University of Padova

Department of Information Engineering

Master's Degree in ICT for Internet and Multimedia

Image Super-Resolution with Adversarial Learning

Supervisor

Prof. Pietro Zanuttigh

Co-supervisors:

Gianluca Agresti

Umberto Michieli

Master's candidate

Davide Boem

ID: 1176946

07/10/2019

Academic Year 2018/2019

Contents

List of Figures

Abstract

Single-image super-resolution refers to the problem of generating a high-resolution image from a low-resolution one. The task arises in a wide number of real-world applications, such as digital imaging software, browsers' renders or image restoration.

In the last 30 years, many different approaches have been proposed and solutions obtaining both good perceptual quality and high evaluation metrics have been designed. The difficulty of finding solutions able to predict detailed and realistic textures, which are common in natural images, represents one of the largest limitations in the field.

In this work we address to the problem of single-image super-resolution of degraded low-resolution images, where the downsampling and degradation models are unknown. In addition, we consider the more challenging task of learning a model for single-image super-resolution that does not require low-high resolution image pairs during its design.

We face this task by adopting Convolutional Neural Networks, in particular we combine adversarial learning with techniques aimed to preserve the color information and the spatial smoothness on the produced high resolution images. We try to implement a cyclic structure in which the images are firstly denoised and deblurred, and then shifted to the desired scale.

1

Introduction

In this work we address the problem of single-image super-resolution (SISR), which consists in generating a high-resolution (HR) image from a low-resolution (LR) one. More specifically, given a LR input image and a scaling factor, the goal of SR is to generate a HR image with dimensions coherent with the specified scale and having an enhanced version of the LR image as content.

The task of single-image super-resolution is useful to a number of real-world applications. A common scenario occurs when it is necessary to increase the resolution of an image when enlarging it using graphics editors.

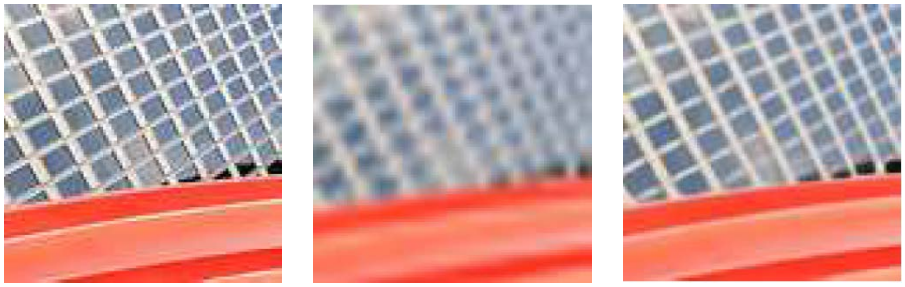
Another application can be found in the renders of web pages. In order to shorten the response time of browsing web pages, images are often shown in LR form (the so called *thumbnails*). The enlarged HR image is shown only when the user clicks on the *thumbnail* and an HTTP request is sent to the web server. This approach requires the HR image to be stored in a web server, and demands usage of time and bandwidth in order to ask for and download the desired image on the user's machine. To save storage space, communication bandwidth and download time, it would be desirable if the LR image is downloaded and then locally enlarged in the user's machine when needed.

Single-image super resolution may also come in help in the post processing of images acquired with cameras of mobile phones. Mobile phones manufacturers tend to save money by using cheap camera hardware and then to enhance the pictures through computer vision algorithms, for example SR techniques.

SR techniques may also be applied in the field of lossy compression. The encoder can downsample HR images using classic approaches and subsequently encode them using a compression algorithm (JPEG, for example). The decoder would simply decode the LR image and then shift it to the HR domain using a SR technique.

A final application arises in the restoration of old, historic photographs, the so called *image inpainting*. Besides reverting deteriorations in the photographs, it

is sometimes beneficial to also enlarge them with increased resolution for display purposes and single-image super-resolution may come in help.
In the last 30 years, a number of solutions have been proposed and dramatic improvements have started to be obtained since the beginning of the deep learning era. Recently, many deep learning models able to produce outstanding results have been proposed. However, large room for improvement is present when dealing with the problems of noisy cameras or texture reconstruction. SR techniques are necessary, since detailed and realistic textures are present in many examples and, as Figure 1.1 shows, classic interpolation techniques have almost no use.



(a) Original HR image.

(b) Reconstruction of the HR image via bicubic interpolation of its downsampled copy.

(c) Reconstruction of the HR image via the SISR technique in [5].

Figure 1.1: Examples of texture reconstruction.

