feature extraction and representation**: This operation extracts patches from the LR images and represents each patch as a high-dimensional vector. These vectors include a set of feature maps and the number of said feature maps is equal to the dimension of the vector. The first convolutional layer is represented by the following equation:

$$F_1(y) = max(0, W_1 * Y + B_1) \tag{3.1}$$

   where W_1 represents the filters and B_1 the offsets. The size of filters W_1 is $c \times f_1 \times f_1 \times n_1$, where c is the number of channels in the input image, f_1 is the channel size of the filter, n_1 is the number of filters, and B_1 is a vector of n_1 dimensions, each of which is associated with the filters.

3. **Non-linear mapping**: This operation maps the features between the LR and HR patches, specifically, it nonlinearly maps each high-dimensional vector onto another high-dimensional vector. These vectors comprise another set of feature maps. The first layer extracts an n1-dimensional feature for each patch. In the second operation, we map each of these n_1-dimensional vectors into an n_2-dimensional one. The operation of the second layer is:

$$F_2(Y) = max(0, W_2 * F_1(Y) + B_2) \tag{3.2}$$

   Here W_2 contains n_2 filters of size $n_1 \times f_2 \times f_2$, and B_2 is n_2-dimensional. To increase non-linearity, it is possible to add more convolutional layers, but it can increase the complexity of the model ($n_2 \times f_2 \times f_2 \times n_2$ parameters for one layer), resulting in more training time.

4. **Reconstruction**: In the traditional methods, the final full image is produced by averaging the predicted overlapping high resolution patches. Inspired by this

[1]Figure taken from: [20]

procedure, this operation aggregates the HR patch representations to generate a final HR image that is as similar as possible to the original HR image. It can be expressed by the following equation:

$$F(Y) = W_3 * F_2(Y) + B_3 \tag{3.3}$$

Here W_3 corresponds to c filters of a size $n_2 \times f_3 \times f_3$, and B_3 is a c-dimensional vector.

Although this architecture is composed by only three layers, its results substantially outperforms other algorithms not based on deep learning, which can be attributed to the CNN's capacity to learn valid representations from big data. Regardless of the accomplishment of the SRCNN, it presents some problems that led to investigate other architectures:

1. The input of SRCNN, the bicubic LR, has some inconveniences: (1) detail-smoothing effects introduced by these inputs can result in wrong estimations of the image structure; (2) employing the interpolated versions as input is very time-consuming.
2. The SRCNN only has three layers, so it is natural to enquire if the addition of more layers can improve the performance.

3.2 Deep single image super-resolution models

3.2.1 Very deep convolutional network (VDSR) and deeply recursive convolutional network (DRCN)

In light of the success of deep convolutional networks, Kim et al. proposed two models: very deep convolutional network (VDSR) [21] and deeply recursive convolutional network (DRCN) [22], both of them composed by 20 convolutional layers. The VDSR network (see Figure 3.2) was presented as an improvement of the SRCNN network, which aimed to solve the following problems:

1. Reliance on the context of small image regions. For a large scale factor, the information contained in a small patch is not sufficient for detail recovery.
2. Slow training convergence.
3. The network only works for a single scale. Each SRCNN model is trained to operate for a single scaling factor, so a model has to be trained and stored for each scaling factor, which is not practical.

For the first problem, the VDSR uses a large receptive field that takes a large image context into account. In convolutional neural networks, hidden units of each layer take as input a subset of units in their previous layer and thus, they form spatially contiguous receptive fields. The architecture ensures that the learned filters produce the strongest response to a spatially local input pattern. In addition, adding more layers leads to filters that become increasingly global, making them responsive to a larger region of pixel space. In [21], Kim et al. demonstrate that the size of the receptive field is proportional to the models depth and that, in the SR task, this corresponds to the amount of contextual information that can be used. In other words, the larger the receptive field, the better, since it means that the network can use more context to predict image details.

On the other hand, to hasten the speed of the convergence, the learning rates were highly increased by using a learning rate of 10^{-1} instead of the SRCNN learning rate, 10^{-4}. However, this boost in the learning rate can lead to exploding gradients, and therefore, to solve this issue, residual-learning and gradient clipping are used.

In SRCNN, the network must conserve all input detail since the image is discarded and the output is generated solely from the learned features. This approach is adequate for that network, since it is composed by only three layers. In contrast, the VDSR has 20 layers, requiring very long-term memory. Residual learning tackles this problem, so that instead of predicting the whole image the model learns the difference between inputs and outputs, resulting in a faster convergence and better accuracy.

To address the third problem, the proposed method is modifying the scaling factor while training, which proves to be effective. Therefore, to verify this statement, the VDSR was trained with a single scaling factor and with scale augmentation, which confirmed that the latter one could cope with any scale used during training.

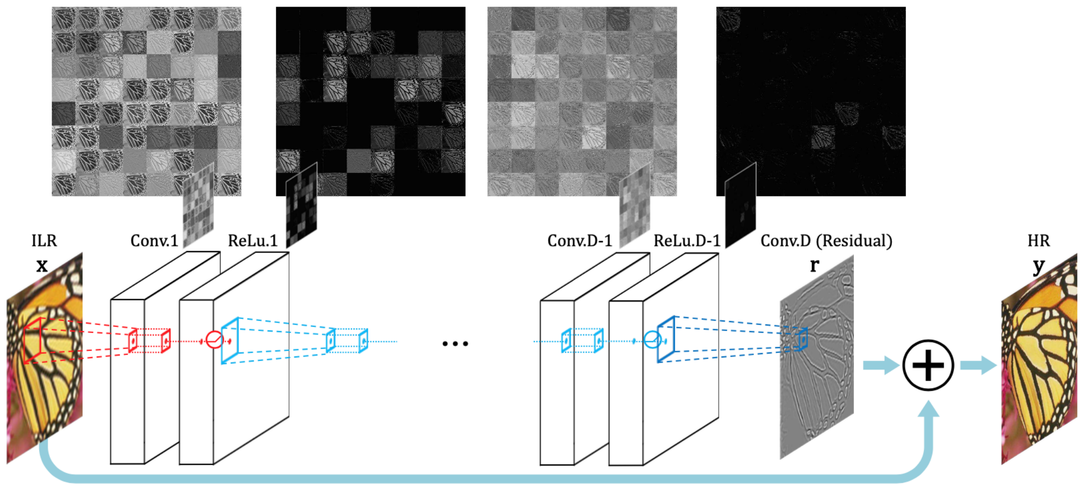


Figure 3.2: VDSR network structure. Composed by pairs of convolutional and nonlinear layers repeatedly. [2]

However, the convolution kernels in the nonlinear mapping part (see step 3 in Section 3.1) are very similar, and thus Kim et al. further proposed DRCN [22] with the aim of reducing the parameters. In [22], it is stated that increasing depth by adding new weight layers introduces more parameters, which can trigger two problems: overfitting and a huge increase in the weight of the model, difficulting its storage. Consequently, the DRCN repeatedly applies the same convolutional layer several times, meaning that the number of parameters stays constant even if more recursions are performed.

[2]Figure taken from: [21]

The network consists of three parts: embedding network, inference network and reconstruction network. The embedding network takes the input image and represents it as a set of feature maps (see Figure 3.3).

The inference network is the main component that solves the super-resolution task. A single recursive layer analyses a large image region and each recursion applies the same convolution followed by a rectified linear unit (ReLU). With convolution filters larger than 1×1, the receptive field is widened with every recursion, and thus, as explained before for the VDSR, the network uses more context to predict image details. Finally, the reconstruction network transforms the multi-channel feature maps into the original image space (one or three channels).

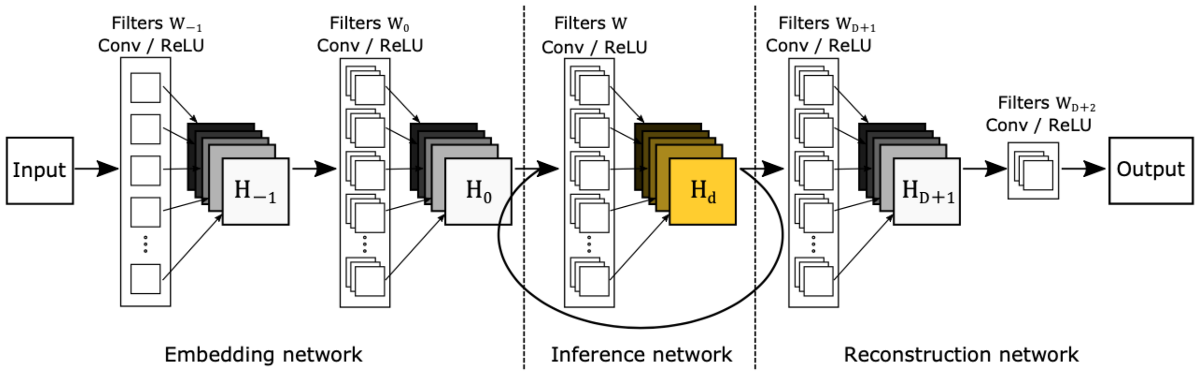


Figure 3.3: Architecture of the DRCN basic model. [3]

3.2.2 Enhanced deep residual network (EDSR) and Wide-activated deep super-resolution network (WDSR)

The Enhanced deep residual network (EDSR), proposed by Lim et al. [3], is based on the improvement of the SRResNet [2]. This network incorporates some enhancements with regard to the SRResNet. Firstly, the EDSR removes the usage of batch normalisation (BN) in the residual blocks (see Figure 3.4). This layer comes from the initial ResNet [1], which was first proposed to solve other higher leveled computer vision problems such as classification and detection, so its performance is not as good for a low-level computer vision problem as super-resolution. Moreover, the BN layer consumes as much memory as the convolutional layer in front of it, so discarding it can save memory resources. Secondly, since the first improvement results in saving

[3]Figure taken from: [22]

memory, the EDSR stacks more layers and it also increases the number of output features for each layer, which leads to a better performance with the same computing resources as the SRResNet. Lastly, during training, the model was trained for a scaling factor of ×2 for the purpose of using the generated weights to train the model for ×3 and ×4 scaling factors, a strategy that reduced the training time and improved the final performance.

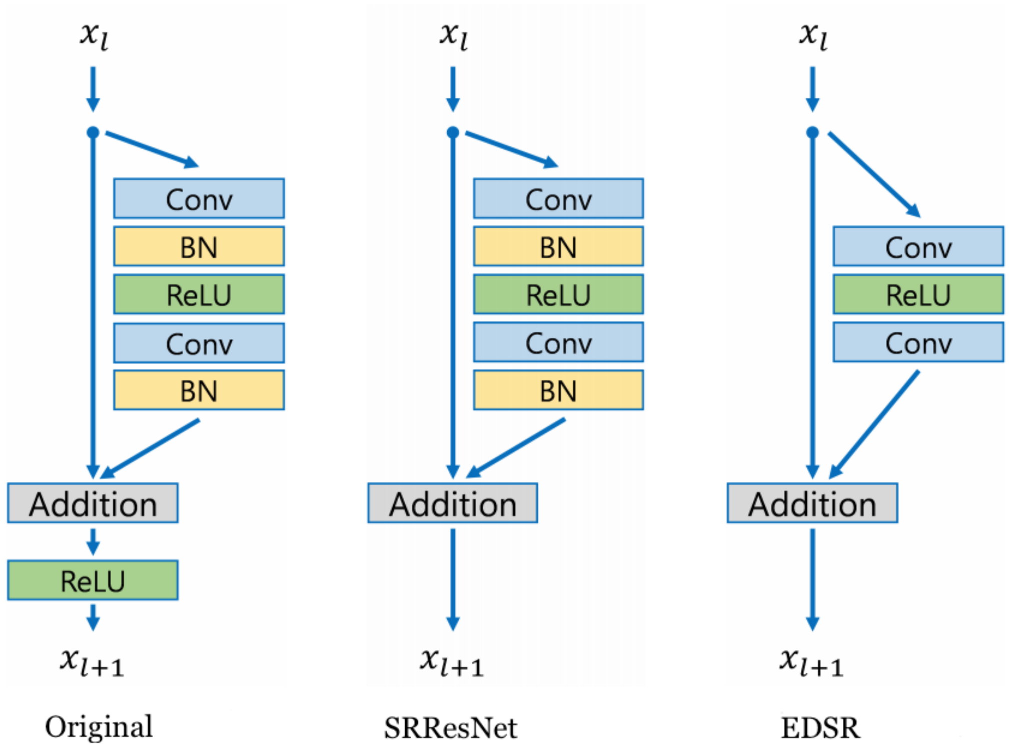


Figure 3.4: Comparison of residual blocks in original ResNet [1], SRResNet [2], and EDSR [3]. As it can be seen, the EDSR removes the batch normalisation blocks on the grounds that it is not useful for low-level computer vision tasks. The disposal of these layers contributes to a reduction of memory usage. [4]

However, JiaHui Yu et al. presented the Wide-activated deep super-resolution network (WDSR) in [23], an improvement of the EDSR. The WDSR introduced two major improvements: the first one is to remove all the redundant convolutional layers (see Figure 3.5) which increases the training speed and reduces the memory. Secondly, the usage of weight normalisation. In [23], JiaHui Yu et al. demonstrates that this strategy obtains better results than batch normalisation (BN) or no normalisation. Weight normalisation consists of rewriting each weight W by multiplying

[4]Figure taken from: [3]

it by a scalar s and the vector v, fixing the v vector. Additionally, using weight normalisation does not need further storage.

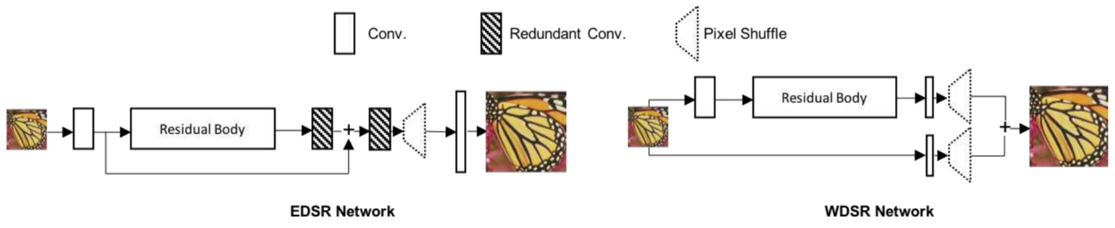


Figure 3.5: Comparison of EDSR (left) and WDSR (right) architectures [5]. The WDSR introduces two improvements in relation to the EDSR: removing all the redundant convolutional layers and the usage of weight normalisation.

3.2.3 Residual channel attention network (RCAN)

Another deep learning network is the residual channel attention network (RCAN) network, proposed in [6], by Zhang et al. The main statement in [6] is that the deeper the networks, the more difficult it is to train them. The low resolution inputs contain a considerable amount of low-frequency information, which is treated equally across channels, thus hindering the representational ability of CNNs. In addition, most of the preceding networks do not have a large network depth, a characteristic that has demonstrated to be very influential in computer vision tasks.

Therefore, with the aim of creating a very deep network with superior results, the residual in residual (RIR) structure is proposed, which consists of a collection of residual groups (RG) with long skip connections where each of these groups contains a set of simplified residual blocks with short skip connections. This structure allows an ample low-frequency information to be bypassed through these skip connections, permitting the main network to focus on the high-frequency information. Moreover, the structure allows to train very deep CNN with high performance. As a matter of fact, the RCAN has been the deepest model for the SISR task, with over 400 layers.

Additionally, the network implements a channel attention (CA) mechanism (see Figure 3.6), whose task is to focus on the most informative features exploiting the interdependencies among feature channels. The first operation is a global average pooling to get the channel with a size of $1 \times 1 \times C$, which is a channel descriptor

[5]Figure taken from: [23]

containing rough information. Then it divides the channel by the ratio r, which is a downsample, and then it upsamples to get the weight coefficient of each channel. The final operation is to multiply the original feature from the residual to obtain a new feature that has been redistributed to the channel weight. This technique allows to control how much information is passed up to the next layer in the hierarchy.

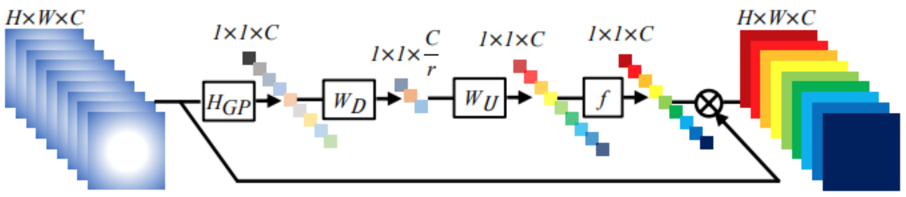


Figure 3.6: Channel attention (CA). $\otimes$ denotes element-wise product and H_{GP} the global average pooling. [6]

Finally, both mechanisms, the RIR and the CA, are combined resulting in the residual channel attention block (RCAB), with which the RCAN network has been created.

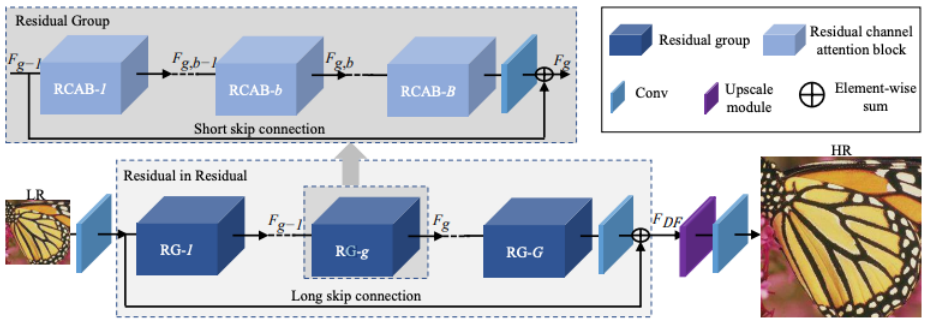


Figure 3.7: RCAN network architecture. [7]This network is composed by a collection of residual groups (RG) with long skip connections where each of these groups contains a set of simplified residual blocks with short skip connections. The network implements a channel attention (CA) mechanism, whose task is to focus on the most informative features exploiting the interdependencies among feature channels. These both mechanisms are then combined creating the residual channel attention block (RCAB).

[6]Figure taken from: [6]
[7]Figure taken from: [6]

3.3 Democratising deep learning for bioimage analysis: ZeroCostDL4Mic

As previously stated (see Section 3.1), deep learning has been a powerful tool for the super-resolution task, since it has brought great progress. However, the training of DL neural networks requires certain resources that beginners can find hard to acquire. Moreover, the creation and use of these methods demand a minimum knowledge in the computer science field, thus remaining out of reach for most researchers. The training of DNNs often needs local servers with high computational capability or expensive DL-ready workstations, which implies financial resources and a commitment to maintain them. Another option is to purchase computational resources provided by cloud services, but the fact remains that not everyone has the resources to train a DL network.

In case of not training, there is the option of using a pretrained network for some common tasks such as cellular segmentation or super-resolution. However, if the datasets to be predictedwor bear no resemblance to the ones with which these models have been trained, it is most likely that the prediction will not be correct, since it has been proved that models perform best on datasets similar to the training dataset. Therefore, this fact leads again to the need of having financial resources and the barrier it entails for several potential users.

Hence, with the purpose of overcoming this barrier, the team of ZeroCostDL4Mic[8] has developed an entry-level, cloud based platform that allows most users to have access to DL technology, and it is specifically targeted for microscopy DL. ZeroCostDL4Mic is a collection of fully annotated Jupyter notebooks with an easy-to-use graphical user interface. The only requirement to be able to use them is having a Google account, since it provides all the requisites to run the notebooks. One of the objectives of ZeroCostDL4Mic is to provide researchers and experts with no coding background with the ability to use DL, so it is not necessary any prior knowledge in coding.

Google colab [9] is a highly used tool among data mining and deep learning experts, yet it is necessary to have coding and deep learning knowledge in order to derive full benefit from it. By creating a user-friendly interface, the team of ZeroCostDL4Mic

[8]Link to ZeroCostDL4Mic site
[9]Link to Google Colab

achieves the goal of leveraging this tool so that more people can take advantage of it. Google Colab provides free access to remote virtual machines with a maximum runtime duration of 12 hours, the maximum time for each session. The virtual machines include a storage of 68GB, which can be used to save training data, models and results, 12 or 25 GB of RAM, which depends on the session, and access to high quality GPUs, commonly Tesla P100, T4 or K80.

The notebooks provided by the team cover a range of important image analysis tasks (e.g. segmentation, denoising, restoration, label-free prediction). Although each of them is different, they follow the same structure pattern, so that the main steps are the following:

1. Installing the relevant libraries, which takes less than a minute.
2. Loading the training datasets.
3. Training the model. The training time varies on the model.
4. Validating the data. Once trained and validated, the models can be downloaded and used on other data.

However, there are limitations in the use of Google Colab. For example, the availability of RAM determines the number of images that can be used in the training and the runtime duration limit of 12 hours constraints the number of epochs. Nevertheless, the ZeroCostDL4Mic notebooks can be efficiently trained with good results with the Google Colab resources, but it is important to mention that these notebooks are best suited for small-scale studies with microscopy data (a few 10GB of data).

For the development of larger scale studies, the team recommends an investment in local infrastructure or cloud-based platforms, and since the ZeroCostDL4Mic notebooks are not dependent on Google Colab, they can be ported to any platform that supports Jupyter notebooks.

In any case, for novice users or students, it is recommended to explore and investigate with the free resources before deciding to invest in paid-for platforms or local infrastructure.

CHAPTER 4

Model description / Methods

4.1 Deep Fourier Channel Attention Network layers

As previously mentioned in Section 1.1.1, the DFCAN [5] is a network influenced by the RCAN network, more specifically, inspired by the spatial domain channel attention mechanism employed in the RCAN (see Section 3.2.3). The main difference between both of them is that the residual channel attention block is replaced by the Fourier channel attention block. The goal of these DL networks is to extract the most relevant features of the images, i.e., the high-frequency information, so the authors hypothesise that instead of using the structural differences in the spatial domain, leveraging the frequency content differences across distinct features in the Fourier domain might enable the DLSR networks to learn hierarchical representations of high-frequency information more precisely.