In this work we tried to make the SR task even more challenging by choosing to:

- Upsample up to a factor of 4, i.e. maximum value usually adopted in SR challenges.
- Apply super-resolution to images affected by degradation and blur. The characteristics of the smoothing and the noise are unknown, so it is not possible to mitigate their effect through some image pre-processing.
- Train the models without using LR-HR image pairs. Hence, the training procedure will be unsupervised.

To tackle this problem we implement a cycle-in-cycle CNN model derived from [8] able to jointly denoise and super-resolve LR image to the desired scale. The basic components of this approach are going to be the Generative Adversarial Networks (GANs), whose applications have shown impressive results since they have been

proposed in 2014, and the CycleGAN framework, one of the most outstanding works in the field of image-to-image translation.
The reminder of this thesis is organized as follows. Chapter 2 is going to introduce the problem of super-resolution and how the SR approaches are usually classified. It will also give an overview on the related work that has been performed on the task in the last 30 years. To conclude, it will provide a small introduction to Convolutional Neural Networks and to the GAN and CycleGAN frameworks, in order to help the reader to better understand the content that will follow. Chapter 3 will describe the model that has been chosen as starting point for our experiments and its various sub-parts, lingering also in the networks' architectures. Chapter 4 will discuss the training techniques adopted for the models and the choice of the hyper-parameters. Chapter 5 is going to the main results obtained from the experiments. Finally, Chapter 6 will summarize the work that has been performed and some conclusions will be given, along with some ideas for future works.

2

Super-Resolution

Single-image super-resolution refers to the task of mapping an input low resolution image to a single high resolution image. Depending on how this mapping is obtained, SR techniques can be classified in three principal approaches:

1. Inverse problem methods, where SR is seen as an ill-posed problem and regularization tools are used to solve it.

2. Machine learning methods, that use a dictionary and aim at learning the mapping through classic ML tools.

3. Deep learning methods, that address the problem of single-image super-resolution by exploiting deep learning techniques, such as supervised or unsupervised learning via deep convolutional neural networks (CNNs), adversarial learning, and other procedures.

The following section will give a brief overview on the work related to the task of single-image super-resolution that has been produced from the mid-nineties up to now. Examples related to all the three principal approaches shall be given, but the focus will remain on the third category, i.e. the one involving deep learning based solutions, in particular CNNs. An overview on CNNs will be given in Section 2.2.1. Deep learning methods can be further partitioned in two main classes: supervised deep learning methods and unsupervised deep learning methods.
In supervised learning, the data samples used in the training procedure come together with their *ground truth*, which means that the real data output, the one that the learning method should be able to reproduce (or, at least, approximate) is available at training time. The ground truth is often used when computing the loss function, i.e. a distance metric between the produced output and the desired output sampled from real data.
In unsupervised learning, the ground truth is not available at training time (and often not even at test time). The challenge posed by this kind of learning consists in finding previously unknown patterns in the examples coming from the training

set and to use these patterns to cluster the data samples into a fixed or variable number of classes. If creation of samples has to be performed instead of classification, the learning becomes even more challenging.
Nowadays, Convolutional Neural Networks are an essential tool when facing Computer Vision problems and for this reason, they will be better introduced in the following sections. A great help in the task of data generation is given by the GAN framework. Generative Adversarial Networks can speed up and improve the task of samples generation by employing two networks that play an adversarial game.
The GAN framework is widely used in many SR techniques and it will be seen in larger detail in Section 2.3. Another framework that is going to be presented in this chapter is CycleGAN [6], a very powerful tool in the context of image-to-image translation.

2.1 Related Work

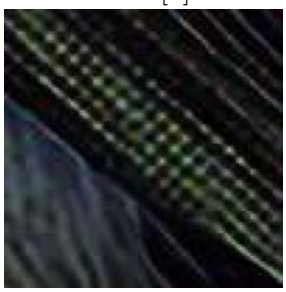
The problem of single image super-resolution has been studied for 30 years. Early approaches used interpolation techniques based on image statistics, as in [14] [12] [13]. These methods tried to adapt the interpolation at higher resolution using image statistics computed on the image at low resolution. Unfortunately, these solutions exhibited limitations in predicting detailed textures.
Other studies [9] [10] [11] relied on natural image statistics in order to reconstruct better high-resolution images.
More advanced works aim to learn mapping functions from the LR domain to the HR domain using dictionaries of images' pairs and machine learning techniques [15] [16] [17].
Recently, deep convolutional neural networks (CNNs) and their powerful capabilities led to dramatic improvements in the problem of single-image super-resolution. Since Dong *et al.* [18] [19] came up with deep learning based SR method outperforming the state of the art, many CNN architectures have been proposed.
The first CNN approaches to the super-resolution problem used to upscale the input image via bicubic upsampling, as it is possible to see in [20] and [21], but in these cases the architecture learnt an interpolation and not a super-resolution technique. A different (and now more common) approach consists in feeding LR images to a CNN containing upsampling modules at the end [22].
The ResNet architecture proposed by He *et al.* [4] allowed the construcion of deeper