As shown in Figure 4.1, the DFCAN architecture starts with a convolutional layer and a Gaussian error linear unit (GELU) [24] which is defined as:

$$GELU(x) = 0.5 \cdot (1 + erf(x/\sqrt{2})) \tag{4.1}$$

where erf is the following error function:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{4.2}$$

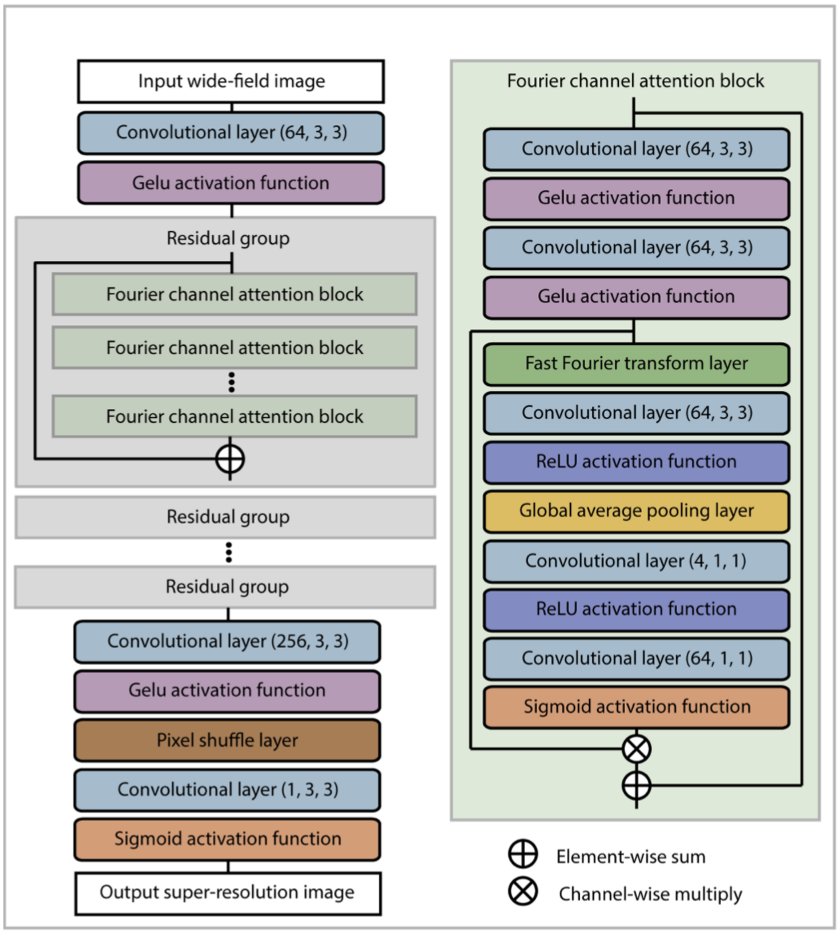

Input wide-field image
Convolutional layer (64, 3, 3)
Gelu activation function
Residual group
Fourier channel attention block
Fourier channel attention block
Fourier channel attention block
Residual group
Residual group
Convolutional layer (256, 3, 3)
Gelu activation function
Pixel shuffle layer
Convolutional layer (1, 3, 3)
Sigmoid activation function
Output super-resolution image
Fourier channel attention block
Convolutional layer (64, 3, 3)
Gelu activation function
Convolutional layer (64, 3, 3)
Gelu activation function
Fast Fourier transform layer
Convolutional layer (64, 3, 3)
ReLU activation function
Global average pooling layer
Convolutional layer (4, 1, 1)
ReLU activation function
Convolutional layer (64, 1, 1)
Sigmoid activation function
Element-wise sum
Channel-wise multiply

Figure 4.1: DFCAN architecture. The left diagram shows the general architecture encapsulating the Fourier Channel Attention Blocks (FCAB). The right diagram shows the FCAB architecture.[1]

Afterwards, the output of the GELU is fed to four identical residual groups, each of them composed by four Fourier channel attention blocks (FCAB) and a skip connection. The residual group is denoted as:

$$RG(x) = x + FCAB^4(x) \tag{4.3}$$

where x is the input feature maps of the RG and $FCAB^4$ is defined as:

$$FCAB^4(x) = FCAB(FCAB(FCAB(FCAB(x)))) \tag{4.4}$$

Then, the output of the RG is fed into another convolutional layer followed by a

[1]Figure taken from: [5]

GELU activation function. The next three layers, i.e., the pixel shuffle [25] layer, the convolutional layer and the sigmoid activation function are responsible of upscaling the image so that the size is the same as the ground truth image. Finally, the DFCAN outputs the predicted super-resolution image.

4.2 Loss function

Another difference between the DFCAN and RCAN is the loss function. The DFCAN loss function is defined as a combination of the SSIM loss and the MSE loss. On one hand, the MSE loss improves pixel accuracy and equalises the dynamic range of prediction. On the other hand, the SSIM loss refines the structural similarity of the output, so the idea is that the combination of the two loss functions can solve both problems. Therefore, the loss function can be formulated as:

$$L_{DFCAN}(\hat{Y}, Y) = MSE(\hat{Y}, Y) + \lambda[1 - SSIM(\hat{Y}, Y)] \tag{4.5}$$

where $\hat{Y}$ is the output image of the DFCAN, Y the ground truth image and λ a scalar weight with the function of balancing the contribution of the SSIM and MSE, which is usually set to 0.1 by the authors.

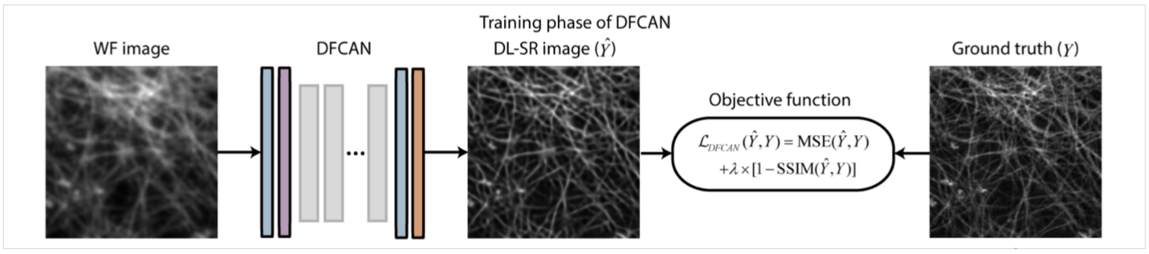

Training phase of DFCAN
WF image
DFCAN
DL-SR image ($\hat{Y}$)
Objective function
$\mathcal{L}_{DFCAN}(\hat{Y},Y) = \mathrm{MSE}(\hat{Y},Y) + \lambda \times [1-\mathrm{SSIM}(\hat{Y},Y)]$
Ground truth (Y)

Figure 4.2: Diagram of the training process of the DFCAN. The low resolution image is fed into the network, which outputs a super-resolution image prediction. The loss function (or the objective function) evaluates the quality of the model taking into account the predicted image and the ground truth image on each epoch in order to improve the performance on the next epoch. [2]

[2]Figure taken from: [5]

CHAPTER 5

Experiments

The objective of this chapter is to show the functionality of the DFCAN network and validate its implementation and application. For a selected set of images, the quality of the images produced by DFCAN will be evaluated taking into account some evaluation metrics mentioned in Section 2.4, some visual comparisons between the LR images, HR images and their respective predictions and the training time for each model.

The chapter starts with an explanation of the dataset provided by Chang et al. in [5]. Secondly, Section 5.2 explains how said dataset has been preprocessed in order to start with the training. Finally, Section 5.3 presents all the experiments and their corresponding parameters used to train the network. This section is divided into different subsections, one for each different set of experiments.

To start with, Section 5.4.1 presents the experiments done using the DFCAN network and F-actin dataset, which will be later explained in Section 5.1. The second set of experiments, displayed in Section 5.4.2, are done by using the RCAN network, another state-of-the-art network for the SISR problem (see Section 3.2.3), with the purpose of comparing the performance of both DFCAN and RCAN networks on the same dataset. Lastly, in Section 5.4.3, the experiments are conducted using the DFCAN network with a dataset external to the one provided by the authors, with the aim of analysing the network's performance on a dataset that has not been tested by the DFCAN's authors.

5.1 Dataset description

The dataset used for the experiments is the one provided by Chang et al. in [5]. In order to create the dataset, the authors employed a structured illumination microscopy (SIM) system, thus obtaining well matched pairs of LR-HR images. This dataset is called BioSR and it is divided into different subdatasets depending on the biological structure they contain. The different structures are clathrin-coated pits (CCPs), endoplasmic reticulum (ER), microtubules (MTs) and F-actin filaments, listed in order of structural complexity, from the simpler structure (CCPs) to the more complex one (F-actin) (See Figure 5.1). Each subdataset contains 50 sets of raw SIM images, each of them at ten escalating levels of excitation light intensity. It is of interest mentioning that the images with the highest excitation level are the best predicted by the DFCAN.

For the experiments of this work specifically, the F-actin dataset was used, the one with the most complex structure in the BioSR dataset. Since the dataset was obtained by using the multimodality structured illumination microscopy, the images are in raw format, meaning that the processing of the images is minimal and that they preserve all of their information. This ensures a better processing of the images later on.

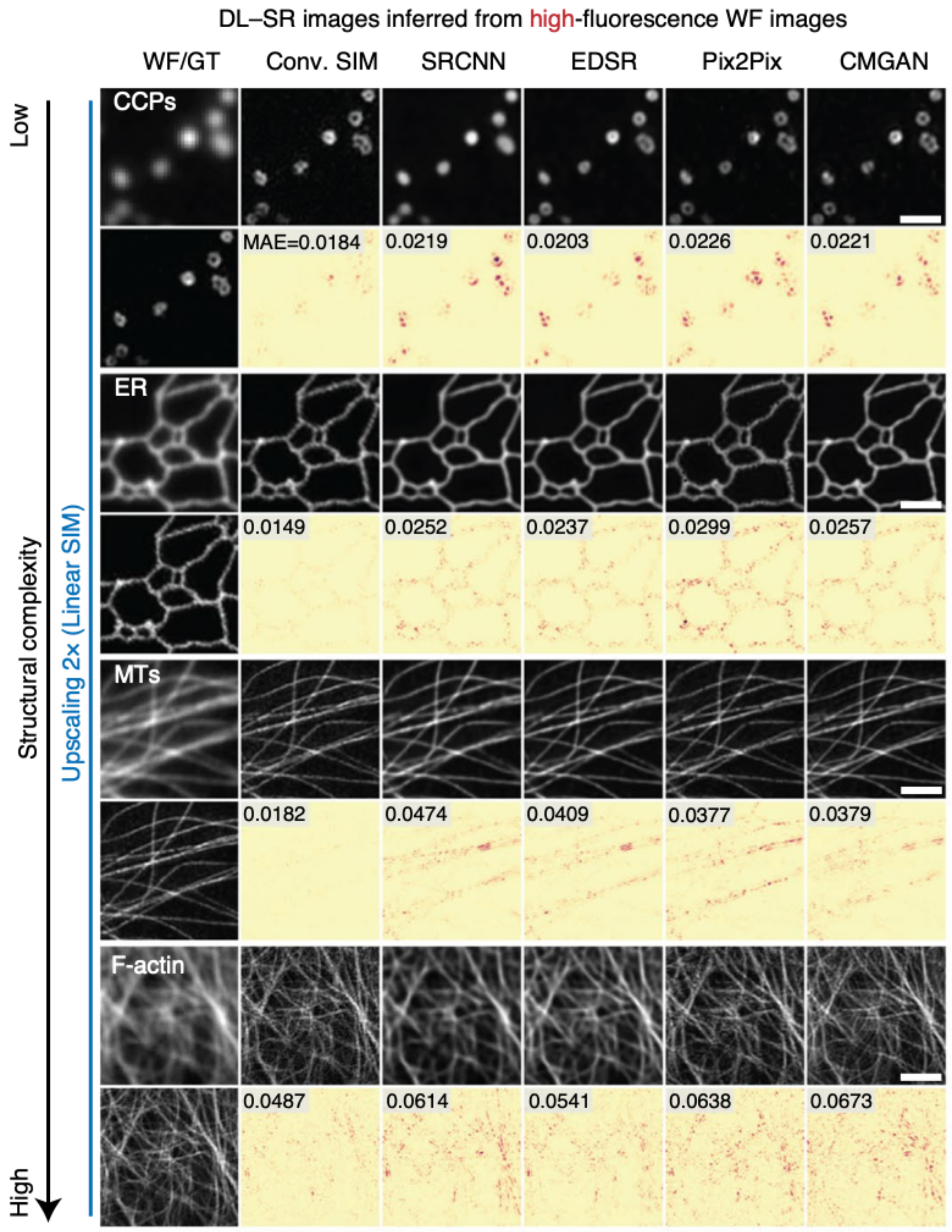


Figure 5.1: Examples of the BioSR dataset, grouped by the different biological structures (CCPs, ER, MTs, F-actin), and ordered by the increasing structural complexity. The first image of each group represents the low resolution image, and the rest on the right are the predicted outputs by different networks for super resolution.[1]

[1]Figure taken from: [5]

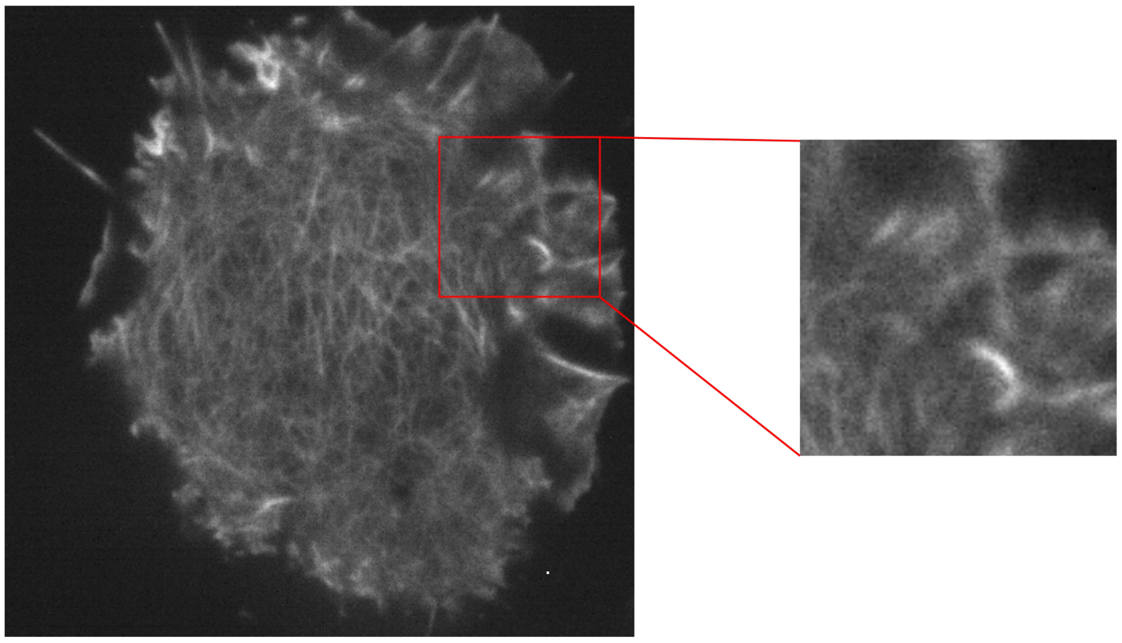
5.2 Dataset preprocessing

Firstly, considering that the images are provided in raw format, the first step before working with the dataset was transforming them into another format, in this case, into Tag Image File Format (TIFF) format. For that purpose, the authors of the network provide a Matlab[2] script that processes the raw images. Additionally, the processing of the images includes data augmentation, but in this case the data augmentation was done later on the notebook, therefore the code was modified so that it only converted the images from raw to TIFF without the augmentation.

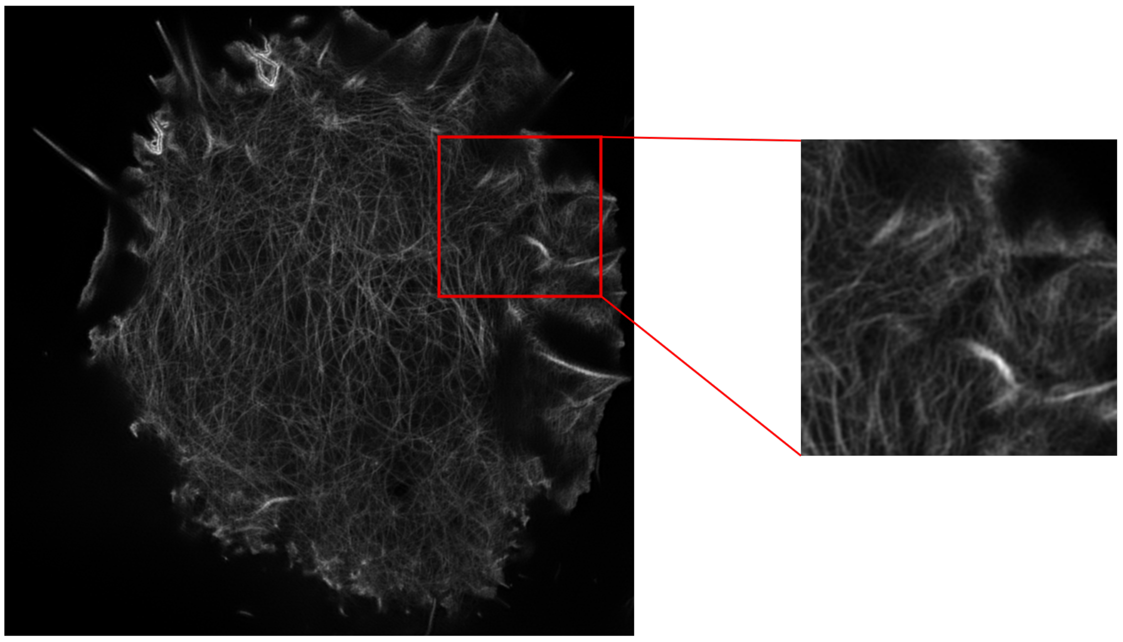
Since the Matlab script makes a partition to separate the training and test sets, the dataset obtained after converting the images to TIFF format contains 492 pairs of LR-HR images for the training and 120 pairs of LR-HR images for the testing. Regarding the dimensions, the LR and HR images have the dimensions of 502×502 pixels and 1004×1004 pixels respectively, thus the scaling factor is 2. The images in Figure 5.4 are an example of an LR-HR pair with their respective dimension axes.

On the other hand, with the purpose of having a more assorted dataset, data augmentation was applied. The first step in the notebook after loading the images was to extract several samples from each original image. This was done by taking random crops from the images respecting the scaling factor, so the dimensions of the crops are 256×256 for the HR images, and 128×128 for the LR images. However, it is important to bear in mind that even though the crops are random, the sample has to be the same for the LR image and its respective HR image, so the function responsible for that task takes the same pair of crops from each pair of images. In Figure 5.2, there is an example of a pair of extracted patches from the respective pair of images.

[2]Matlab website



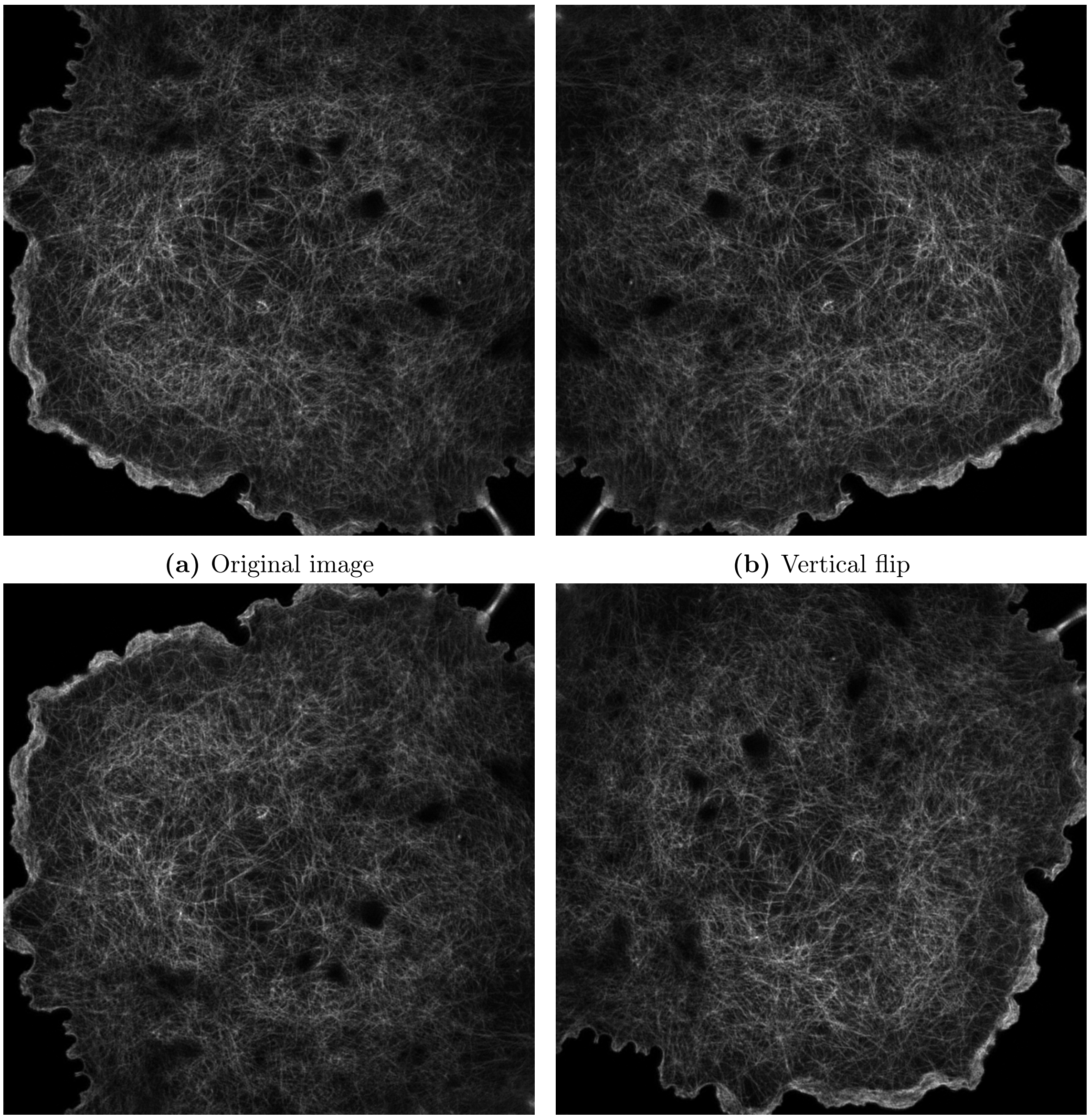
(a) Extracted patch from a low resolution image. LR image dimensions: 502 × 502; LR patch dimensions:128 × 128



(b) Extracted patch from a high resolution image. HR image dimensions: 1004 × 1004; HR patch dimensions: 256 × 256

Figure 5.2: Example of a patch extraction. Even though the extracted regions are arbitrary, for each extracted patch, the region of the LR and HR patches have to be the same, so that the network will train with the LR patches and then compare their prediction with the HR counterpart patches

After doing the cropping, the new dataset was subjected to other data augmentation techniques, more precisely, horizontal flips, vertical flips and random rotations by a multiple of 90 degrees, all of them commonly used in super-resolution since they do not require the use of pixel interpolation. Thus, with this last step the dataset is ready to train the network.