CNNs by exploiting residual blocks: the nested skip connections of residual blocks allow fast and improved convergence by preventing the vanishing gradient problem. Ledig *et al.* successfully applied the ResNet architecture and set a new state of the art for image SR with high upscaling factors (×4) with a 16 blocks deep ResNet (SRResNet) optimized for MSE.
Minimizing the mean Squared Error is equivalent to maximize the *peak signal to noise ratio* (PSNR), as it will be shown in Chapter 5, but it does not guarantee better perceptual quality, in particular failing to reproduce texture details. Moreover, all these SR solutions did not address the multi-scale problem: the seen SR algorithms treat super resolution of different scale factors as independent problems without considering and utilizing mutual relationships among different scales. All those algorithms require many scale-specific networks that need to to be trained independently to deal with various scales.
These issues are addressed by EDSR and MDSR, proposed by B. Lim *et al.* in [5]. In their work they develop an enhanced deep super-resolution network (EDSR) with performance exceeding those of the state-of-the-art SR methods, which is trained by minimizing the mean absolute error. EDSR manages to reproduce detailed textures, guaranteeing higher perceptual quality.

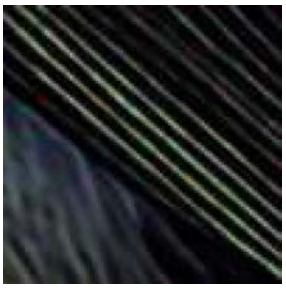
In Figure 2.1 it is possible to see that EDSR outperforms both the classic bicubic interpolation method and the SRResNet reconstruction. It preserves the detail of the ground truth image and avoids the creation of image artifacts.
Lim addresses the problem of multi-scale SR with MDSR, a multi-scale deep super-resolution system which can reconstruct high-resolution images of different - yet fixed - upscaling factors in a single model. This multi-scale architecture shares most of its variables across different scales and uses significantly fewer parameters compared with multiple single-scale models, showing comparable performances.
A milestone not only in the field of super-resolution algorithms, but in the entire deep learning era, are Generative Adversarial Networks (GANs), proposed by Ian Goodfellow *et al.* in [2]. Goodfellow proposed a new framework for estimating generative models through an adversarial process in which two networks are simultaneously trained: a generative model and a discriminative model. The generator is trained by maximizing the probability of the discriminator making a mistake. GANs were not directly thought for SR problems, but they have been exploited in a number of different works ([8] [23]) and for this reason they will be better studied in Section 2.3.
The model that is going to be examined in Chapter 3, proposed by Yuan *et al.*

(a) Example of texture from image 0853 from DIV2K dataset [7].

(b) Bicubic interpolation reconstruction.



(c) SRResNet reconstruction.



(d) EDSR reconstruction.

Figure 2.1: Comparison between SR algorithms. Classic methods as the bicubic interpolation fail to reproduce textures, SRResNet improves the reconstruction but creates some artifacts. EDSR leads to a better reconstruction with absence of artifacts.

in [8], addresses the problem of super-resolution of degraded images. It exploits both the previous models (EDSR) and the GANs framework in a cycle-in-cycle structure that allows both denoising and resolution augmentation by maintaining a high PSNR and a good perceptual quality.
This model is going to be the starting point for our work.

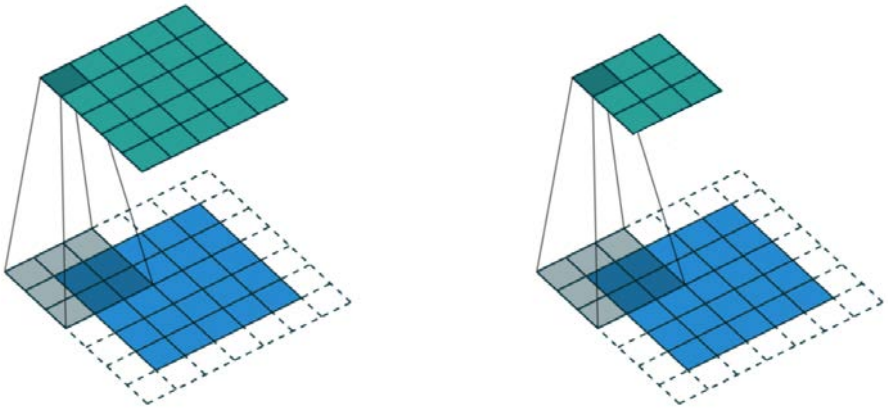
2.2 Introduction to CNNs

Convolutional Neaural Networks are a class of artificial neural networks that in the last decade has become dominant in various Computer Vision tasks.
The name *Convolutional Neural Network* indicates that the network employs a mathematical operation called convolution that is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
CNNs are particularly suited for processing data that have a grid pattern, such as