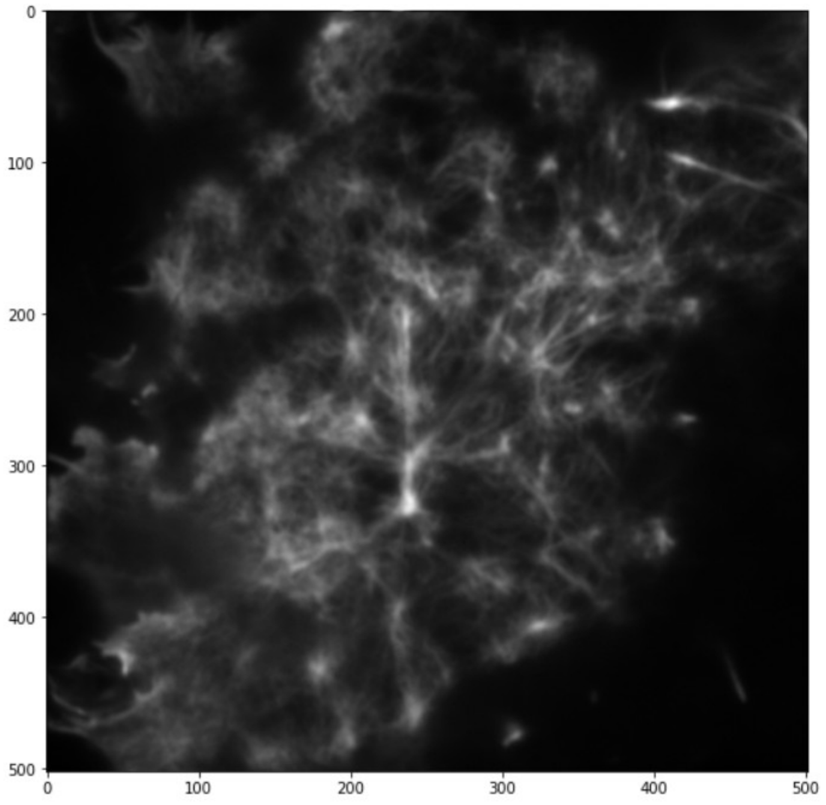
(a) Original image

(b) Vertical flip

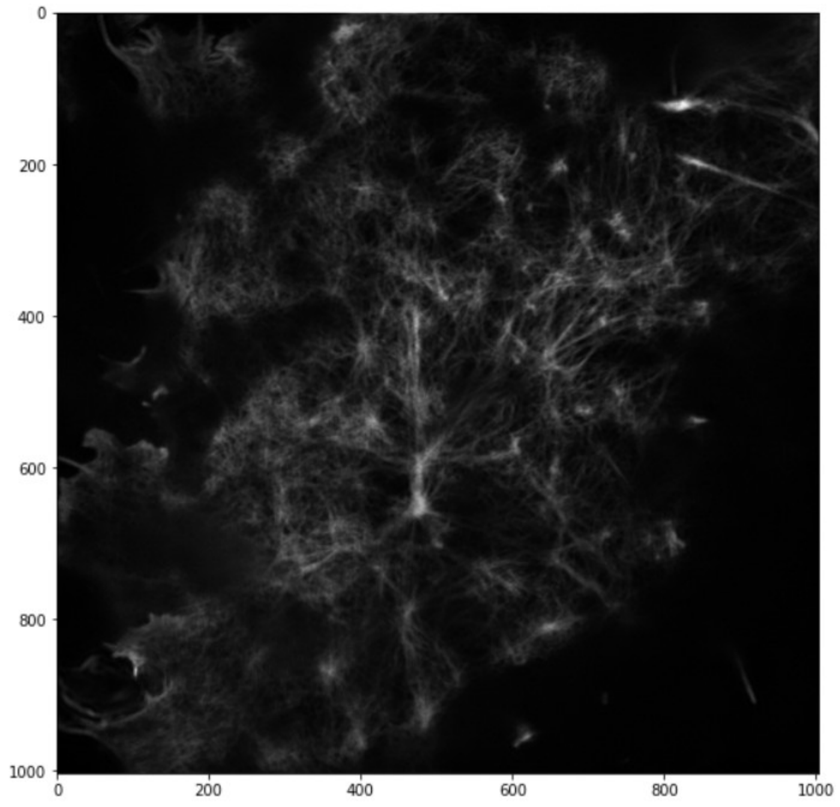
(c) Horizontal flip

(d) Rotation of 90 degrees

Figure 5.3: Examples of the techniques of data augmentation. (a) original image, (b) image transformed with a vertical flip, (c) image transformed with an horizontal flip, (d) image transformed with a rotation of 90 degrees



(a) Training image at low resolution



(b) Training image at high resolution

Figure 5.4: Example of a pair of LR-HR images from the F-actin dataset with the dimension axes.

5.3 Training

The training process was done in a notebook making use of the services provided by Google Colab (see Section 3.3). Some of the parameters varied with the purpose of searching the best hyperparamaters, but several are inherent to the network:

1. **Optimizer**: Adam [26]. It is defined as an extension to Stochastic gradient descent method.
2. **Loss function**: The loss function defined by Chang et al. in [5] (see Equation 4.5)
3. **Reduce on plateau**: Function that schedules the learning rate on a 1 cycle policy as per Leslie Smith's paper [27]
4. **Evaluation metric**: Mean squared error (see Equation 2.9)
5. **Number of residual groups**: four
6. **Number of RCAB**: four

Additionally, the callback *Earlystopping* was used, also from the Keras library [3]. This callback stops the training when a monitored metric has stopped improving, in this case the validation loss. This method checks at the end of every epoch whether the loss is no longer decreasing and, if that's the case, the training finishes. The function has a patience parameter, in the case of these experiments the patience is set to ten, so the training will be stopped after ten epochs with no improvement.

5.4 Search of hyperparameters / Results

Before the implementation of the Zerocost notebook, some experiments were performed with the aim of testing the results of the network and finding the parameters that obtained the best results. This section will present the different experiments and their results.

All of the experiments were done after setting a reproducibility seed, so that the obtained results were as reproducible as possible.

[3]Earlystopping from the Keras library: Earlystopping

5.4.1 DFCAN with the F-actin dataset

The first experiments were conducted on the F-actin dataset, composed by images of F-actin filaments, the most complex biological structures in the dataset provided by Chan et al. [5]. These experiments were done by creating a training set of 1476 patches extracted from the original images, with a validation set of the 10% of the training set. Table 5.1 shows the parameters and results for eight different experiments.

No.	LR	# Eps	BS	PSNR	SSIM	MSSIM	Time
1	0.0003	10	8	26.358	0.759	0.861	30
2	0.0005	10	8	27.392	0.809	0.881	30
3	0.001	10	8	28.522	0.837	0.896	65
4	0.001	5	8	27.447	0.767	0.876	15
5	0.0015	5	8	27.444	0.779	0.878	33
6	**0.001**	**10**	**4**	**29.427**	**0.846**	**0.903**	**30**
7	0.001	10	1	18.076	0.115	0.329	34
8	0.005	10	8	18.076	0.115	0.329	29
9	0.0007	10	1	18.076	0.115	0.329	35

Table 5.1: Experiments of DFCAN network with F-actin dataset and DFCAN loss (Equation 4.5). The column *No.* is used to denote the number of experiment. The column *Network* represents the network used in this experiments. The column *LR* is filled by the learning rates used in each experiment. The column *#Eps* indicates the number of epochs the model has trained in each experiment. *BS* represents the batch size. The next 3 columns indicate the evaluation metrics, PSNR (Section 2.4.3), SSIM (Section 2.4.1) and MSSIM (Section 2.4.2) on the test set. The last column, *Time* indicates the training time for each experiment in minutes. The bold line highlights the experiment with the best results.

As shown in Table 5.1, Experiment 1 obtains decent results, although compared to the rest it is not one of the best. However, in Figure 5.6, it can be seen that the loss and MSE stabilise on the fourth epoch, finally reaching a loss value of 0.0089 for the training and a loss value of 0.0091 for the validation. Both values are almost identical, a sign that the network is not overfitting.

Figure 5.5 shows three sets of LR, ground truth (HR) and prediction images. Although the three might seem the same image to the naked eye, all of them are different samples, because as mentioned in Section 5.1, the dataset contains ten samples for ten escalating levels of excitation light intensity for each different biological structure. The aim of this dataset is to show that the predictions for images with high levels of light intensity are better than the predictions with low levels of light intensity. The sets of images in Figure 5.5 are ordered by the light intensity, from lower to higher intensity. Therefore, as it can be seen, the last set obtains the best result among the three of them.

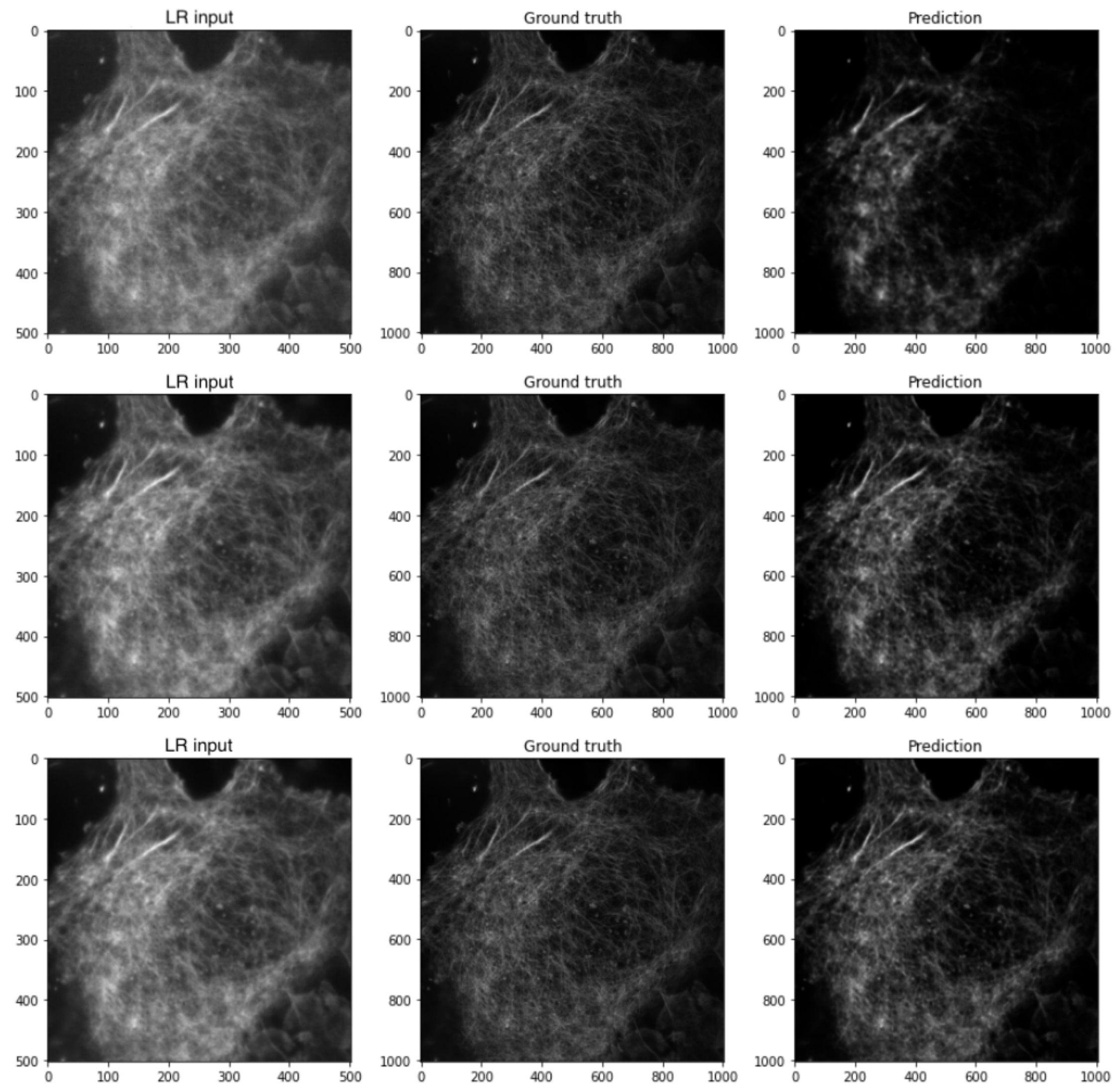


Figure 5.5: Results for Experiment 1. This Figure shows three sets of LR, ground truth (HR) and prediction images, ordered by light intensity. The set with the best results is the last one, which has the highest level of light intensity.

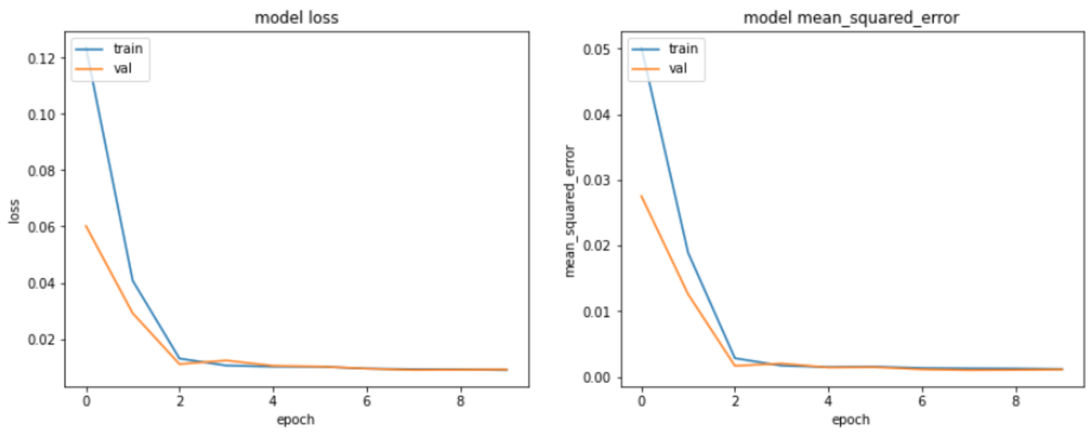


Figure 5.6: Loss and MSE training history for Experiment 1

For Experiment 2, it can be seen that by increasing the learning rate from 0.0003 to 0.0005 the metrics improve, and since the training time remains the same, it can be concluded that the set of hyperparameters of the second experiment is better than the first one. As shown in Figure 5.7, the loss and MSE are stabilised even earlier than in Experiment 1, reaching a loss value of 0.0089 and a MSE value of 0.0011 for the validation. In addition, Figure 5.8 shows better results if compared with the results in Figure 5.5. A comparison is displayed in Figure 5.9, where the improvement is more noticeable.

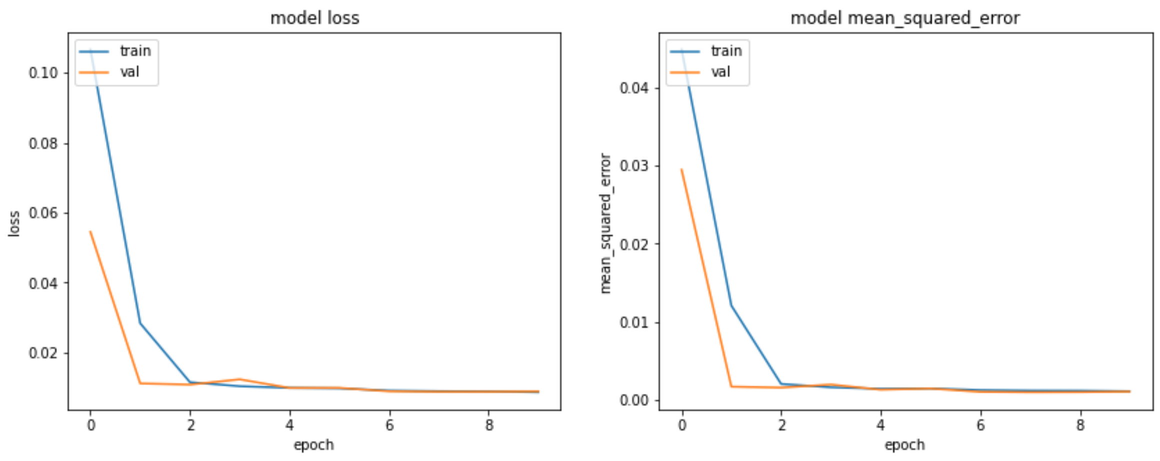


Figure 5.7: Loss and MSE training history for Experiment 2

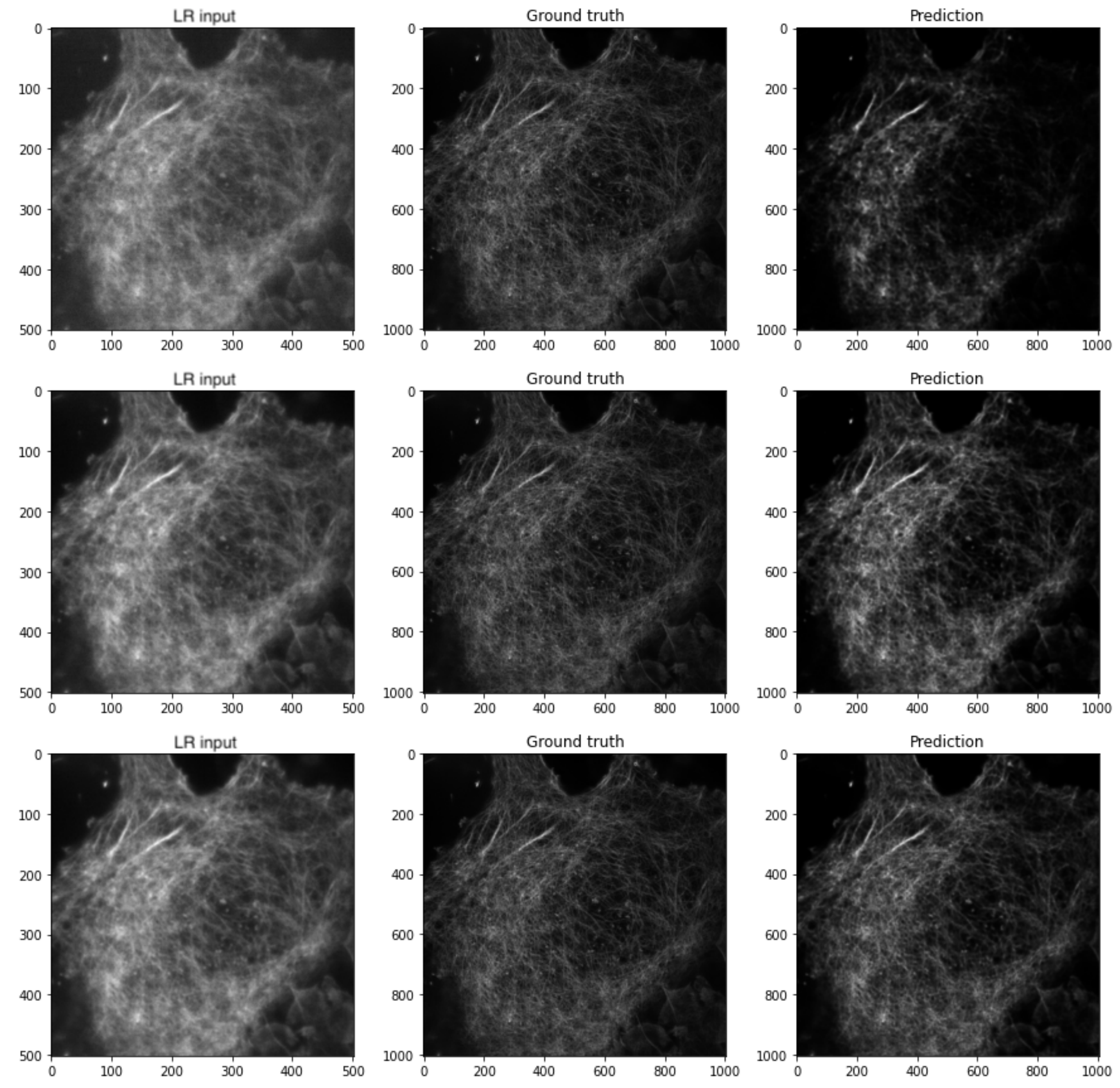
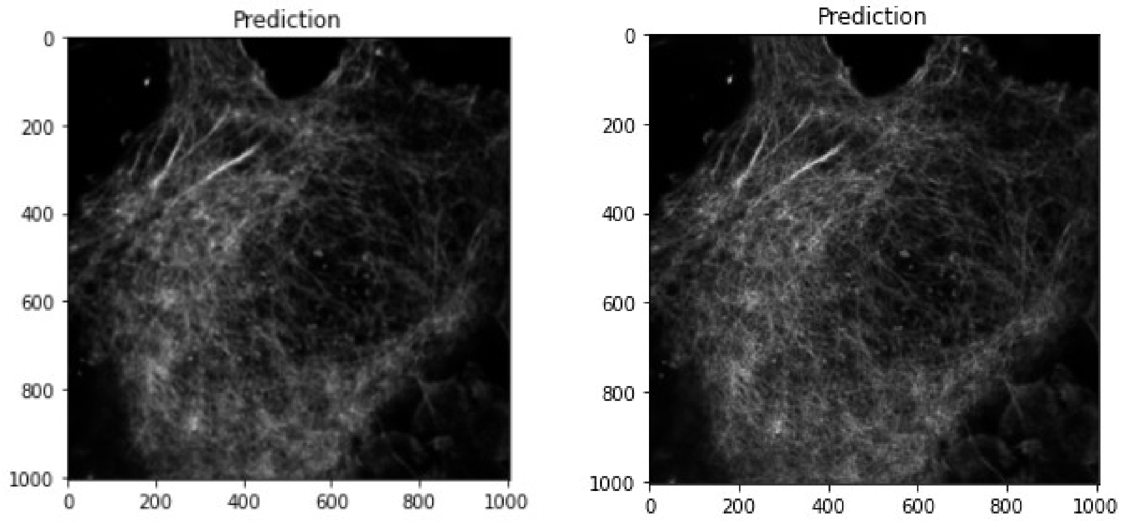


Figure 5.8: Results for Experiment 2



(a) Result from Experiment 1

(b) Result from Experiment 2

Figure 5.9: Comparison of the third image from experiments number 1 and 2. As it can be seen the image in (b) shows better results than (a).

Continuing with Experiment 3, since an increment of the learning rate resulted in an improvement of the prediction, the rest of the parameters were left unchanged and the learning rate was set to 0.001. As shown in Table 5.1, the results are the best of the three and the second best of all of the experiments, but the training time is twice as much as in the previous experiments. As for the training loss history and MSE history, they are almost identical to the ones shown in Figure 5.7 since the training finishes with a validation loss of 0.0087 and a MSE of 0.0011.

For Experiment 4, the idea was to check if training with less epochs, five in this case, made a big difference, and as the results show, the metrics are worse than in the third experiment and similar to the first two, but since the training time is 15 minutes, it is worth taking this set of hyperparameters into consideration if speed is needed.

Experiment 5 was made with the same set of hyperparameters as the fourth one, except for the learning rate, which was set to 0.0015. The results however do not show an improvement.