images, and they are designed to automatically and adaptively learn spatial hierarchies of features, from low to high-level patterns. This is possible because of the employment of some particular building blocks. Some common examples incude: convolutional layers, pooling layers, and fully connected layers, that are designed to automatically learn spatial hierarchies of features through a backpropagation algorithm.
The CNN architecture includes several building blocks, such as convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by a series of fully connected layers.
In this thesis, the CNNs adopted to solve the task of super-resolution of degraded images slightly differ from the ones just described. No fully connected layer is going to be employed and some particular block architectures, such as Resolution Blocks, are going to be used. The overall architecture will be better explained in Chapter 3.

2.2.1 Convolutional Layer

Convolution is a specialized type of linear operation which in the particular case of Convolutional Neural Networks is used for feature extraction. In the convolution operation, a kernel (composed by a 2D matrix of numbers), is applied across the input (denoted as *input tensor*). An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called *feature map*. This procedure is repeated applying multiple kernels, with different sizes, to each input tensor in order to form an arbitrary number of feature maps. In this way, different characteristics of the input tensors can be represented.
Some variations in the convolution operation described above can be performed. For example, it is possible to skip over some positions of the kernel to reduce the computational cost. This operation is called strided convolution and it can be seen as a downsampling of the output of the full convolution function. An example showing different types of strided convolution is represented in Figure 2.2. It is also possible to define a separate stride for each direction of motion of the kernel. If the input tensor is padded before being convolved, it is possible to maintain size coherence between input and output. A common practice is to pass the output of



(a) 2D convolution with kernel of size 3 and sride 1.

(b) 2D convolution with kernel of size 3 and sride 2.

Figure 2.2: Example of 2D convolutions.

the convolutional layer through a non-linear operation, called *activation function*. The most common activation functions used in the deep learning field are:

- Rectified Linear Unit (ReLU), defined as

$$f(x) = \begin{cases} x & \text{if x} > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{2.1}$$

- Leaky Rectified Linear Unit (Leaky ReLU), defined as

$$f(x) = \begin{cases} x & \text{if x} > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.2}$$

  where $\alpha \in [0, 1]$.

- Sigmoid function, defined as

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

- Hyperbolic tangent, defined as

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.4}$$

2.2.2 Pooling Layer

Pooling layers are responsible for reducing the spatial size of the Convolved Feature. This is necessary in order to decrease the computational power required to process the data. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.
The most common types of Pooling are:

- Max Pooling returns the maximum value from the portion of the image covered by a kernel. It also performs as a noise suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

- Average Pooling returns the average of all the values from the portion of the image covered by the kernel. It simply performs dimensionality reduction as a noise suppressing mechanism.

In most cases, Max Pooling performs quite better than Average Pooling and it is hence more used.

2.2.3 Batch Normalization Layer

Batch normalization is a method that can be used to normalize the inputs of each layer, in order to fight the internal covariate shift problem. The internal covariate shift is a problem that appears at the intermediate layers because of the continuous change of the distribution of the activations during training. This slows down the training process because each layer must learn to adapt themselves to a new distribution in every training step.
During training time, a batch normalization layer does the following:

1. Compute mean and variance of the batch:

$$\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.5}$$

$$\sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2 \tag{2.6}$$

2. Normalize the layer inputs using the previously calculated batch statistics:

$$\bar{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 - \epsilon}} \tag{2.7}$$

3. Scale and shift in order to obtain the output of the layer:

$$y_i = \gamma \bar{x}_i + \beta. \tag{2.8}$$

The parameters ϵ, γ, and β have to be learned during training.

2.3 Generative Adversarial Networks

Before the invention of GANs, the largest successes in deep learning have involved discriminative models, usually those that mapped a high-dimensional, rich sensory input to a class label. Deep generative models had less success due to the large complexity of the task.
In the work *Generative Adversarial Nets* [2], Goodfellow *et al.* proposed a new framework for estimating generative models. The GAN framework sees two actors: a generative model, that produces data samples, and a discriminative model, that learns to determine if a sample comes whether from the model distribution or the data distribution. The *discriminator* maximizes its probability of distinguishing between real samples and generated samples, while the *generator* is trained to fool it and thus to minimize that probability. This type of training can be seen as a minimax game played by the generator and the discriminator.
To go deeper in detail we have to define:

- the generator's distribution p_g over data $\mathbf{x}$;
- a prior on input noise variables $p_{\mathbf{z}}(\mathbf{z})$;
- a function G learned by the generator, where $G(\mathbf{z})$ is a mapping from the noise distribution to the data space;
- a function D learned by the generator, where $D(\mathbf{x})$ takes the value 1 if $\mathbf{x}$ comes from the data distribution and 0 if it comes from the model's distribution.