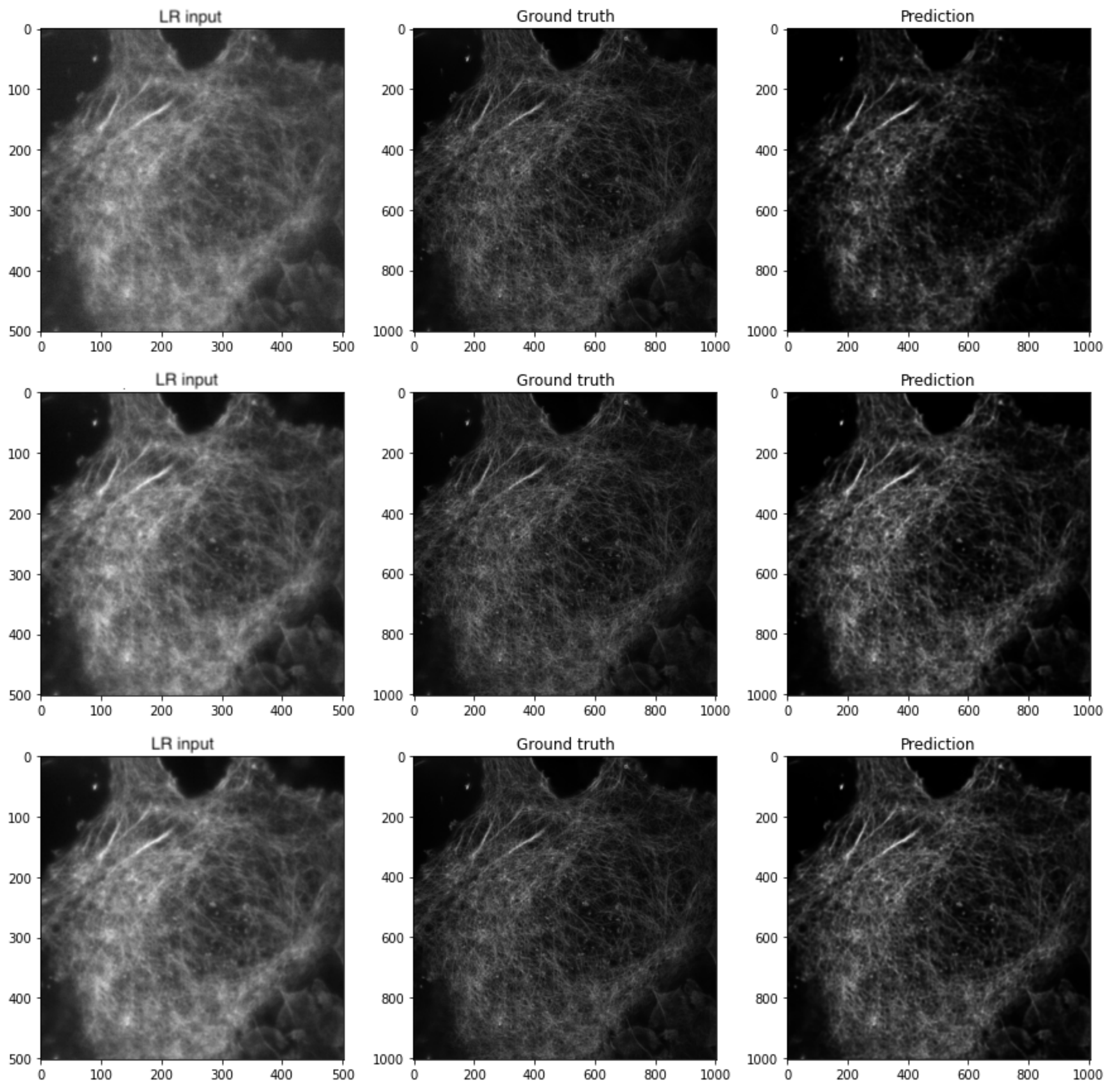


Figure 5.10: Results for Experiment 3

Regarding Experiment 6, it can be seen that the metrics surpass those of the rest of the experiments. Comparing it to the third experiment, the results are better in every sense since aside from improving the metrics, the training time is half of the former's training time.

As depicted in the training history shown in Figure 5.12, the loss and MSE descend drastically, reaching a loss value of 0.0084 and a MSE value of 0.000097 for the validation, represented by the orange line in the figure. In addition, the train and validation are almost identical from the fourth epoch on, so the model is not overfitting.

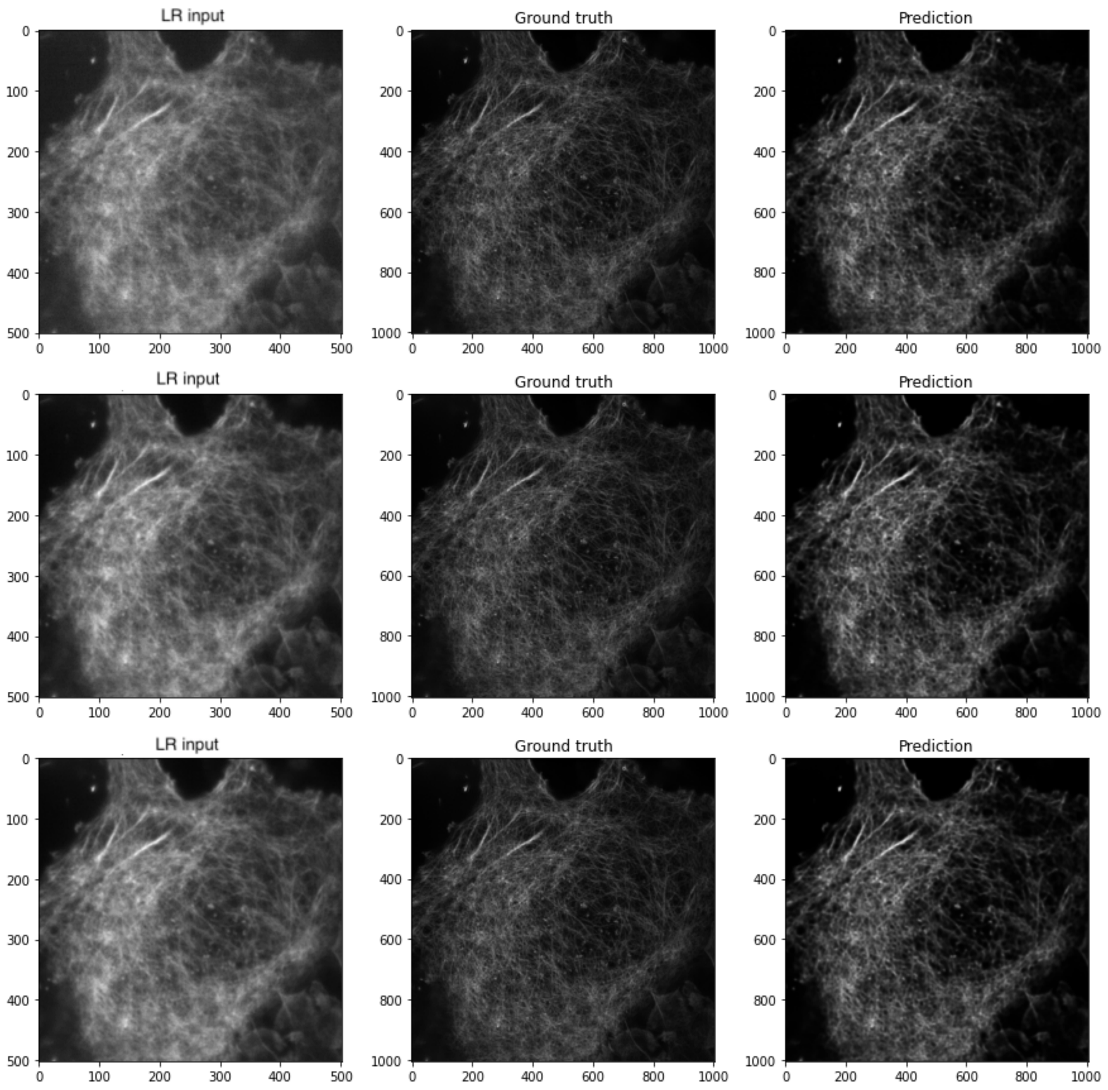


Figure 5.11: Results for Experiment 6

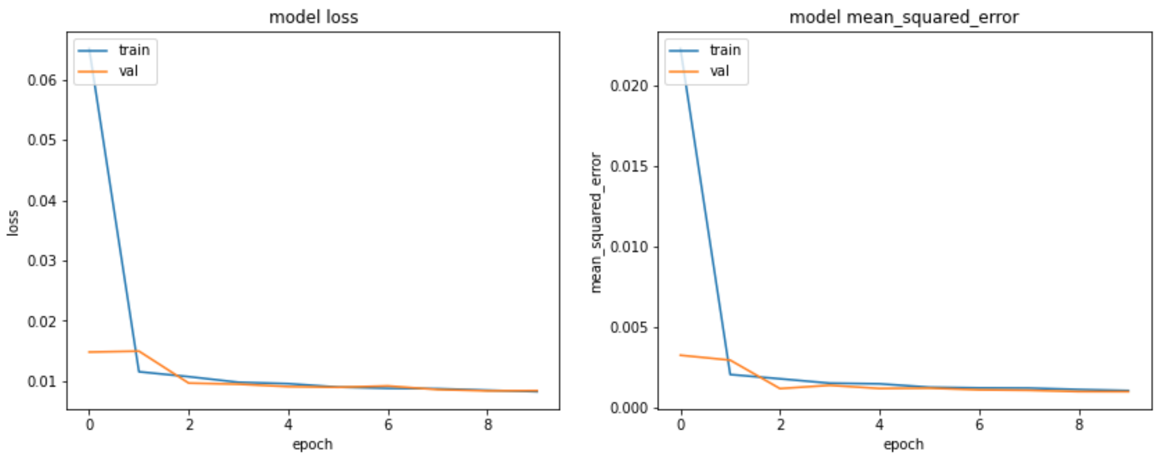


Figure 5.12: Loss and MSE training history for Experiment 6

However, there is a detail that needs to be taken into account: the set of hyperparameters for this sixth experiment is on the limit. In other words, an increase in the learning rate or a decrease in the batch size results in a high sudden increase in the loss and MSE of the model's training, which leads to suboptimal predictions. Experiment 7 reflects this issue, where the only difference regarding the previous experiment is that the batch size is 1 instead of 4. In Figure 5.13, it can be seen that the training for this experiment starts in relatively good loss and MSE values but they immediately scale upwards and they stay the same throughout all the training time.

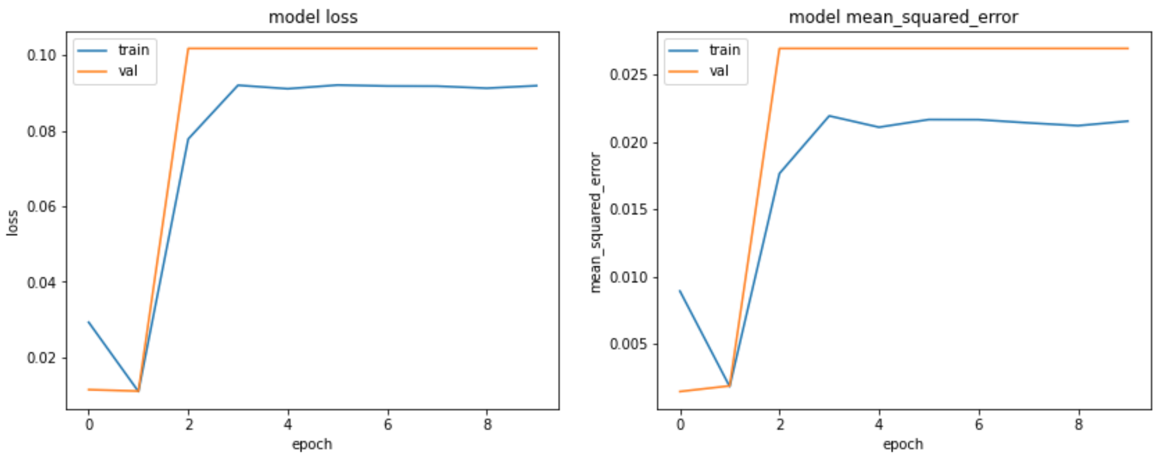


Figure 5.13: Loss and MSE training history for Experiment 7

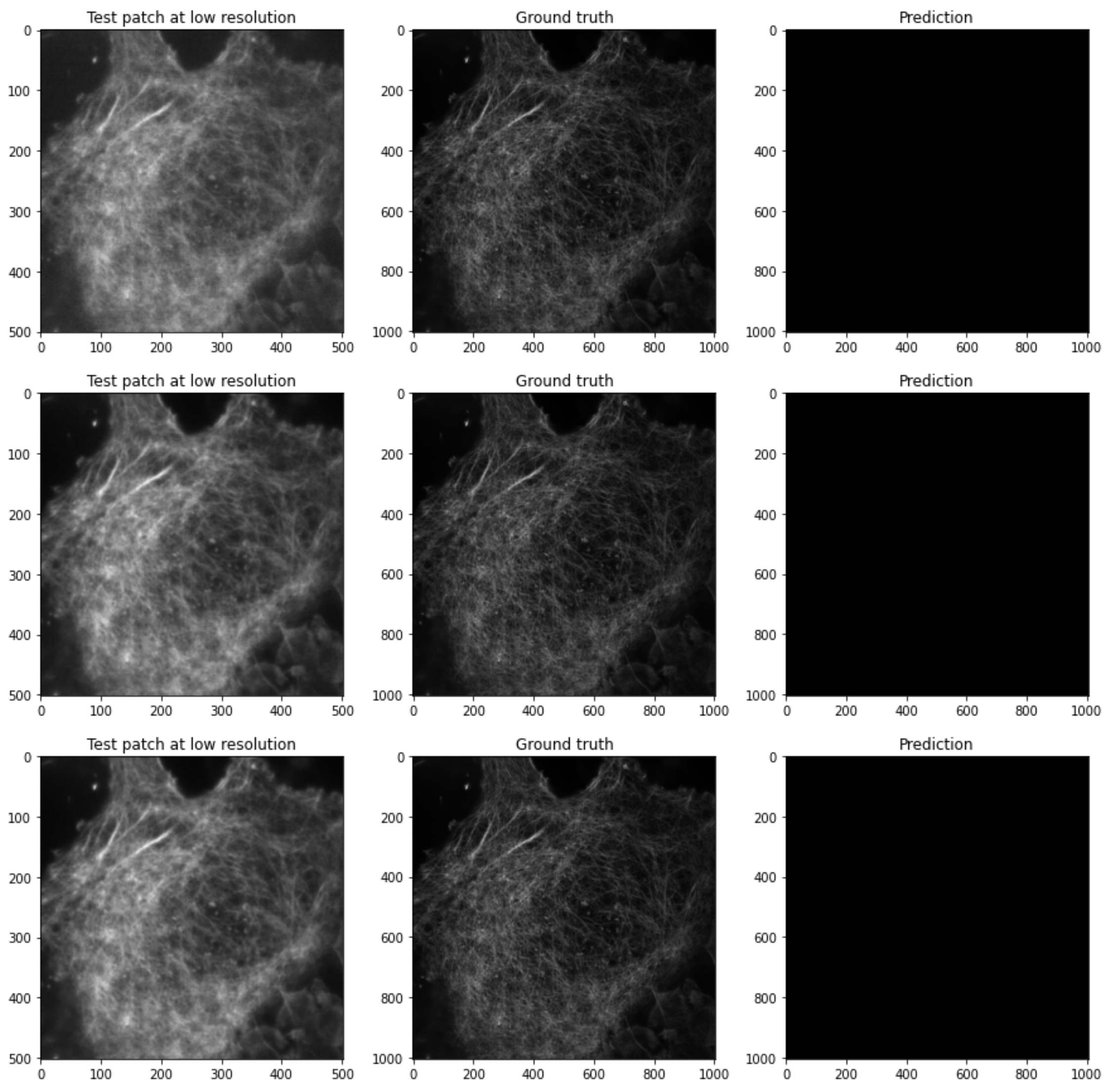


Figure 5.14: Results for Experiment 7

The consequences of the model not learning in the training is that the resulting predictions are completely black images, as it can be seen in the third column of Figure 5.14. This happens because a very high number of pixels are black, so these poor predictions tend to predict black pixels. The results, as shown in Table 5.1 greatly differ from all the previous experiment metrics.

Experiments 8 and 9 were done to reinforce this statement. Regarding the former, even with a batch size of eight, which was a solid value for the first five experiments, it produces poor results if combined with a learning rate of 0.005. The same happens with the latter, in which the learning rate was set to 0.0007 which is not as high as the 0.0015 learning rate of Experiment 5. When combined with a batch size of one, the results are the same as in the previous experiment. As for the predicted images, all of them are completely black images for both experiments.

Finally, to complete this set of experiments, instead of using the loss function created specifically for the DFCAN (see Equation 4.5), some tests were made using the Mean Absolute Error (MAE) as the loss function. As shown in Table 5.2, the metrics from the test set tend to be lower than in Table 5.1, so these experiments verify that the DFCAN loss indeed improves the performance of the network.

No.	LR	# Eps	BS	PSNR	SSIM	MSSIM	Time
1	0.0003	10	8	25.341	0.762	0.864	63
2	0.0005	10	8	24.986	0.755	0.853	29
3	0.001	10	8	26.948	0.811	0.874	28
4	0.001	5	8	25.226	0.759	0.854	15
5	0.0015	5	8	26.408	0.785	0.873	14
6	**0.001**	**10**	**4**	**29.359**	**0.864**	**0.891**	**76**

Table 5.2: Experiments of DFCAN network with F-actin dataset and MAE loss. The column *No.* is used to denote the number of experiment. The column *Network* represents the network used in this experiments. The column *LR* is filled by the learning rates used in each experiment. The column *#Eps* indicates the number of epochs the model has trained in each experiment. *BS* represents the batch size. The next 3 columns indicate the evaluation metrics, PSNR (Section 2.4.3), SSIM (Section 2.4.1) and MSSIM (Section 2.4.2) on the test set. The last column, *Time* indicates the training time for each experiment in minutes. The bold line highlights the experiment with the best results.

5.4.2 RCAN with the F-actin dataset

The second set of experiments was conducted with the RCAN network (see Section 3.2.3) and the F-actin dataset with the aim of comparing the performance of both networks on the same dataset. These experiments were done by creating a training set of 1476 patches extracted from the original images, with a validation set of the 10% of the training set. Table 5.3 reflects all of the experiments with their respective training parameters and prediction metrics. As it can be seen, in general, the RCAN outperforms the DFCAN, since all of the metrics from the test set show higher values than the DFCAN results. For these experiments, the loss function used for the training was the MAE.

No.	LR	#Eps	BS	PSNR	SSIM	MSSIM	Time
1	0.0003	10	8	30.775	0.893	0.916	25
2	0.0005	10	8	30.834	0.895	0.917	26
3	0.001	10	8	30.916	0.898	0.919	25
4	0.0005	5	8	30.763	0.890	0.915	7
5	0.0005	5	4	30.831	0.894	0.916	13
6	**0.0005**	**5**	**1**	**31.006**	**0.900**	**0.921**	**43**

Table 5.3: Experiments of RCAN network with F-actin dataset and MAE loss. The column *No.* is used to denote the number of experiment. The column *Network* represents the network used in this experiments. The column *LR* is filled by the learning rates used in each experiment. The column *#Eps* indicates the number of epochs the model has trained in each experiment. *BS* represents the batch size. The next 3 columns indicate the evaluation metrics, PSNR (Section 2.4.3), SSIM (Section 2.4.1) and MSSIM (Section 2.4.2) on the test set. The last column, *Time* indicates the training time for each experiment in minutes. The bold line highlights the experiment with the best results.

Starting with Experiment 1, as the hyperparameters are the same as for the first experiment of the DFCAN (see Table 5.1), they can be directly compared. The results of the RCAN are higher than the DFCAN, but the training time almost the same. Figure 5.15 shows the training history of the model, where it starts with a validation loss of 0.0438 and a validation mean squared error of 0.0048 and it finishes with a validation loss of 0.0254 and a validation mean squared error of 0.0014. Although the training values and validation values start with a big gap, they quickly stabilise, and in the tenth epoch the values are practically the same.

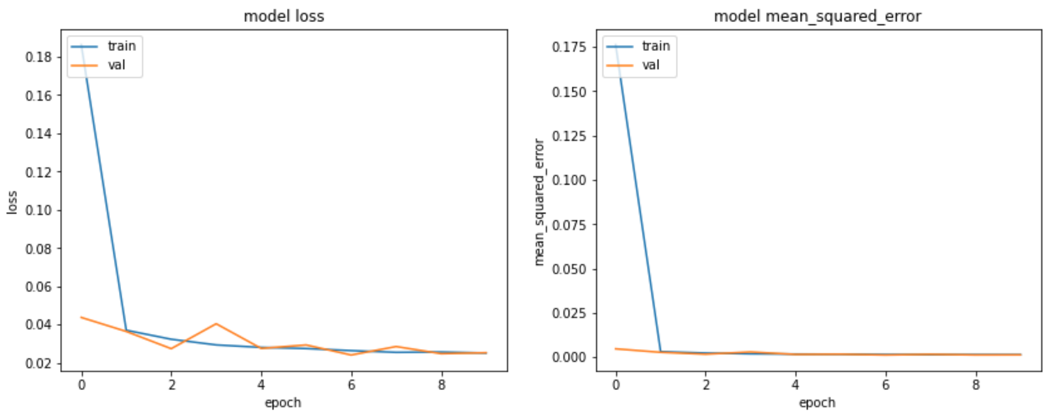


Figure 5.15: Loss and MSE training history for Experiment 1

As for the predicted images, the results are remarkable, and as previously explained, the first LR-HR-Prediction set has a more contrasting prediction than the next two sets.

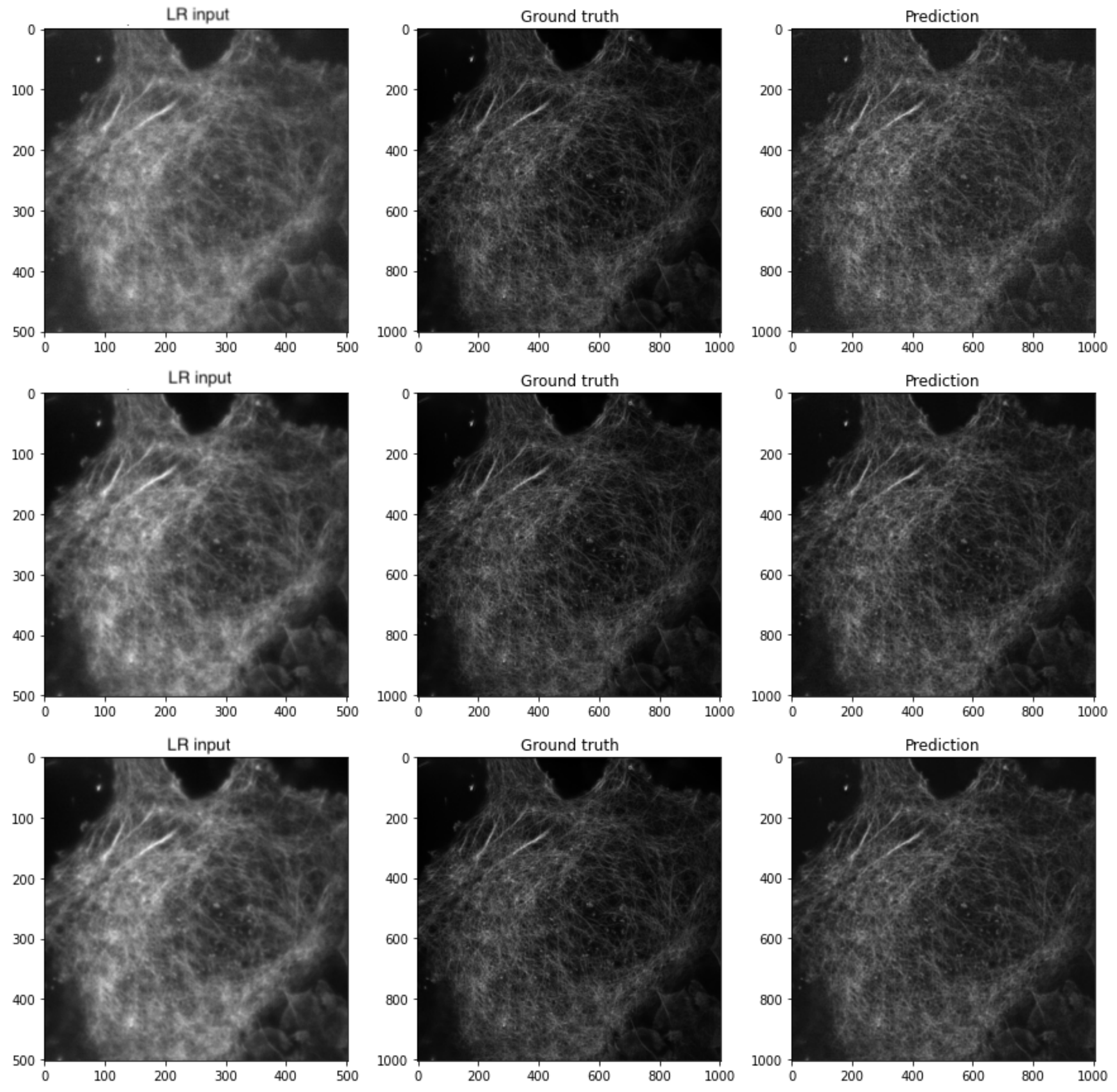


Figure 5.16: Results for Experiment 1

Following with the next two experiments, as Table 5.3 shows, they all obtain very similar results to the first one, with an identical training time. Experiment 4, however, obtains almost the same results as the first experiment but with the advantage that the training time is only seven minutes comparing with the 25 minutes of the first one. This was done by reducing the training epochs from ten to five. The same happens with Experiment 5 which has almost the same results as the second one but with a training time of 13 minutes instead of 26. Finally, the sixth experiment obtains the best results as it finishes its training with a validation loss of 0.0238 and a validation mean squared error of 0.0012. Figure 5.17 shows the training history of this last experiment.