The function D is trained to maximize the probability to output the correct value to both training examples and samples from G. In the meantime, the function G is trained to minimize the value $\log(1 - D(G(\mathbf{z})))$. The minimax game played by the generator and the discriminator is hence

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

A generative adversarial network reaches its equilibrium when $p_g = p_{\mathbf{x}}$ i.e. when the generator is able to perfectly reproduce examples coming from the data distribution. When this happens, the discriminator is not able to distinguish between real or fake examples anymore and hence its output distribution assumes a uniform shape taking value 0.5.
The GAN framework is powerful, and some examples generated by a deep model applying GANs can be seen in Figure 2.3. The generated faces look almost indistinguishable from pictures of real human faces. These examples, generated just 4 years after the introduction of GANs, should just give a hint about the powerfulness and the versatility of generative adversarial networks.

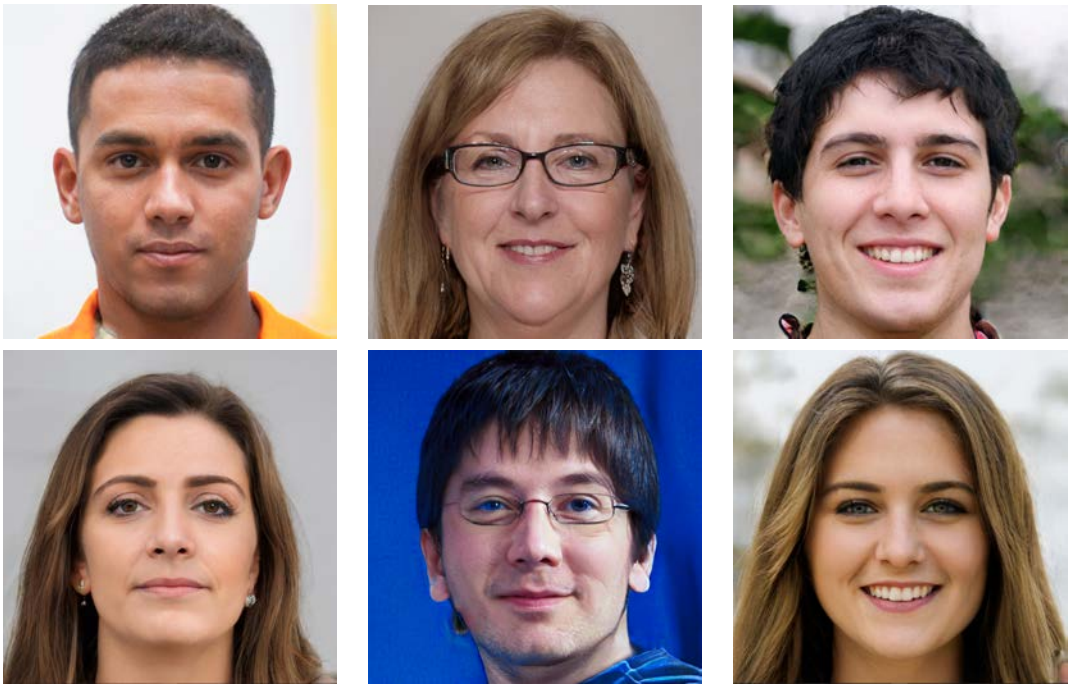


Figure 2.3: Examples of faces generated by StyleGAN [24].

Nevertheless, the GANs framework comes with some disadvantages, as the slowness and the instability of the training procedure. Since the discriminator has a simpler task, it may happen that sometimes it outperforms the generator by

correctly labelling all the samples. This fact strongly affect the effectiveness of adversarial training, since it does not allow the generator to learn the data distribution and the whole minimax game leads to nothing.

2.4 CycleGAN

Image-to-image translation is a class of Computer Vision problems where the goal is to learn the mapping between an input image from a domain X and an output image to a domain Y using a training set of aligned image pairs. If X is the domain of LR images and Y is the one of HR images, we see that we can exploit image-to-image translation techniques for the task of single image super resolution. One of the major frameworks for unsupervised image-to-image translation that we are going to use in the following chapters is CycleGAN [6]. Given one set of images from a domain X and a different set in domain Y and assuming underlying relationship between the domains – for example, that they are two different renderings of the same underlying scene – CycleGAN learns to translate between domains without paired input-output examples. In the framework, a mapping $G : X \rightarrow Y$ is learned such as the produced output $\hat{\mathbf{y}} = G(\mathbf{x})$, $\mathbf{x} \in X$, is indistinguishable from images coming from the domain Y by an adversarial classifier able to distinguish $\hat{\mathbf{y}}$ from $\mathbf{y} \in Y$. This objective can induce an output distribution over $\hat{Y}$ that matches the empirical distribution $p_{data}(\mathbf{y})$ by translating the domain X to a domain $\hat{Y}$ distributed identically to Y. However, such a translation does not guarantee that an individual input $\mathbf{x}$ and output $\hat{\mathbf{y}}$ are paired up in a meaningful way, since there are infinitely many mappings G that will induce the same distribution over $\mathbf{y}$. This issue is solved by introducing a second mapping $F : Y \rightarrow X$ trained simultaneously to G and by adding a *cycle consistency loss*, which forces $F(G(\mathbf{x})) \approx \mathbf{x}$ and $G(F(\mathbf{y})) \approx \hat{\mathbf{y}}$. This *cycle consistency loss* is thought to be placed in support of an adversarial loss, obtained by employing two discriminators, D_X and D_Y.
To sum up, the actors of this framework are:

- a mapping $G : X \rightarrow Y$ from the input domain to the output domain;
- a mapping $F : Y \rightarrow X$ from the output domain to the input domain;
- a discriminator D_X which aims to distinguish between images $\mathbf{x} \in X$ and images $F(\mathbf{y})$, $\mathbf{y} \in Y$;

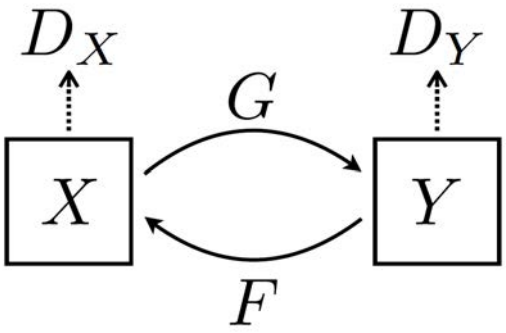


Figure 2.4: The CycleGAN model contains two mappings, $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and two adversarial discriminators D_X and D_Y. D_X encourages F to shift inputs from a domain Y into outputs indistinguishable from samples coming from X, while D_Y works on the opposite domain shift.

- a discriminator D_Y which aims to distinguish between images $\mathbf{y} \in Y$ and images $G(\mathbf{x})$, $\mathbf{x} \in X$.

As anticipated, the loss function to be minimized during the training procedure consists in the combination of an *adversarial loss* and a *cycle-consistency loss*. The adversarial loss for the mapping function G is expressed as

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{\mathbf{y} \sim p_{data}(y)}[log(D_Y(\mathbf{y}))] + \mathbb{E}_{\mathbf{x} \sim p_{data}(x)}[log(1 - D_Y(G(\mathbf{x})))], \tag{2.9}$$

where G tries to fool the discriminator by producing images $G(x)$ similar to the ones coming from the domain Y, while D_Y aims to distinguish between real samples from the domain Y and samples generated from the domain X. The minimax game played by the two actors is hence

$$\min_{G} \max_{DY} \mathcal{L}_{GAN}(G, D_Y, X, Y).$$

Similarly, for the mapping F, the objective of the game is

$$\mathcal{L}_{GAN}(F, D_X, Y, X).$$

The *cycle-consistency loss*, which assures consistency in the domain shift operation, is expressed as

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{\|\mathbf{y} \sim p_{data}(y)}[G(F(\mathbf{y})) - \mathbf{y}\|] + \mathbb{E}_{\mathbf{x} \sim p_{data}(x)}[\|F(G(\mathbf{x})) - \mathbf{x}\|]. \tag{2.10}$$

The full objective is hence the sum of the *adversarial losses* and the *cycle-consistency loss*

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \mathcal{L}_{cyc}(G, F), \quad (2.11)$$

and the training procedure is aimed to compute

$$G^*, F^* = arg \min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y) \quad (2.12)$$

i.e. the optimal mapping functions to perform a correct domain shift able to fool the adversarial discriminators.

3

Models

The main denoising strategy that we chose as starting point to tackle the task of unsupervised Single-Image Super Resolution is represented in Figure 3.1. The model is made of two nested cycles. The inner cycle learns to bring images from a low resolution, noisy and blurred domain (X) to a low resolution noise-free and blur-free domain (Y). The outer cycle aims to perform a domain shift from low resolution to high resolution domain (Z).
A cycle between two domains is intended here as a learning model able to bring an image from a domain A to a domain B and vice versa. The need of a cyclic structure comes from the fact that the training of the models is unsupervised, so cycle-consistency between the input and the output of each model is needed.