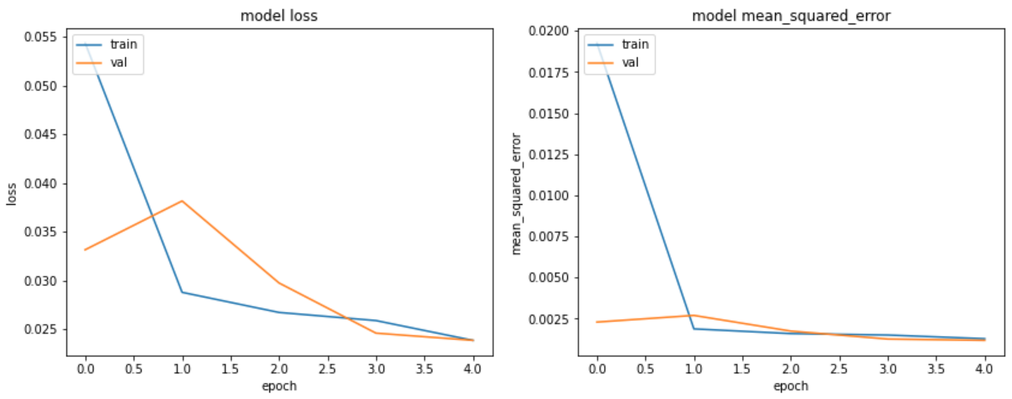


Figure 5.17: Loss and MSE training history for Experiment 6

Lastly, to complete this set of experiments, all of the experiments in Table 5.3 but using the DFCAN loss as the loss function, obtaining the results displayed in Table 5.4. The results do not differ greatly from the RCAN trained with MAE loss but the training time is doubled for all of the experiments except for the sixth one, which in fact, is reduced to the half.

No.	LR	# Eps	BS	PSNR	SSIM	MSSIM	Time
1	0.0003	10	8	30.669	0.889	0.918	64
2	0.0005	10	8	30.769	0.892	0.919	67
3	0.001	10	8	30.910	0.897	0.921	69
4	0.0005	5	8	30.605	0.886	0.918	35
5	0.0005	5	4	30.725	0.890	0.918	37
6	**0.0005**	**5**	**1**	**30.939**	**0.898**	**0.922**	**19**

Table 5.4: Experiments of RCAN network with F-actin dataset DFCAN loss (Equation 4.5). The column *No.* is used to denote the number of experiment. The column *Network* represents the network used in this experiments. The column *LR* is filled by the learning rates used in each experiment. The column *#Eps* indicates the number of epochs the model has trained in each experiment. *BS* represents the batch size. The next 3 columns indicate the evaluation metrics, PSNR (Section 2.4.3), SSIM (Section 2.4.1) and MSSIM (Section 2.4.2) on the test set. The last column, *Time* indicates the training time for each experiment in minutes. The bold line highlights the experiment with the best results.

5.4.3 DFCAN with the electron microscopy image dataset

Finally, the last set of experiments was done using the DFCAN with another dataset composed by Electron Microscopy (EM) images. This dataset was produced by Lichtman Lab at Harvard University (Daniel R. Berger, Richard Schalek, Narayanan "Bobby" Kasthuri, Juan-Carlos Tapia, Kenneth Hayworth, Jeff W. Lichtman). Their corresponding biological findings were published in [28]. The training and test data sets are both 3D stacks of 100 sections from a serial section Scanning Electron Microscopy (ssSEM) data set of mouse cortex. These experiments were done by using creating a training set of 1600 patches of 256 × 256 extracted from the original images, using a validation set of the 10% of the training set. The applied data augmentation techniques were horizontal flips, vertical flips and random rotations by a multiple of 90 degrees, presented in Figure 5.3.

This dataset does not include the LR image for each high resolution one, so the LR images were created synthetically by adding noise and downsampling the images (see [29]). Figure shows an example of a LR-HR patch images.

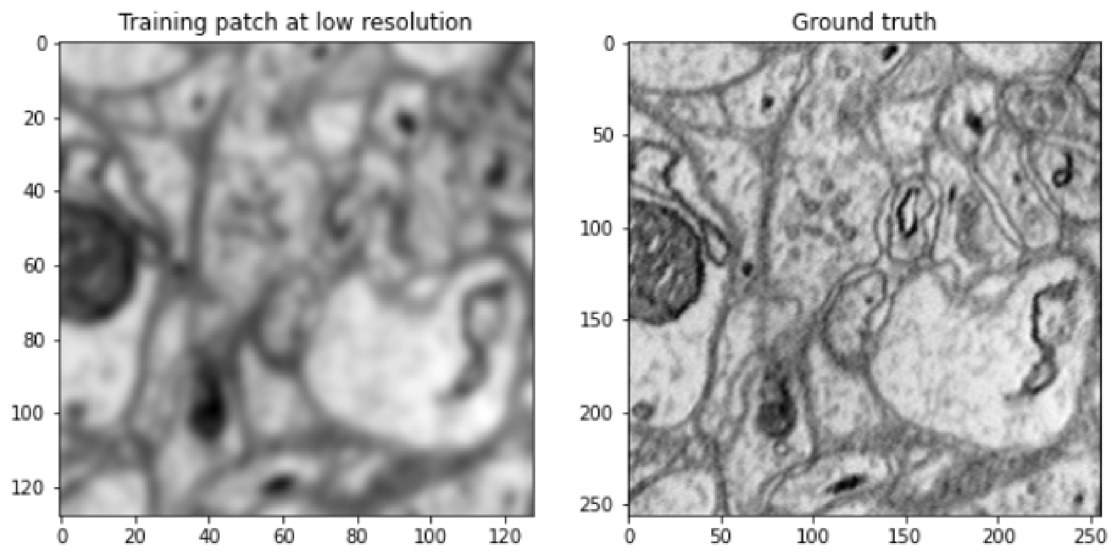


Figure 5.18: Example of a LR-HR patch pair from the EM dataset.

Regarding the results of the training, Table 5.5 shows the experiments done for this dataset. As it can be seen, the values of the metrics tend to be higher comparing them with the results in Table 5.1, and the training times remain very similar to the times of Table 5.1.

No.	LR	# Eps	BS	PSNR	SSIM	MSSIM	Time
1	0.0003	10	8	27.751	0.912	0.980	31
2	0.0005	10	8	27.711	0.915	0.985	32
3	**0.001**	**10**	**8**	**31.006**	**0.94575**	**0.989**	**31**
4	0.001	5	8	29.338	0.929	0.985	16
5	0.0015	5	8	30.768	0.949	0.989	16

Table 5.5: Experiments of DFCAN network with EM dataset and DFCAN loss (Equation 4.5). The column *No.* is used to denote the number of experiment. The column *Network* represents the network used in this experiments. The column *LR* is filled by the learning rates used in each experiment. The column *#Eps* indicates the number of epochs the model has trained in each experiment. *BS* represents the batch size. The next 3 columns indicate the evaluation metrics, PSNR (Section 2.4.3), SSIM (Section 2.4.1) and MSSIM (Section 2.4.2) on the test set. The last column, *Time* indicates the training time for each experiment in minutes. The bold line highlights the experiment with the best results.

Starting with Experiment 1, the training history, as it can be seen in Figure 5.19, shows a big difference between the training and validation metrics in the beginning but it rapidly lowers and stabilises reaching a validation loss of 0.0023 and a validation mean squared error of 0.00091 in the last epoch. On the other hand, Figure 5.20 shows the results for three input LR images. The predictions are visually good but if they are closely inspected it can be seen that they do not have the same quality as the ground truth. Figure 5.21 shows an example of a ground truth patch and its prediction, where the prediction is slightly more blurred. Experiment 2 shows results very similar to Experiment 1.

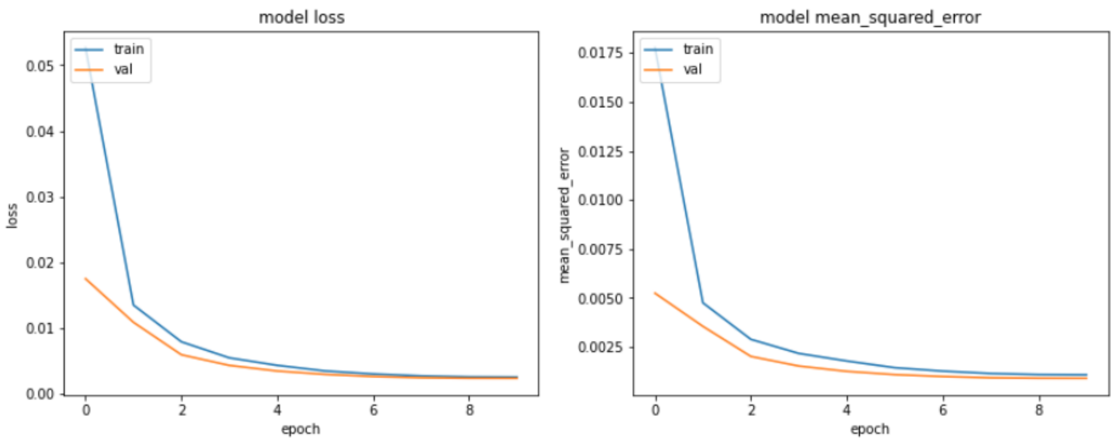


Figure 5.19: Loss and MSE training history for Experiment 1

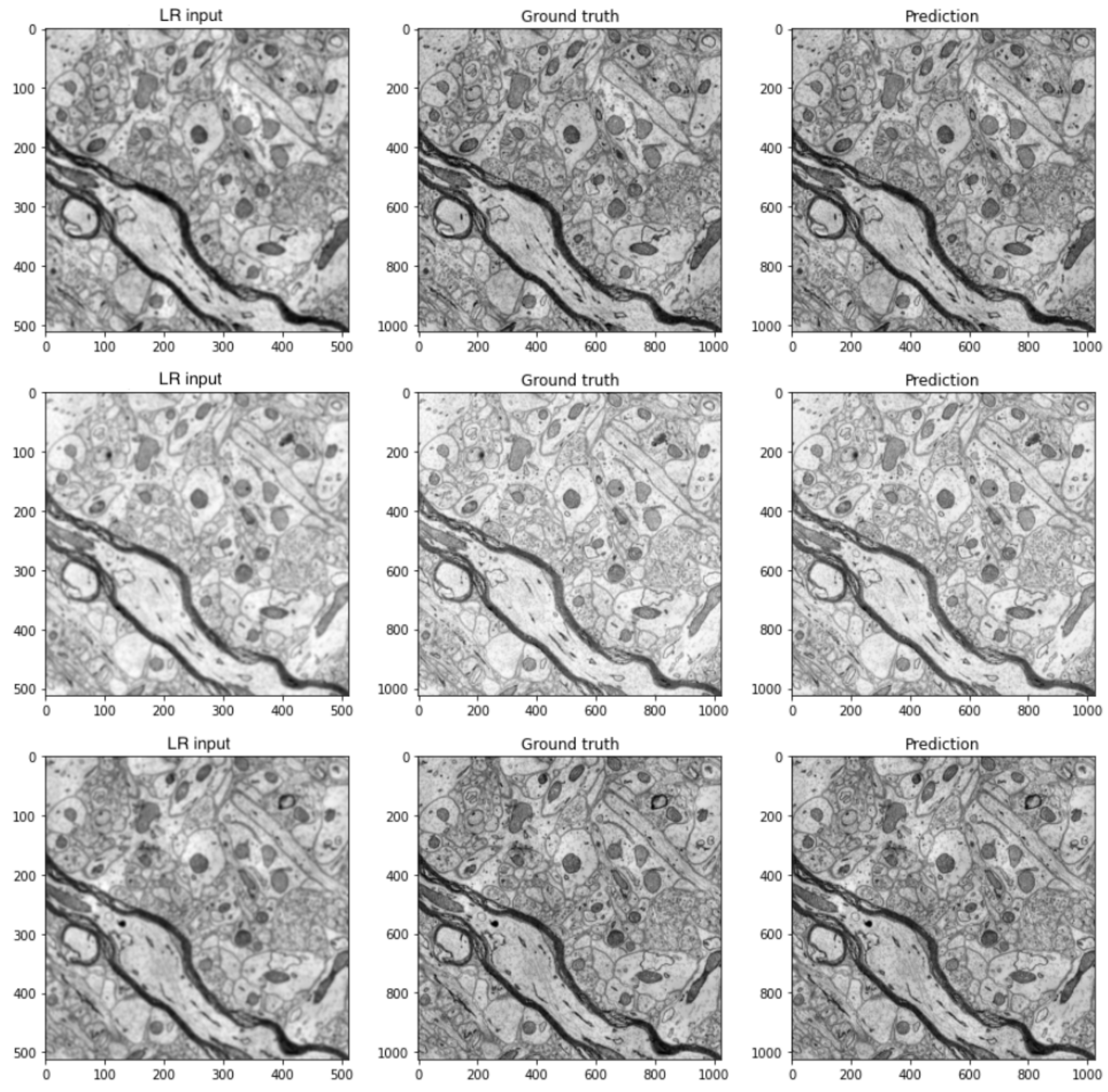


Figure 5.20: Results for Experiment 1

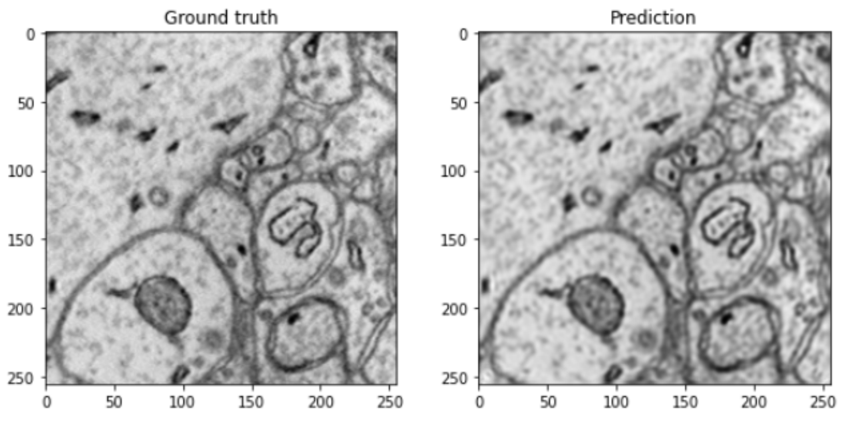


Figure 5.21: Ground truth patch (left) and its prediction (right), from Experiment 1

The third experiment however shows higher values of the metrics than the previous two. If compared to the experiments from the F-actin dataset in Table 5.1, the Experiment 3 from both tables have the same set of hyperparameters, and in both cases the results are one of the best from their respective Tables. Figures 5.22 and 5.23 show the image results and training history respectively.

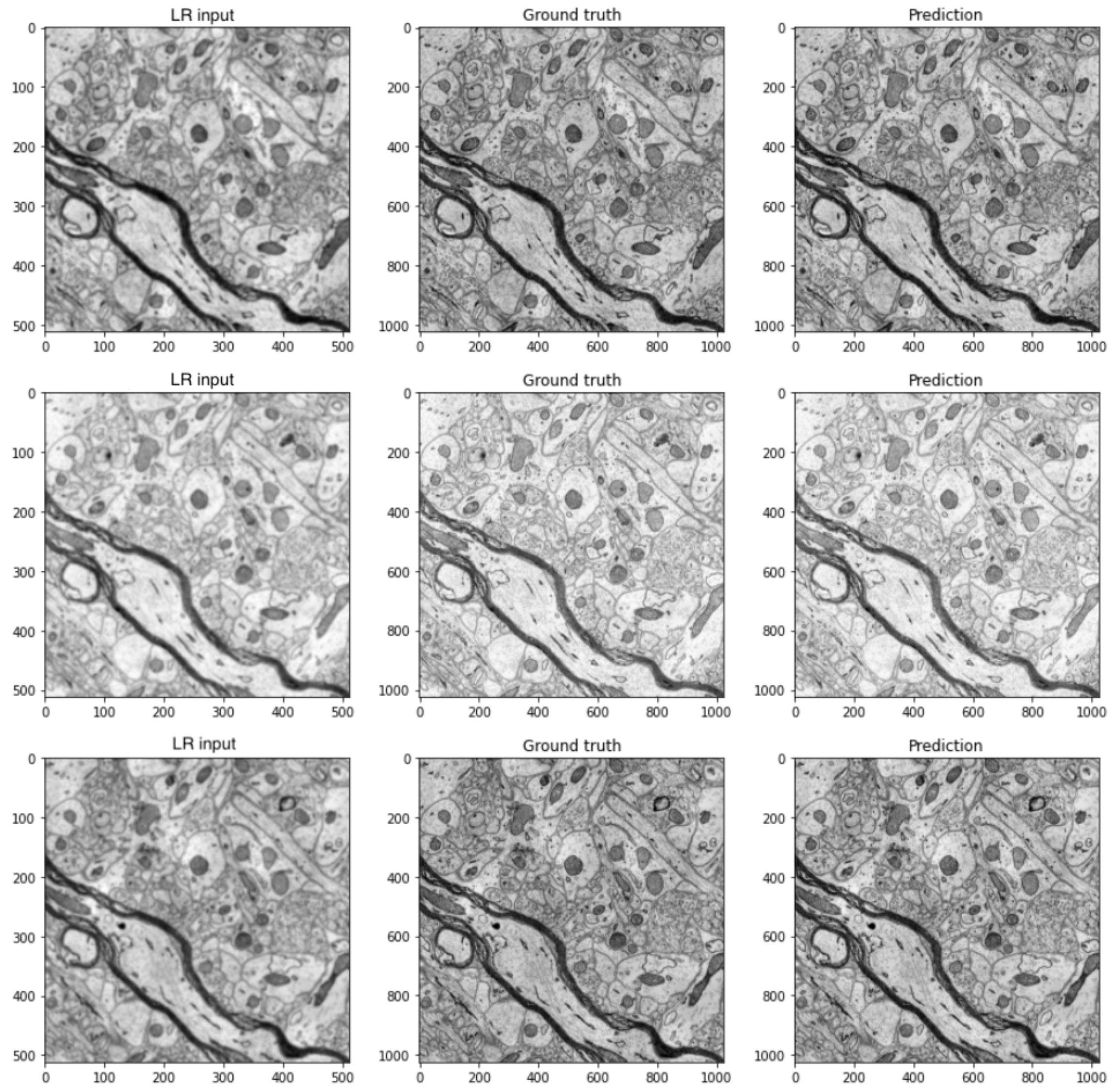


Figure 5.22: Results for Experiment 3

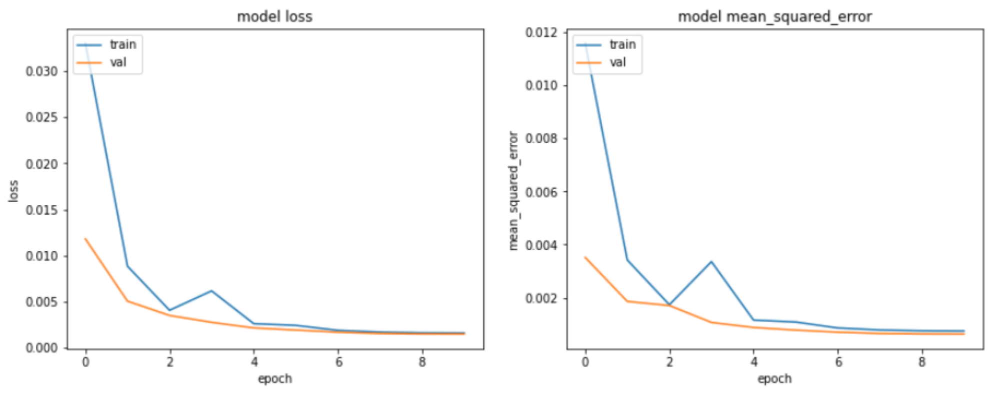


Figure 5.23: Loss and MSE training history for Experiment 3

As for the experiments in Section 5.4.1, an increase in the learning rate shows an improvement in the results, but learning history is not as stable as for the experiments with a lower learning rate. The training history shows this in Figure 5.23, where in the third epoch it presents a peak in the loss and MSE. However, in this case, the loss and MSE quickly stabilise again.

Experiment 4 is the same as Experiment 3 but with five training epochs instead of ten. The results of the metrics, as expected, are a bit lower but since the training time is 16 minutes instead of 31, it can be taken into consideration if the training time is of great importance. Lastly, the fifth experiment, with a learning rate of 0.0015 and five training epochs, shows the best SSIM and MSSIM and the second best PSNR of all the experiments for the EM dataset. The training time is 16 minutes so it can be considered that this set of hyperparameters is the best in terms of results and training time. Figures 5.24 and 5.25 show the training history and the results respectively.

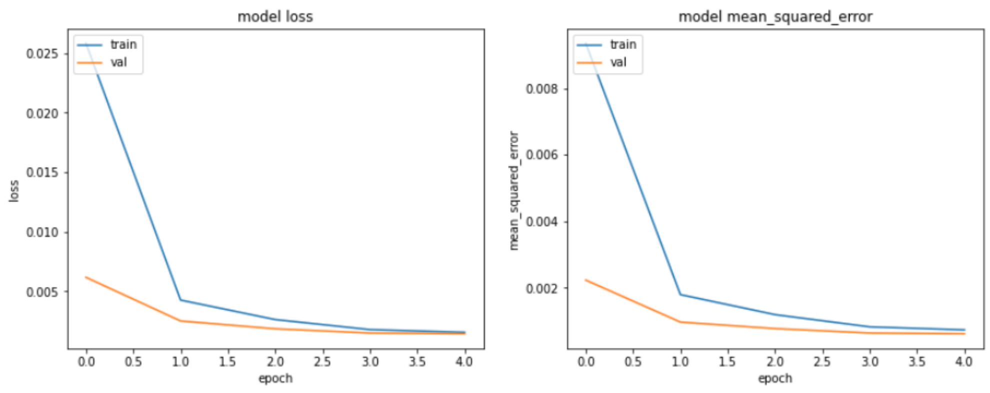


Figure 5.24: Loss and MSE training history for Experiment 5

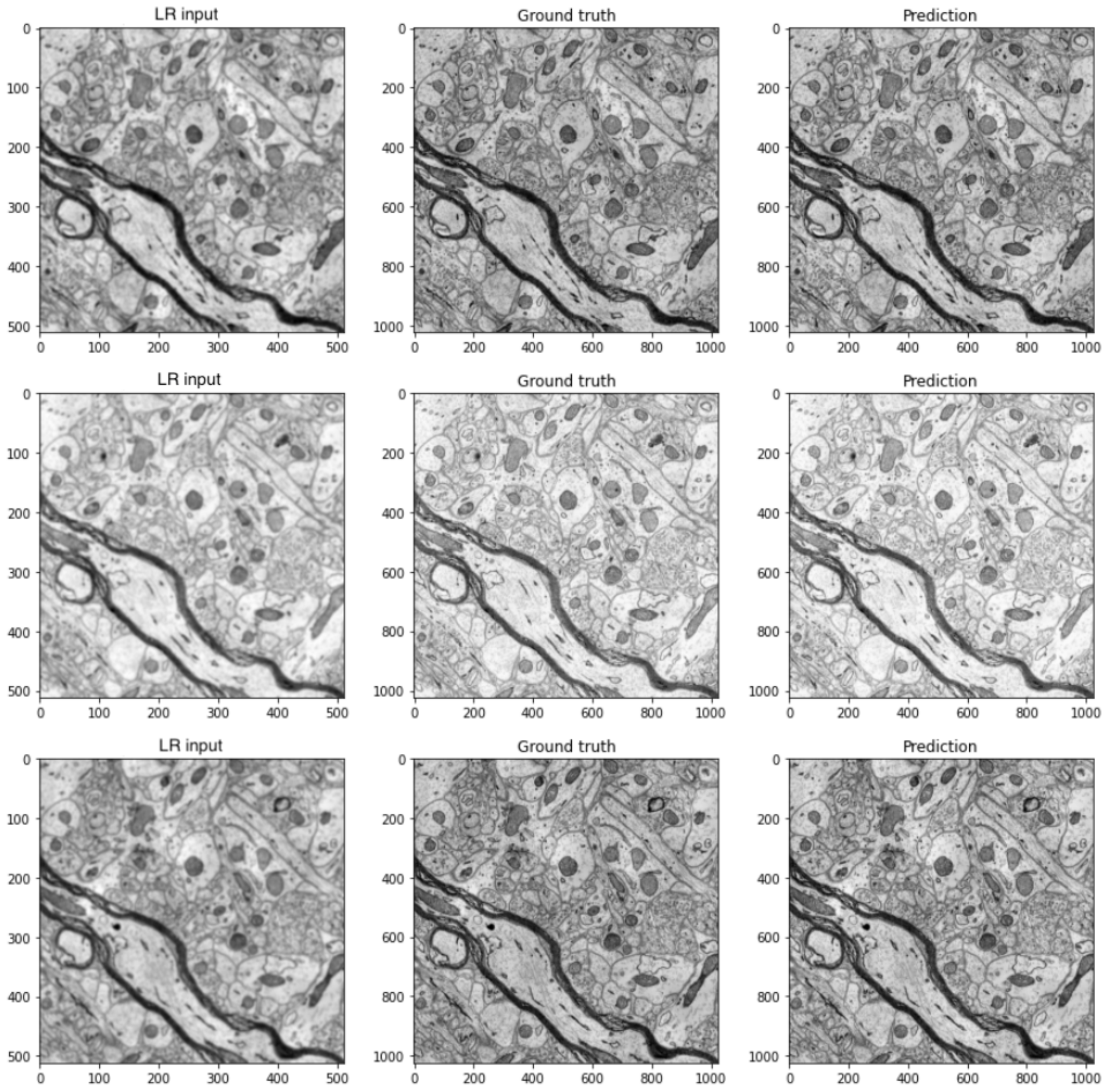


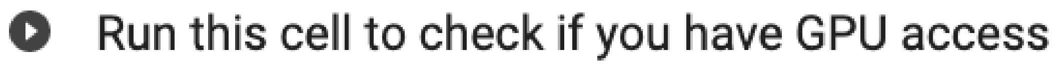
Figure 5.25: Results for Experiment 5

CHAPTER 6

Zerocost notebook

Finally, the last task for this thesis was to create an easy-to-use notebook so that anyone could experiment with the network without the need of having coding skills. As previously stated (see Section 3.3), in order to make this network usable for everyone, a Jupyter notebook was developed from the ZeroCostDL4Mic team.

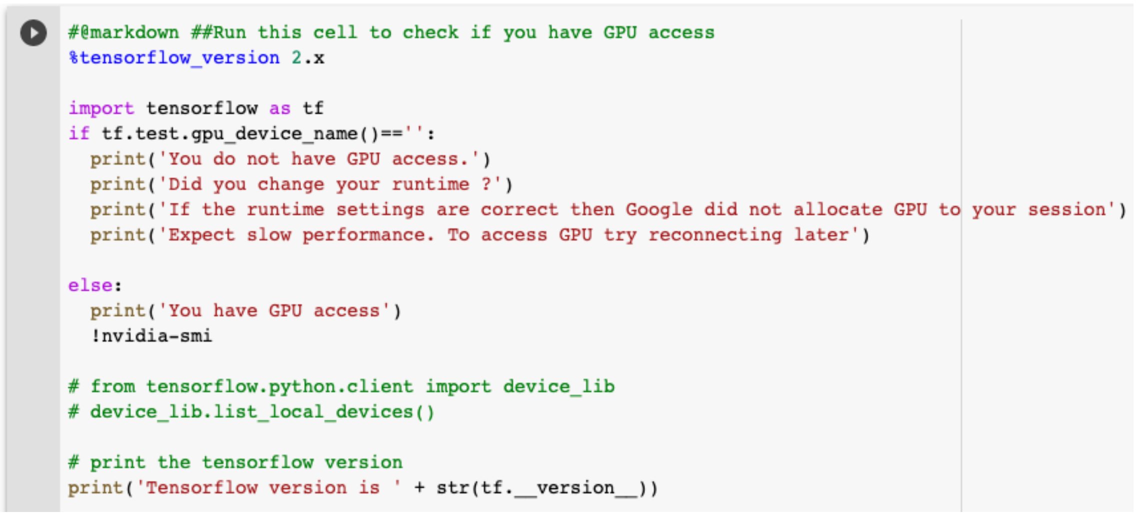
To start with, Jupyter notebooks are composed by cells. There are two types of cells, text cells and executable cells. The former ones are purely informative cells, but the later ones can be executed by clicking a play button positioned on the top left side of the cell.



Run this cell to check if you have GPU access

Figure 6.1: Example of an executable cell in the notebook. As it can be seen, the cell has a play button on the top left side.

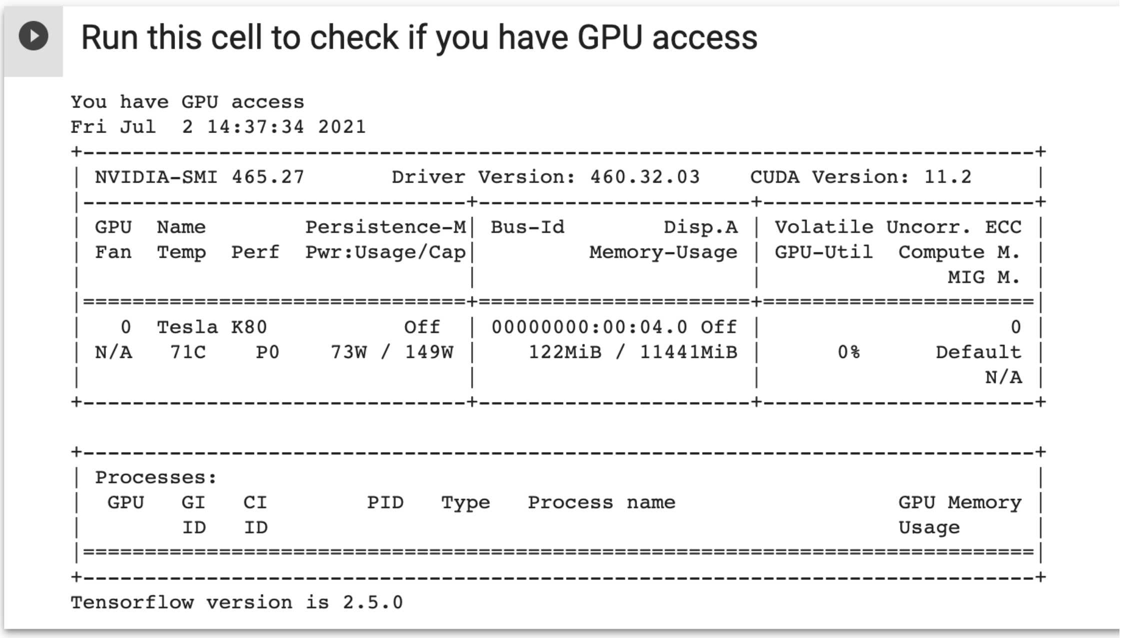
Executable cells are composed by the code that has to be executed and the code is usually not hidden, but in this case, as the code is not relevant for the users, it is concealed so that it does not confuse them. For example, for the cell showed in Figure 6.1, which is the first executable cell of this notebook, the code that is behind the text is the one showed in Figure 6.2. However, the code could be seen by double clicking on the cell.



```
#@markdown ##Run this cell to check if you have GPU access
%tensorflow_version 2.x

import tensorflow as tf
if tf.test.gpu_device_name()=='':
  print('You do not have GPU access.')
  print('Did you change your runtime ?')
  print('If the runtime settings are correct then Google did not allocate GPU to your session')
  print('Expect slow performance. To access GPU try reconnecting later')

else:
  print('You have GPU access')
  !nvidia-smi

# from tensorflow.python.client import device_lib
# device_lib.list_local_devices()

# print the tensorflow version
print('Tensorflow version is ' + str(tf.__version__))
```

Figure 6.2: Code of the cell displayed in Figure 6.1. Usually, the code is not hidden and it is showed as it is in this figure, but in this case the code is out of view with the aim of being more user-friendly. More specifically, this cell's functionality is to check if the user has access to a GPU for that session.

Once executed, the cells usually have an output, although not always. For example, for the cell in Figure 6.1, the output after executing it is the one showed in Figure 6.3, which displays if the user has GPU access in that notebook session and the technical data of the granted GPU.

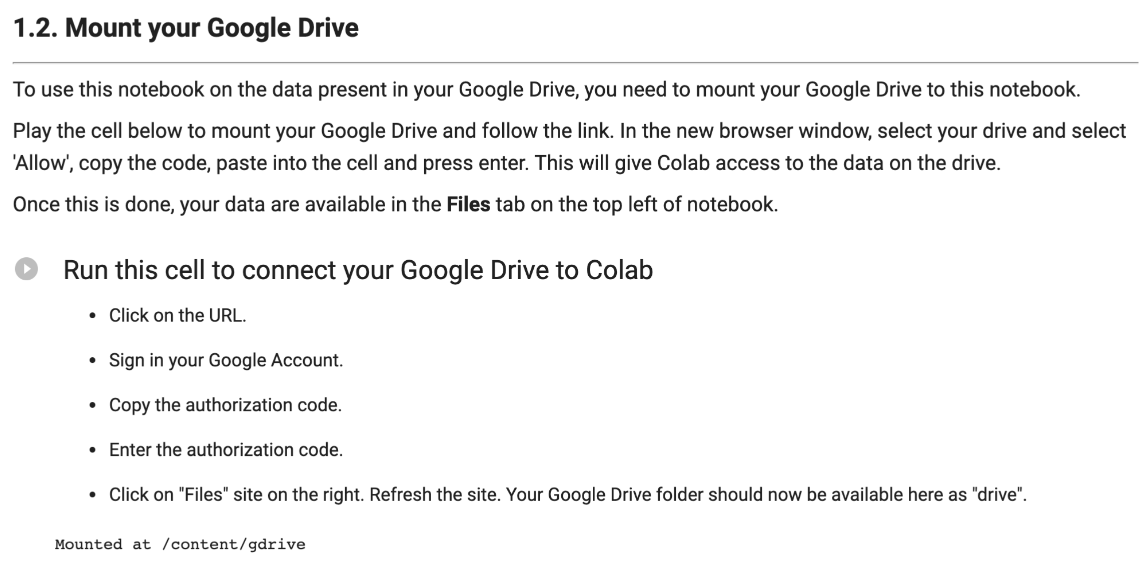


Run this cell to check if you have GPU access

```
You have GPU access
Fri Jul  2 14:37:34 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       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 K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   71C    P0    73W / 149W |    122MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
Tensorflow version is 2.5.0
```

Figure 6.3: Output of the cell showed in Figure 6.1. It shows if the user has GPU access in that notebook session and the specifics of the GPU.

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 has the dataset he/she intends to use for that session it can be easily accessed.



1.2. Mount your Google Drive

To use this notebook on the data present in your Google Drive, you need to mount your Google Drive to this notebook.

Play the cell below to mount your Google Drive and follow the link. In the new browser window, select your drive and select 'Allow', copy the code, paste into the cell and press enter. This will give Colab access to the data on the drive.

Once this is done, your data are available in the **Files** tab on the top left of notebook.

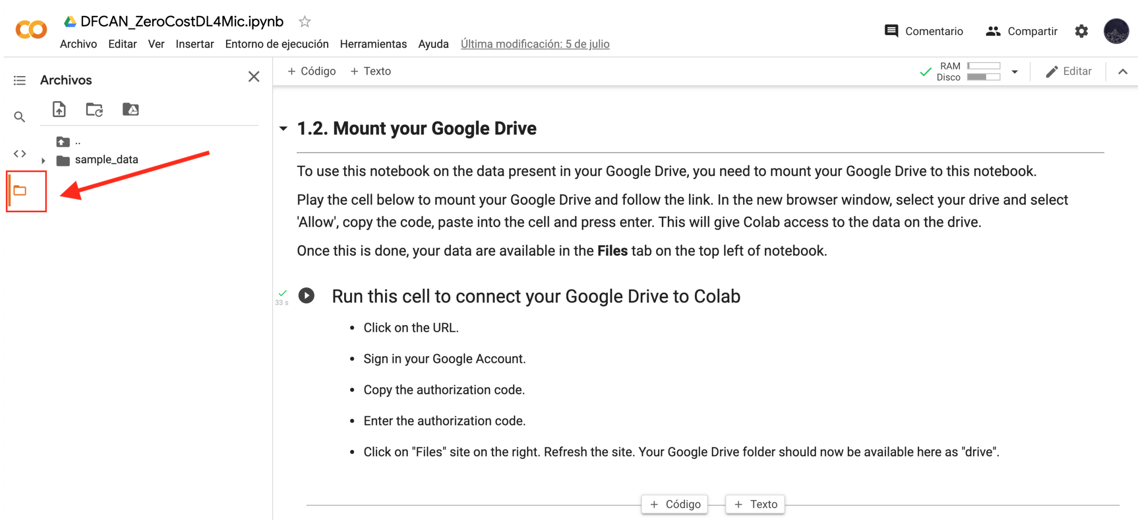
Run this cell to connect your Google Drive to Colab

- 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".

```
Mounted at /content/gdrive
```

Figure 6.4: Cell that connects the notebook session to the user's Google Drive account. In the cell, it is explained step by step what has to be done to connect to the Google Drive.

After connecting the session to the Google Drive, the files and folders can be accessed from the Files site situated on the notebook's left side.


DFCAN_ZeroCostDL4Mic.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Última modificación: 5 de julio
Comentario
Compartir
+ Código + Texto
RAM Disco
Editar
Archivos
..
sample_data
+ Código
+ Texto

1.2. Mount your Google Drive

To use this notebook on the data present in your Google Drive, you need to mount your Google Drive to this notebook.

Play the cell below to mount your Google Drive and follow the link. In the new browser window, select your drive and select 'Allow', copy the code, paste into the cell and press enter. This will give Colab access to the data on the drive.

Once this is done, your data are available in the **Files** tab on the top left of notebook.

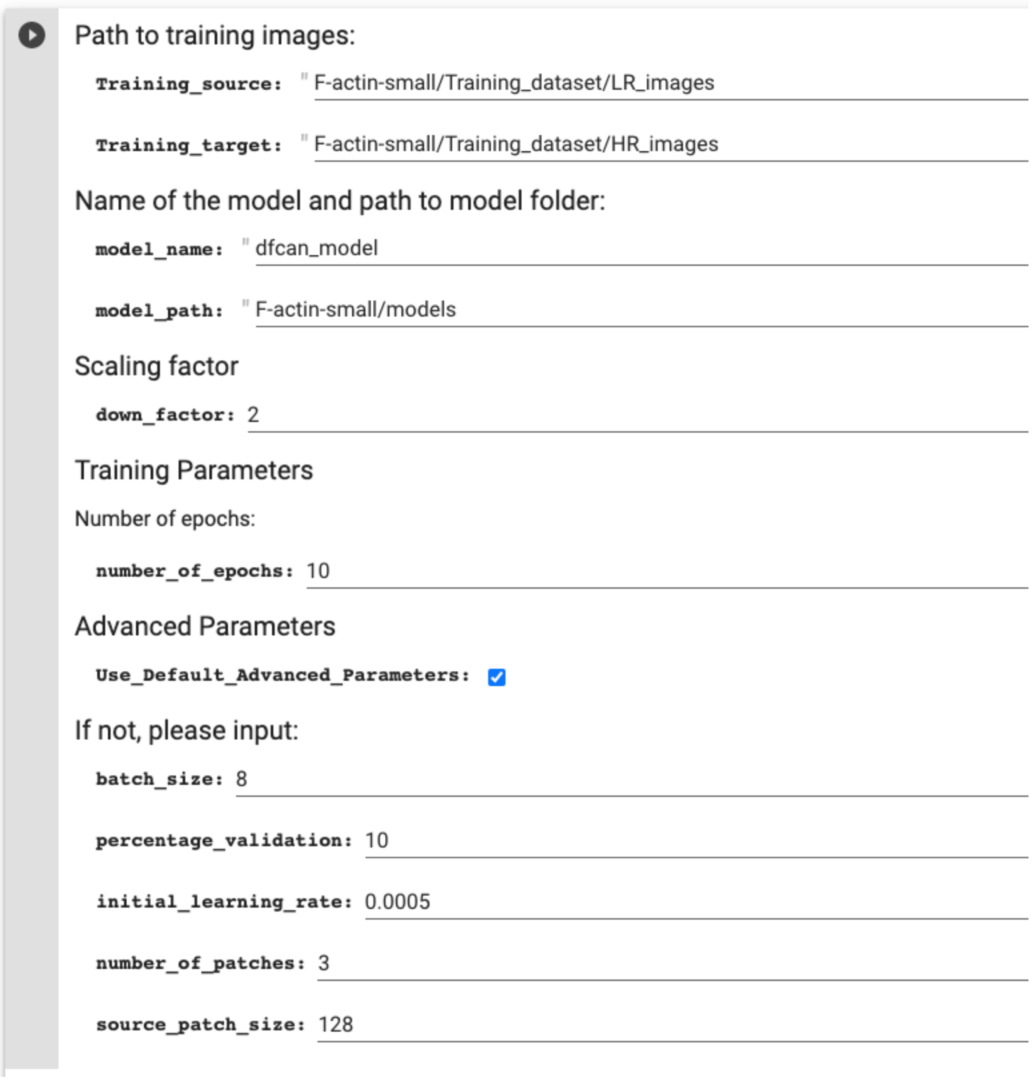
Run this cell to connect your Google Drive to Colab

- 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 6.5: Files site of the notebook. If the session is connected to the Google Drive account, the files and folders will be accessible from here.

However, if the user does not have a dataset to start the testing of the network, there is a default dataset that can be downloaded from the cell in the Section 3.1 of the notebook. After downloading it, the files and folders will be displayed in the Folders site showed in Figure 6.5.

The next 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 located above the cell displayed in Figure 6.6. All the parameters have a default value, which are the values that gave one of the best results for the default dataset.


Path to training images:

Training_source: " F-actin-small/Training_dataset/LR_images

Training_target: " F-actin-small/Training_dataset/HR_images

Name of the model and path to model folder:

model_name: " dfcan_model

model_path: " F-actin-small/models

Scaling factor

down_factor: 2

Training Parameters

Number of epochs:

number_of_epochs: 10

Advanced Parameters

Use_Default_Advanced_Parameters: ✓

If not, please input:

batch_size: 8

percentage_validation: 10

initial_learning_rate: 0.0005

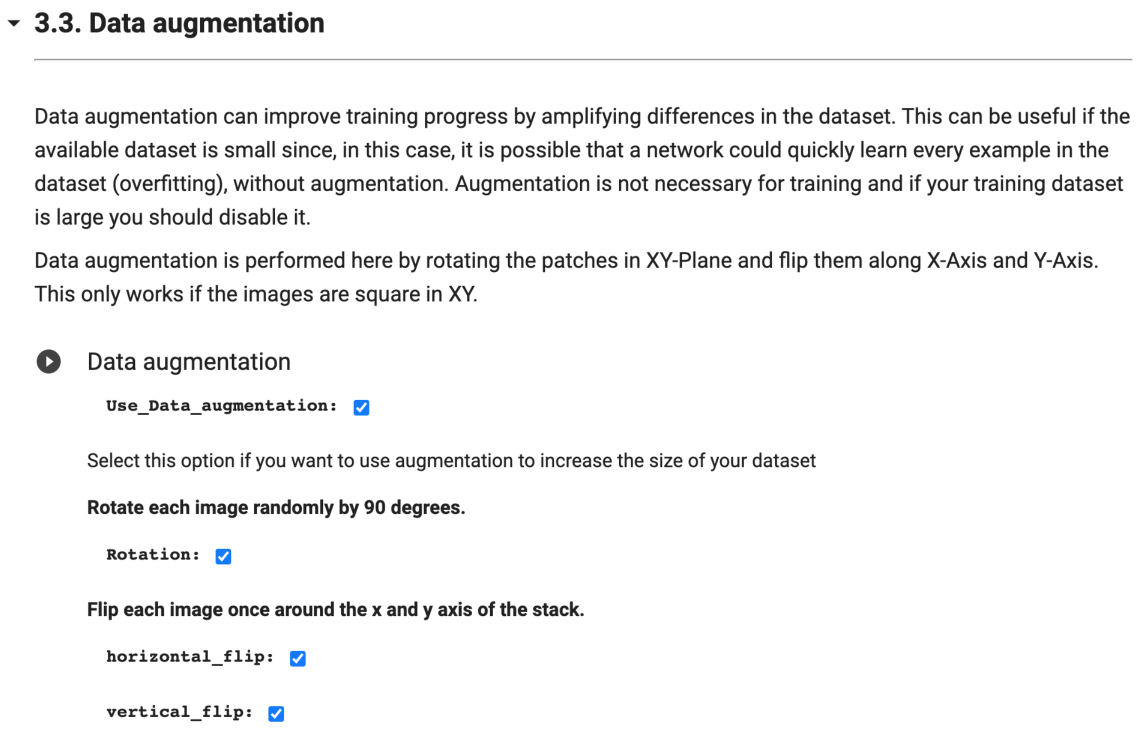
number_of_patches: 3

source_patch_size: 128

Figure 6.6: Definable parameters for the training of the network. The variables are pre-defined to adjust to the default dataset, but they have to be modified for the training with different datasets.