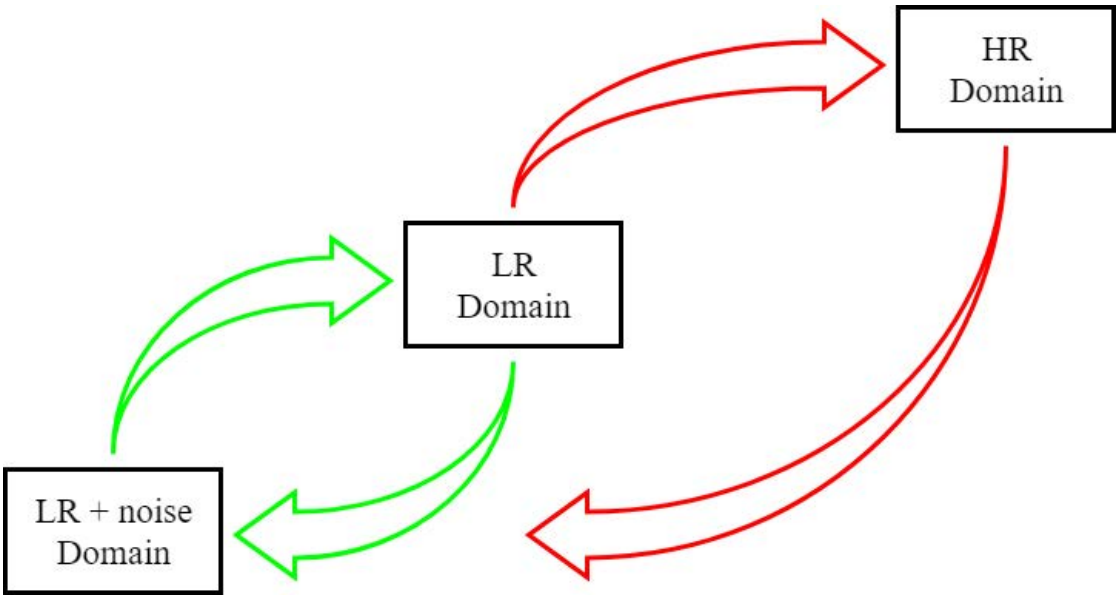


Figure 3.1: Nested cycles model.

Each cycle is made of:

- a **generative branch** consisting in one or more CNNs. This part of the cycle aims to produce an example which is going to be used in other parts of the network or to be the output of the whole model.

- a **discriminator**, which aims to distinguish between artificial examples produced by the generative branch and real samples. It is going to be used to train the generative branch in an adversarial way in order to improve the quality of the produced samples.
- a **feedback network** which is necessary in the process of unsupervised learning. It aims to reproduce the input of the network from a produced example, thus to maintain consistency between input and output.

The main parts of the model are going to be studied in the following sections. Section 3.1 is going to examine in depth the inner cycle. In particular it is going to explain how the networks are linked to each other and to introduce the loss functions that are used to train the networks. Also the architectures of the CNNs are going to be described. Section 3.2 is going to deal with EDSR, one of the more important blocks of the model. EDSR is responsible of the resolution augmentation and, differently from the other networks, it is pre-trained in a supervised way. Section 3.3 is going to better analyze the cycle-in-cycle model, its CNNs and its loss functions.
Then we propose some modifications to the CNNs and the modules presented before. First, in section 3.4 we propose a denoising solution to force the removal of some image artifacts. Then, we introduce a modification to the architecture in order to bring balance to the cycle and to improve the denosing performances, Section 3.5. Driven by the need of a faster model in terms of training time, in section 3.6 we slightly modify the architecture of EDSR [5]. Finally, in Section 3.7 we combine the two previous models in order to jointly perform image denoising and super-resolution.