The last parameters are the advanced parameters, which are not recommended to be changed by the beginners. If the *Use_ Default_ Advanced_ Parameters* checkbox is checked, the network will use the values displayed in Figure 6.6 even if they have been changed. If the checkbox is not checked, the training will use the ones specified by the user.

Following the definition of the parameters, there is a cell to indicate if data augmentation is going to be applied as part of the training. As showed in Figure 6.7, if the *Use_ data_ augmentation* checkbox is checked, the dataset will be augmented by the next three parameters: rotation, horizontal flip, and/or vertical flip. Any of them can be checked for the data augmentation.



3.3. Data augmentation

Data augmentation can improve training progress by amplifying differences in the dataset. This can be useful if the available dataset is small since, in this case, it is possible that a network could quickly learn every example in the dataset (overfitting), without augmentation. Augmentation is not necessary for training and if your training dataset is large you should disable it.

Data augmentation is performed here by rotating the patches in XY-Plane and flip them along X-Axis and Y-Axis. This only works if the images are square in XY.

Data augmentation

Use_Data_augmentation: ✓

Select this option if you want to use augmentation to increase the size of your dataset

Rotate each image randomly by 90 degrees.

Rotation: ✓

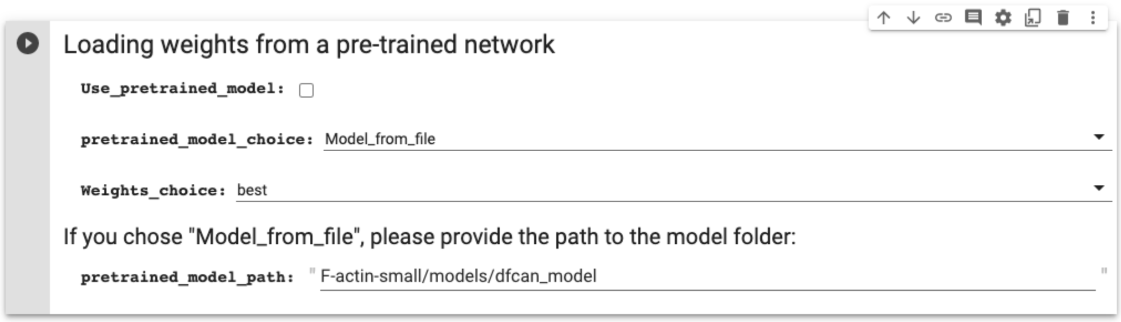
Flip each image once around the x and y axis of the stack.

horizontal_flip: ✓

vertical_flip: ✓

Figure 6.7: Executable cell that gives the option of choosing if data augmentation is desired and if affirmative, what type of data augmentation is going to be selected.

The next step after deciding about the data augmentation is choosing if the model should start training with pre-trained weights. As shown in Figure 6.8, if the checkbox is not checked, the training will start from scratch, but if not, the next parameters should be specified. *Pretrained_ model_ choice* only has one option so it can be left like that. *Weights_ choice* has two options: last and best. The former are the weights that gave the best results in the training of the model and the latter are the last weights of the training. Lastly, the *pretrained_ model_ path* must be correctly specified since it is the path where the weights file will be searched.



Loading weights from a pre-trained network

Use_pretrained_model: ☐

pretrained_model_choice: Model_from_file

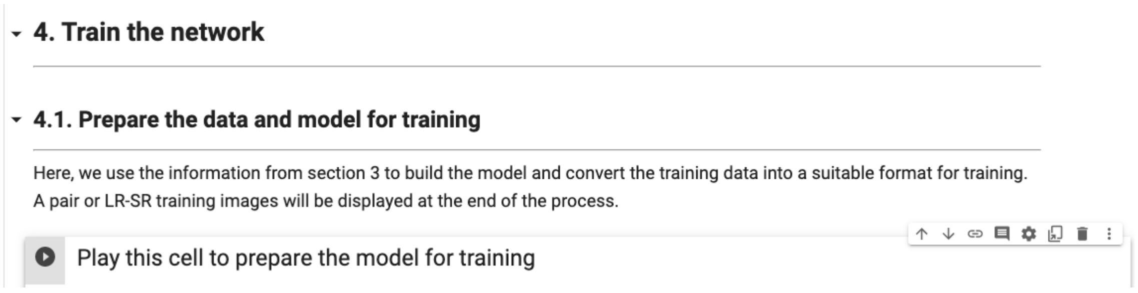
Weights_choice: best

If you chose "Model_from_file", please provide the path to the model folder:

pretrained_model_path: " F-actin-small/models/dfcan_model "

Figure 6.8: Executable cell that gives the option of loading pre-trained weights as the initial weights of the training. There are two types of weights that can be chosen, *Best* and *Last*. The former are the weights that gave the best results in the training of the model and the latter are the last weights of the training. When a DFCAN model is trained in this notebook, both types of weights are saved for future use.

The next phase consists of the training. The first cell in the training section is executed to prepare the model for the training (see Figure 6.9)



4. Train the network

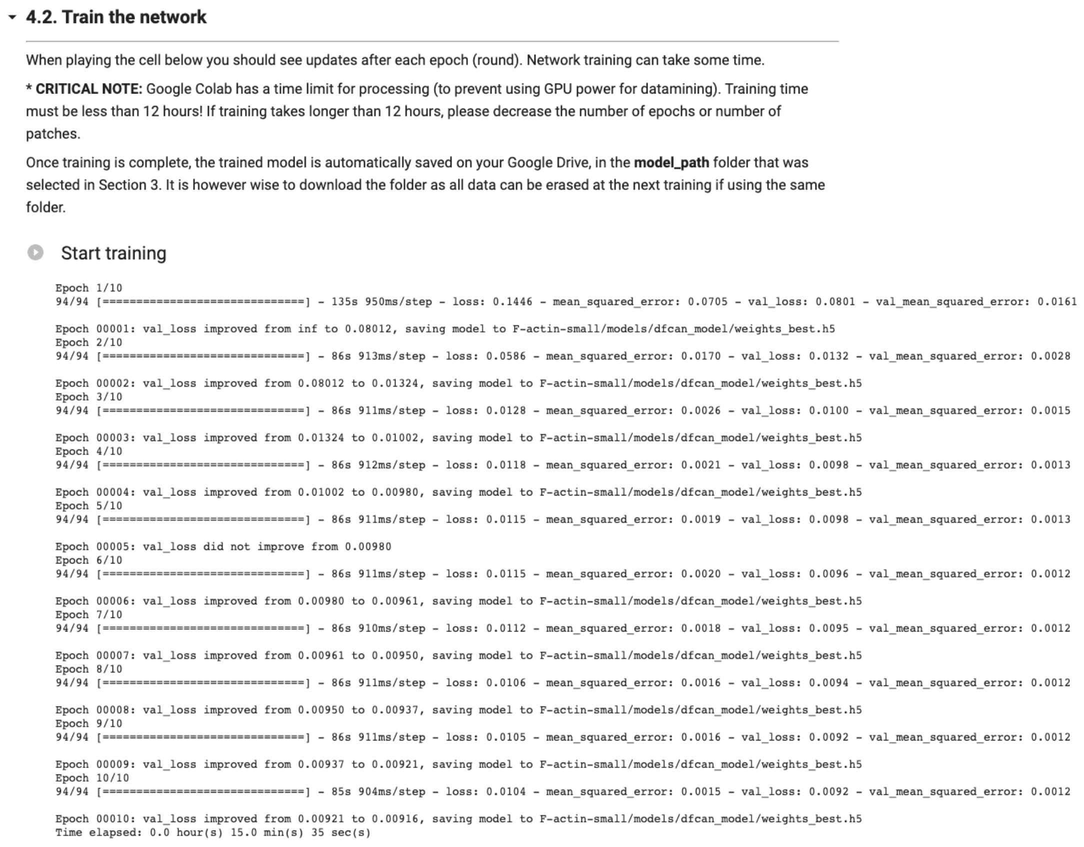
4.1. Prepare the data and model for training

Here, we use the information from section 3 to build the model and convert the training data into a suitable format for training.
A pair or LR-SR training images will be displayed at the end of the process.

Play this cell to prepare the model for training

Figure 6.9: Executable cell that prepares the model for the training. It is compulsory to execute this cell before starting the training.

On the other hand, the second cell starts the training (see Figure 6.10). After executing it, the output will show epoch by epoch what are the loss and MSE for the training and validation. For each epoch, the weights will be saved if the loss metric is better than all the loss values of the previous epochs. On the last epoch, the weights will also be saved. It can happen that the last and best weights are the ones of the last epoch, so in that case the *Best* and *Last* weight files will contain the same information.



4.2. Train the network

When playing the cell below you should see updates after each epoch (round). Network training can take some time.

* **CRITICAL NOTE:** Google Colab has a time limit for processing (to prevent using GPU power for datamining). Training time must be less than 12 hours! If training takes longer than 12 hours, please decrease the number of epochs or number of patches.

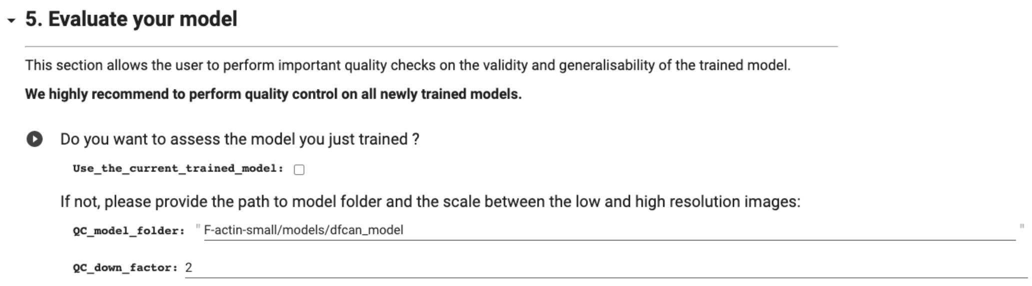
Once training is complete, the trained model is automatically saved on your Google Drive, in the **model_path** folder that was selected in Section 3. It is however wise to download the folder as all data can be erased at the next training if using the same folder.

Start training

```
Epoch 1/10
94/94 [==============================] - 135s 950ms/step - loss: 0.1446 - mean_squared_error: 0.0705 - val_loss: 0.0801 - val_mean_squared_error: 0.0161

Epoch 00001: val_loss improved from inf to 0.08012, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 2/10
94/94 [==============================] - 86s 913ms/step - loss: 0.0586 - mean_squared_error: 0.0170 - val_loss: 0.0132 - val_mean_squared_error: 0.0028

Epoch 00002: val_loss improved from 0.08012 to 0.01324, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 3/10
94/94 [==============================] - 86s 911ms/step - loss: 0.0128 - mean_squared_error: 0.0026 - val_loss: 0.0100 - val_mean_squared_error: 0.0015

Epoch 00003: val_loss improved from 0.01324 to 0.01002, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 4/10
94/94 [==============================] - 86s 912ms/step - loss: 0.0118 - mean_squared_error: 0.0021 - val_loss: 0.0098 - val_mean_squared_error: 0.0013

Epoch 00004: val_loss improved from 0.01002 to 0.00980, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 5/10
94/94 [==============================] - 86s 911ms/step - loss: 0.0115 - mean_squared_error: 0.0019 - val_loss: 0.0098 - val_mean_squared_error: 0.0013

Epoch 00005: val_loss did not improve from 0.00980
Epoch 6/10
94/94 [==============================] - 86s 911ms/step - loss: 0.0115 - mean_squared_error: 0.0020 - val_loss: 0.0096 - val_mean_squared_error: 0.0012

Epoch 00006: val_loss improved from 0.00980 to 0.00961, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 7/10
94/94 [==============================] - 86s 910ms/step - loss: 0.0112 - mean_squared_error: 0.0018 - val_loss: 0.0095 - val_mean_squared_error: 0.0012

Epoch 00007: val_loss improved from 0.00961 to 0.00950, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 8/10
94/94 [==============================] - 86s 911ms/step - loss: 0.0106 - mean_squared_error: 0.0016 - val_loss: 0.0094 - val_mean_squared_error: 0.0012

Epoch 00008: val_loss improved from 0.00950 to 0.00937, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 9/10
94/94 [==============================] - 86s 911ms/step - loss: 0.0105 - mean_squared_error: 0.0016 - val_loss: 0.0092 - val_mean_squared_error: 0.0012

Epoch 00009: val_loss improved from 0.00937 to 0.00921, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Epoch 10/10
94/94 [==============================] - 85s 904ms/step - loss: 0.0104 - mean_squared_error: 0.0015 - val_loss: 0.0092 - val_mean_squared_error: 0.0012

Epoch 00010: val_loss improved from 0.00921 to 0.00916, saving model to F-actin-small/models/dfcan_model/weights_best.h5
Time elapsed: 0.0 hour(s) 15.0 min(s) 35 sec(s)
```

Figure 6.10: Executable cell that starts the training of the model. The training will go on for as many epochs as defined in the parameter *number_ of_ epochs* from Figure 6.6.

Once the training has finished, there is an option to evaluate the model by using a part of the dataset that the model has not seen in the training. This evaluation is very recommendable for newly trained models. The first step is to indicate if the model to be evaluated is the one trained in that session or a saved one. If the chosen option is a saved model, it has to be indicated where is it saved in the *QR_ model_folder* parameter (see Figure 6.11) and the scale factor of the saved model.



5. Evaluate your model

This section allows the user to perform important quality checks on the validity and generalisability of the trained model.

We highly recommend to perform quality control on all newly trained models.

Do you want to assess the model you just trained ?

Use_the_current_trained_model: ☐

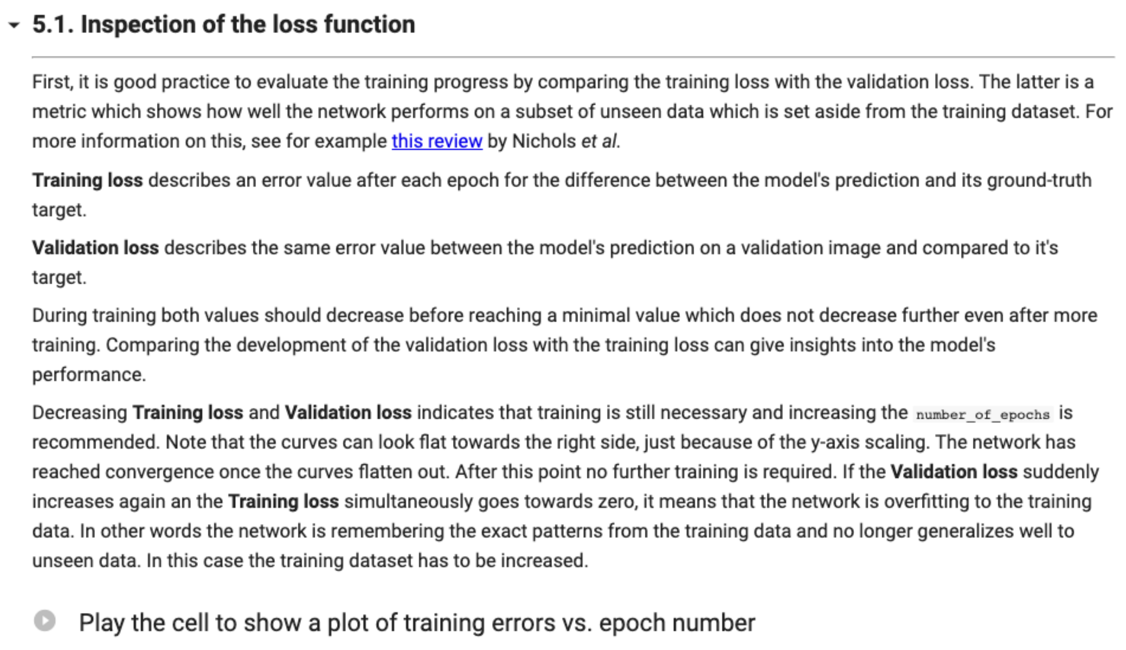
If not, please provide the path to model folder and the scale between the low and high resolution images:

QC_model_folder: " F-actin-small/models/dfcan_model "

QC_down_factor: 2

Figure 6.11: Executable cell that provides the option to evaluate the currently trained model or a saved model.

After all the parameters have been chosen, 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 6.12).



5.1. Inspection of the loss function

First, it is good practice to evaluate the training progress by comparing the training loss with the validation loss. The latter is a metric which shows how well the network performs on a subset of unseen data which is set aside from the training dataset. For more information on this, see for example this review by Nichols *et al.*

Training loss describes an error value after each epoch for the difference between the model's prediction and its ground-truth target.

Validation loss describes the same error value between the model's prediction on a validation image and compared to it's target.

During training both values should decrease before reaching a minimal value which does not decrease further even after more training. Comparing the development of the validation loss with the training loss can give insights into the model's performance.

Decreasing **Training loss** and **Validation loss** indicates that training is still necessary and increasing the `number_of_epochs` is recommended. Note that the curves can look flat towards the right side, just because of the y-axis scaling. The network has reached convergence once the curves flatten out. After this point no further training is required. If the **Validation loss** suddenly increases again an the **Training loss** simultaneously goes towards zero, it means that the network is overfitting to the training data. In other words the network is remembering the exact patterns from the training data and no longer generalizes well to unseen data. In this case the training dataset has to be increased.

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

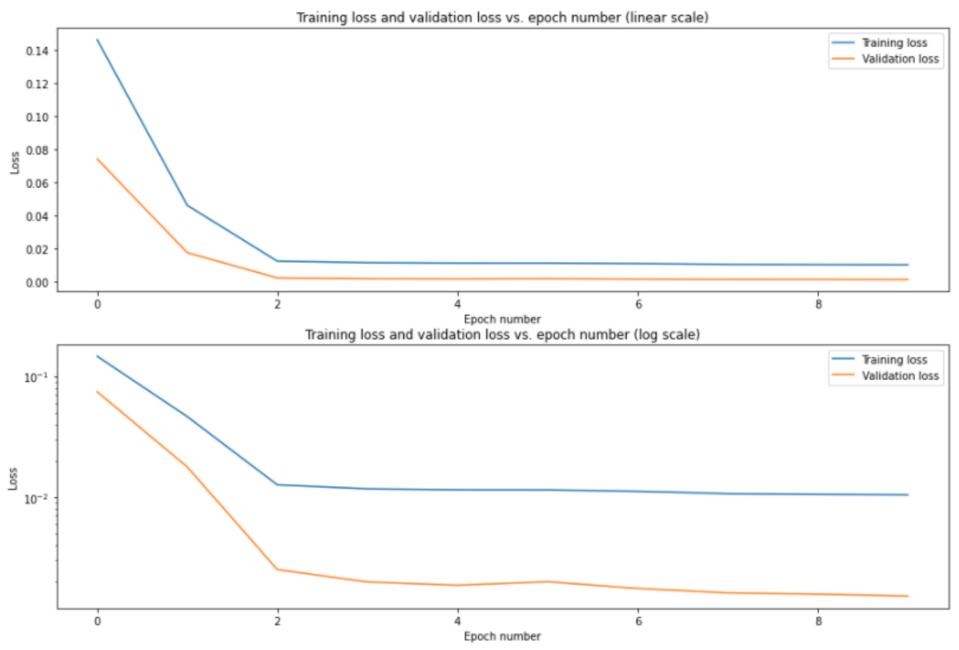
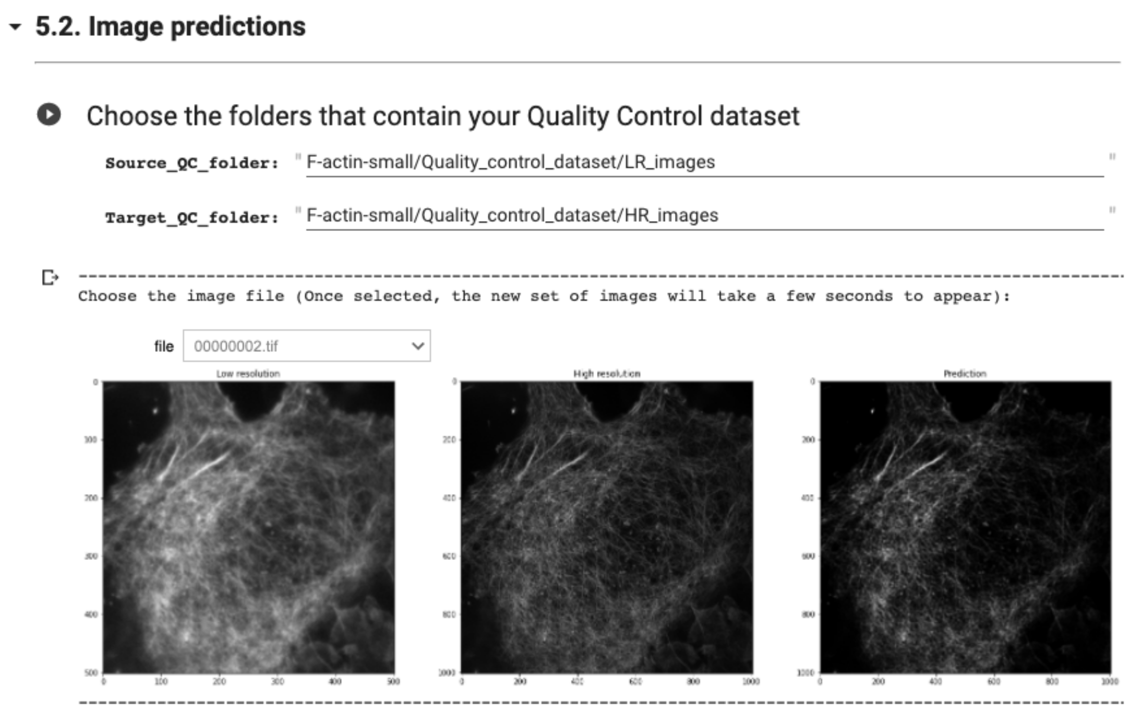


Figure 6.12: Evaluation charts, the output of the evaluation executable cell. The guide to interpret the charts is just above the executable cell.

The following cell in Section 5.2 of the notebook, predicts the images of the quality control dataset, whose path has to be indicated before executing the cell in Figure 6.13. The output is a widget from which it can be chosen the wanted file from all of the predicted ones. Each option displays the low resolution, high resolution and predicted images.



5.2. Image predictions

Choose the folders that contain your Quality Control dataset

Source_QC_folder: "F-actin-small/Quality_control_dataset/LR_images"

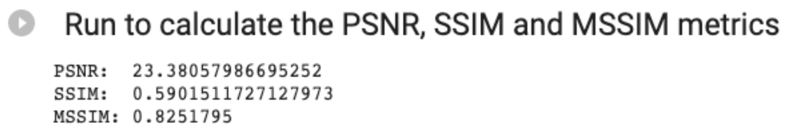
Target_QC_folder: "F-actin-small/Quality_control_dataset/HR_images"

```
-----------------------------------------------------------------------------------------------------
Choose the image file (Once selected, the new set of images will take a few seconds to appear):
```

file 00000002.tif

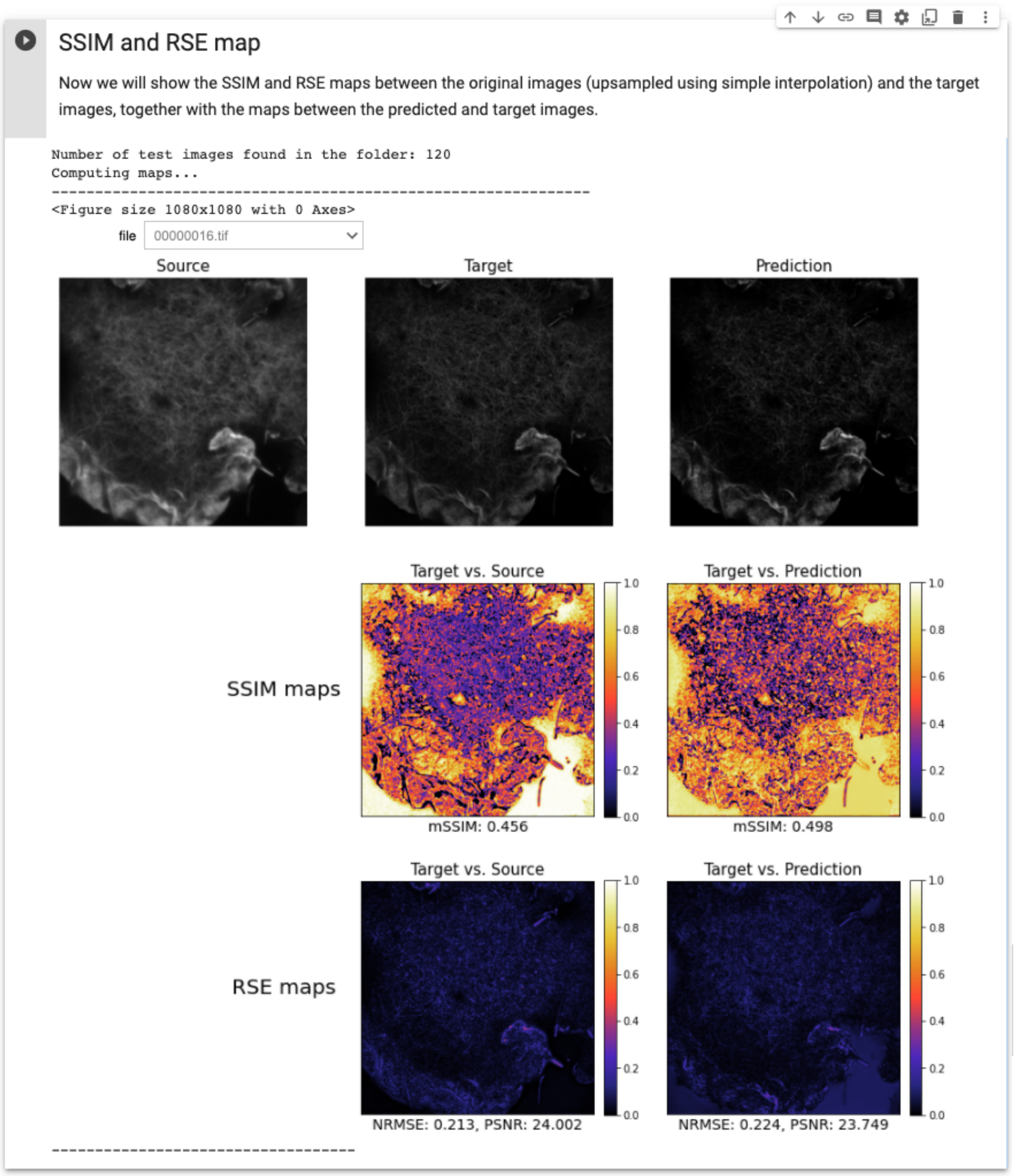
Figure 6.13: Executable cell that displays a set of low resolution, high resolution and predicted images. As the output is a widget, it provides the option of choosing among all the predicted images.

Finally, the last evaluation method is calculating the PSNR, SSIM and MSSIM metrics. The first executable cell from the Section 5.3 of the notebook (see Figure 6.14) calculates those metrics as an average of all the dataset. On the other hand, the second executable cell of the section provides the option of watching the SSIM and RSE maps between the original images (upsampled using simple interpolation) and the target images, together with the maps between the predicted and target images. Figure 6.15 presents an example of the way the metrics are computed and displayed. This method allows the user to visually compare the difference between the low definition image and the predicted one, as both of them are compared to the ground truth image.


Run to calculate the PSNR, SSIM and MSSIM metrics

```
PSNR:  23.38057986695252
SSIM:  0.5901511727127973
MSSIM: 0.8251795
```

Figure 6.14: Executable cell that calculates the PSNR, SSIM and MSSIM metrics. Each metric is an average of all the image predictions. The higher the values, the better the predictions.


SSIM and RSE map

Now we will show the SSIM and RSE maps between the original images (upsampled using simple interpolation) and the target images, together with the maps between the predicted and target images.

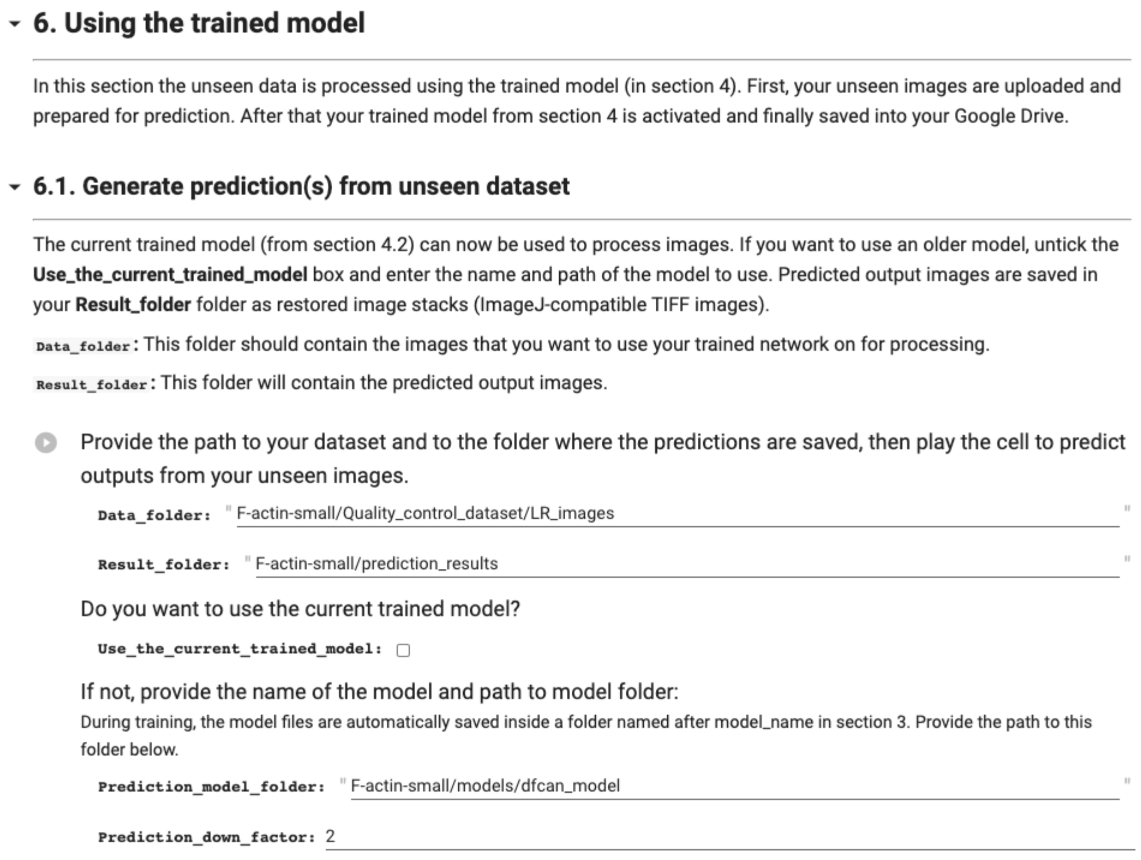
```
Number of test images found in the folder: 120
Computing maps...
-------------------------------------------------------------
<Figure size 1080x1080 with 0 Axes>
```

file 00000016.tif

```
-----------------------------------
```

Figure 6.15: Executable cell that displays the SSIM and RSE Target vs Source maps, SSIM and RSE Target vs. prediction maps for every predicted image. The image can be selected on the widget.

The final section of the notebook gives the option of using the model for predicting the super-resolution images for a dataset composed by low-resolution images. The functioning of this section is similar to the one of the predictions of the quality control dataset of the notebook (Section 5.2 of the notebook). For these last cells, there are some parameters that need to be defined. Firstly, the user needs to input the folder where the dataset to be predicted is located. Then the path where user wants the predictions to be saved. Finally, there is an option of choosing the model trained in the session for the prediction or another one, so it is not compulsory to train a network in order to use the prediction functionality of the notebook if the user has a previously trained model that she/he wants to use. If this second option is chosen there are two additional parameters that need to be specified: the path where the trained model is saved and the scale factor of that model. This can be seen in Figure 6.16.


6. Using the trained model

In this section the unseen data is processed using the trained model (in section 4). First, your unseen images are uploaded and prepared for prediction. After that your trained model from section 4 is activated and finally saved into your Google Drive.

6.1. Generate prediction(s) from unseen dataset

The current trained model (from section 4.2) can now be used to process images. If you want to use an older model, untick the **Use_the_current_trained_model** box and enter the name and path of the model to use. Predicted output images are saved in your **Result_folder** folder as restored image stacks (ImageJ-compatible TIFF images).

`Data_folder`: This folder should contain the images that you want to use your trained network on for processing.

`Result_folder`: This folder will contain the predicted output images.

Provide the path to your dataset and to the folder where the predictions are saved, then play the cell to predict outputs from your unseen images.

Data_folder: " F-actin-small/Quality_control_dataset/LR_images "

Result_folder: " F-actin-small/prediction_results "

Do you want to use the current trained model?

Use_the_current_trained_model: ☐

If not, provide the name of the model and path to model folder:

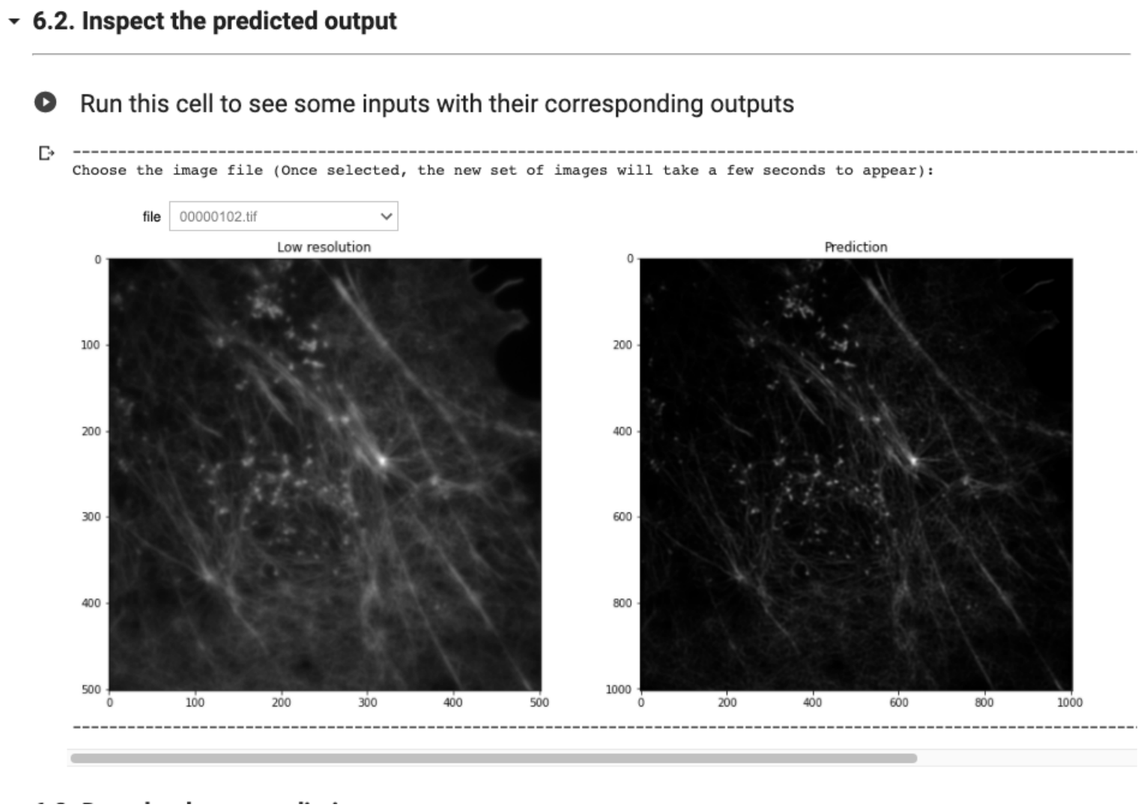
During training, the model files are automatically saved inside a folder named after model_name in section 3. Provide the path to this folder below.

Prediction_model_folder: " F-actin-small/models/dfcan_model "

Prediction_down_factor: 2

Figure 6.16

Finally, as it is shown in Figure 6.17, in the last executable cell there is a widget to choose among all the images, which displays the selected input image and its corresponding prediction. After making the predictions it is recommended to download the images if they are not saved in a folder located in the Google Drive account, as when the session is finished every file that is not saved in the Drive account is deleted.



6.2. Inspect the predicted output

Run this cell to see some inputs with their corresponding outputs

```
-------------------------------------------------------------------------------------------------
Choose the image file (Once selected, the new set of images will take a few seconds to appear):
```

Figure 6.17: Executable cell containing a widget to choose among all the LR-Prediction images, which displays the selected input image and its corresponding prediction.

CHAPTER 7

Conclusions

As previously mentioned in Chapter 1, one of the objectives of this project was solving the super-resolution task by studying modern deep learning methods, particularly for microscopy images. This goal was achieved in Chapters 2 and 3 by investigating different deep learning methods, their functioning and results.

The second goal was to implement one of the most recent networks to solve the super-resolution task. The chosen network to complete this goal was the DFCAN network, one of the most recent networks for the SR task as it was published the same month that this project was started. To further develop this project, the notebook was made following the guidelines of the ZeroCostDL4Mic team, 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.

Additionally, the DFCAN network was submitted to a thorough testing explained in Chapter 5. Firstly, the network was tested with the F-actin dataset, provided by the network's authors. The best hyperparameter set was the number 6 from Table 5.1 and the predicted images show good results, as they drastically improve the quality of the low resolution images. Secondly, the network was tested with an electron microscopy image dataset and as shown in Table 5.5, the results are slightly better, so the type of image to be predicted can influence the network's performance.

Lastly, in the original DFCAN publication, the authors claim their network surpasses the performance of the RCAN network, so in order to strengthen the experiments, the same dataset used for DFCAN was used with the RCAN network. The results obtained by this second network are in fact better than the DFCAN results, so even though the DFCAN is a more recent network, the RCAN still obtains better quantitative results, as the RCAN evaluation metrics were better than the DFCAN ones.

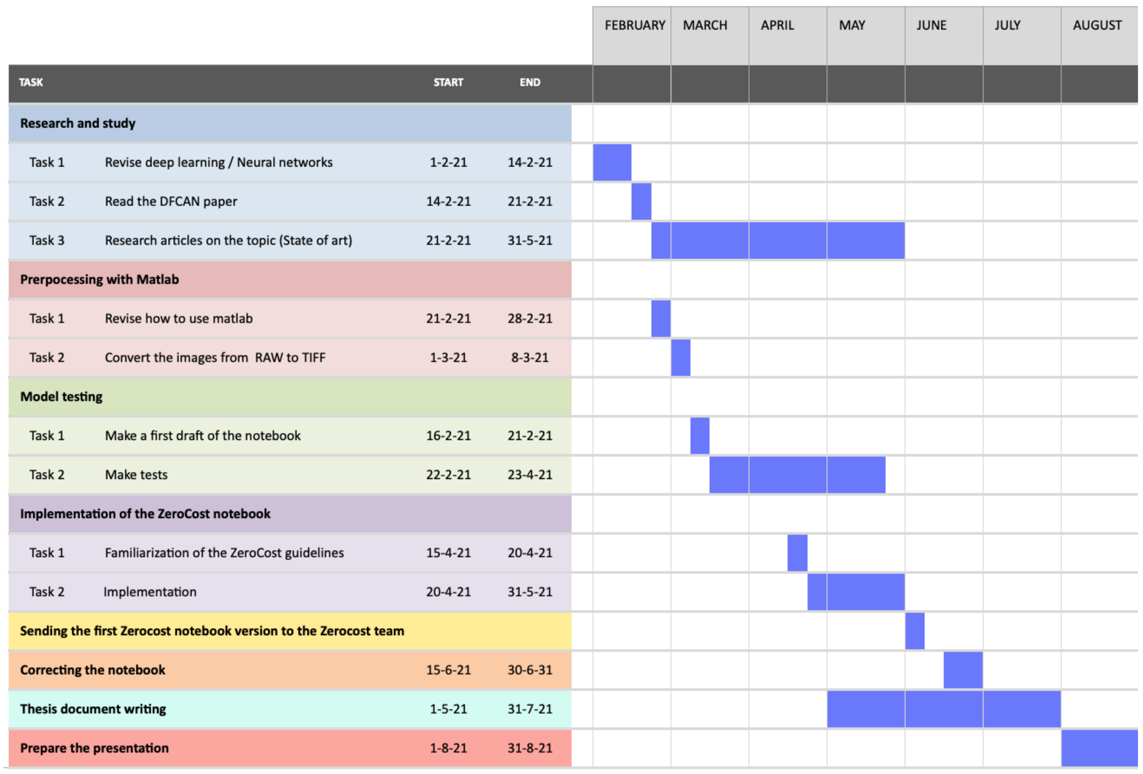
That said, none of the results achieve the quality of the high resolution images, which leads to the conclusion that the investigation of methods to solve the SR task is not finished.

All in all, it is important to point out that even though the predicted images are not as good as the HR images, the results can be of great use for the biologists, since the improvement from a LR image to a SR image 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

APPENDIX A

Gantt Diagram



TASK		START	END	FEBRUARY	MARCH	APRIL	MAY	JUNE	JULY	AUGUST
Research and study										
Task 1	Revise deep learning / Neural networks	1-2-21	14-2-21	▨						
Task 2	Read the DFCAN paper	14-2-21	21-2-21	▨						
Task 3	Research articles on the topic (State of art)	21-2-21	31-5-21	▨	▨	▨	▨			
Perprocessing with Matlab										
Task 1	Revise how to use matlab	21-2-21	28-2-21	▨						
Task 2	Convert the images from RAW to TIFF	1-3-21	8-3-21		▨					
Model testing										
Task 1	Make a first draft of the notebook	16-2-21	21-2-21		▨					
Task 2	Make tests	22-2-21	23-4-21		▨	▨	▨			
Implementation of the ZeroCost notebook										
Task 1	Familiarization of the ZeroCost guidelines	15-4-21	20-4-21			▨				
Task 2	Implementation	20-4-21	31-5-21			▨	▨			
Sending the first Zerocost notebook version to the Zerocost team								▨		
Correcting the notebook		15-6-21	30-6-31					▨		
Thesis document writing		1-5-21	31-7-21				▨	▨	▨	
Prepare the presentation		1-8-21	31-8-21							▨

Figure A.1: Gantt diagram displaying the tasks of the thesis with their respective duration

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[2] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.*, pp. 4681–4690, 2017.

[3] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition Workshops*, pp. 136–144, 2017.

[4] D.-H. Trinh, M. Luong, F. Dibos, J.-M. Rocchisani, C.-D. Pham, and T. Q. Nguyen, "Novel example-based method for super-resolution and denoising of medical images," *IEEE Transactions on Image processing*, vol. 23, no. 4, pp. 1882–1895, 2014.

[5] C. Qiao, D. Li, Y. Guo, C. Liu, T. Jiang, Q. Dai, and D. Li, "Evaluation and development of deep neural networks for image super-resolution in optical microscopy," *Nature Methods*, vol. 18, no. 2, pp. 194–202, 2021.

[6] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[7] L. von Chamier, R. F. Laine, J. Jukkala, C. Spahn, D. Krentzel, E. Nehme, M. Lerche, S. Hernández-Pérez, P. K. Mattila, E. Karinou, *et al.*, "Democratis-

ing deep learning for microscopy with ZeroCostDL4Mic," *Nature Communications*, vol. 12, no. 1, pp. 1–18, 2021.

[8] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang, and L. Zhang, "Image super-resolution: The techniques, applications, and future," *Signal Processing*, vol. 128, pp. 389–408, 2016.

[9] C. E. Duchon, "Lanczos filtering in one and two dimensions," *Journal of Applied Meteorology and Climatology*, vol. 18, no. 8, pp. 1016–1022, 1979.

[10] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[11] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, 2002.

[12] H. Chang, D.-Y. Yeung, and Y. Xiong, "Super-resolution through neighbor embedding," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, IEEE, 2004.

[13] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[15] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2, pp. 1398–1402, Ieee, 2003.

[16] D. M. Rouse and S. S. Hemami, "Analyzing the role of visual structure in the recognition of natural image content with multi-scale SSIM," in *Human Vision and Electronic Imaging XIII*, vol. 6806, p. 680615, International Society for Optics and Photonics, 2008.

[17] V. Naidu, "Discrete cosine transform-based image fusion," *Defence Science Journal*, vol. 60, no. 1, p. 48, 2010.

[18] V. H. Phung and E. J. Rhee, "A deep learning approach for classification of cloud image patches on small datasets," *Journal of Information and Communication Convergence Engineering*, vol. 16, no. 3, pp. 173–178, 2018.

[19] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on Computer Vision*, pp. 184–199, Springer, 2014.

[20] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao, "Deep learning for single image super-resolution: A brief review," *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[21] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1646–1654, 2016.

[22] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1637–1645, 2016.

[23] Y. Fan, J. Yu, and T. S. Huang, "Wide-activated deep residual networks based restoration for bpg-compressed images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[24] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016.

[25] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[27] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[28] N. Kasthuri, K. J. Hayworth, D. R. Berger, R. L. Schalek, J. A. Conchello, S. Knowles-Barley, D. Lee, A. Vázquez-Reina, V. Kaynig, T. R. Jones, *et al.*, "Saturated reconstruction of a volume of neocortex," *Cell*, vol. 162, no. 3, pp. 648–661, 2015.

[29] L. Fang, F. Monroe, S. W. Novak, L. Kirk, C. R. Schiavon, B. Y. Seungyoon, T. Zhang, M. Wu, K. Kastner, A. A. Latif, *et al.*, "Deep learning-based point-scanning super-resolution imaging," *Nature Methods*, vol. 18, no. 4, pp. 406–416, 2021.