3.1 Denoising

The inner cycle of the CinCGAN aims to deblur and denoise a low resolution image using a unsupervised model similar to CycleGAN. This model employs two generators and one discriminator in the adversarial creation of a deblurred and denoised version of an image, as it can be seen in Figure 3.2. In the figure, a generative CNN G_1 takes in input a blurred and noisy image $\mathbf{x}$ and learns to produce an image $\tilde{\mathbf{y}}$.
To force the Generator G_1 to produce realistic samples, it is put in a competition against a discriminative CNN D_1 whose task is to distinguish between real samples

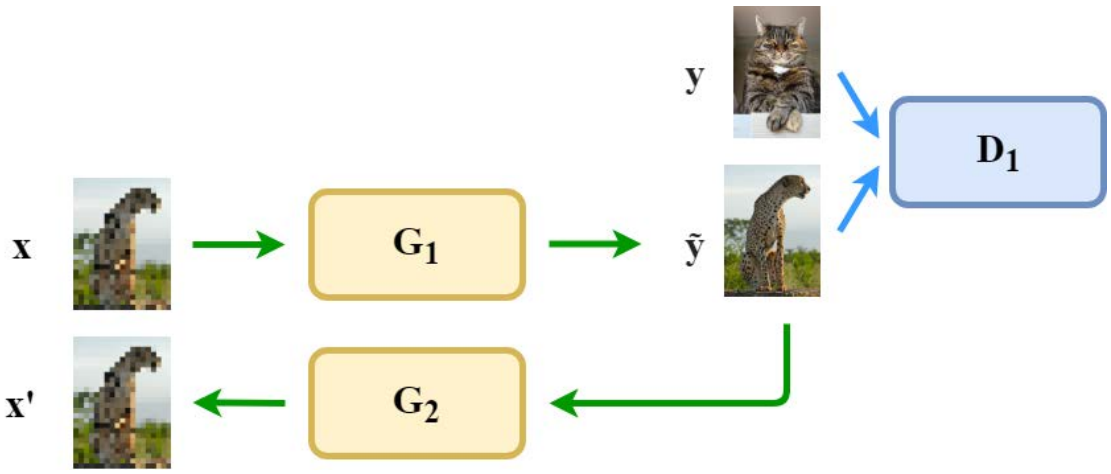


Figure 3.2: Denoising model.

$\mathbf{y}$ and generated samples $\tilde{\mathbf{y}} = G_1(\mathbf{x})$. The *generator adversarial loss* is then

$$\mathcal{L}_{GAN}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|D_1(G_1(\mathbf{x}_i)) - 1\|_2 \tag{3.1}$$

where N is the size of the batch used for a training iteration. The least squares loss is used instead of the cross entropy loss because, as the authors claim in [8], it is supposed to stabilize the training procedure.

Since the learning task is unsupervised, it is necessary to maintain consistency between the input $\mathbf{x}$ and the output $\tilde{\mathbf{y}}$ of the generator G_1. To do so, a *feedback* generator G_2 is introduced. It is trained to produce a sample $\mathbf{x}' = G_2(G_1(\mathbf{x}))$ which should be identical to the input $\mathbf{x}$. The associated *cycle consistency loss* is:

$$\mathcal{L}_{cyc}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|G_2(G_1(\mathbf{x}_i)) - \mathbf{x}_i\|_2. \tag{3.2}$$

In order to preserve the color of the produced images $\mathbf{x}$ an *identity loss*

$$\mathcal{L}_{idt}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|G_1(\mathbf{y}_i) - \mathbf{y}_i\|_1 \tag{3.3}$$

can be used, as claimed in [6].

The last cost function that is added to the model is a *total variation loss*

$$\mathcal{L}_{TV}^{LR} = \frac{1}{N}\sum_{i=1}^{N} (\|\nabla_h G_1(\mathbf{x}_i)\|_2 + \|\nabla_w G_1(\mathbf{x}_i)\|_2) \tag{3.4}$$

where $\nabla_h G_1(\mathbf{x}_i)$ and $\nabla_w G_1(\mathbf{x}_i)$ are the horizontal and vertical gradients of $G_1(\mathbf{x}_i)$. It is used to impose spatial smoothness on the generated image $\tilde{\mathbf{y}}$.
To conclude, the generators are trained in order to minimize a weighted sum of the loss functions that have just been introduced, i.e.

$$\mathcal{L}_{total}^{LR} = \mathcal{L}_{GAN}^{LR} + w_1 \mathcal{L}_{cyc}^{LR} + w_2 \mathcal{L}_{idt}^{LR} + w_3 \mathcal{L}_{TV}^{LR}, \tag{3.5}$$

where w_1, w_2, and w_3 are the weight parameters for the different losses.
The discriminator instead is trained to distinguish between real (by outputting 1) and artificial (by outputting 0) examples and his loss is defined as

$$\mathcal{L}_D^{LR} = \frac{1}{2}\left[\frac{1}{N}\sum_{i=1}^{N} \|D_1(\mathbf{y}_i) - 1\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|D_1(G_1(\mathbf{x}_i))\|_2\right]. \tag{3.6}$$

3.1.1 Networks' Architecture

To implement the denoising model, three Convolutional Neural Networks are needed. The generative models share the same architecture, which can be seen in Figure 3.3.

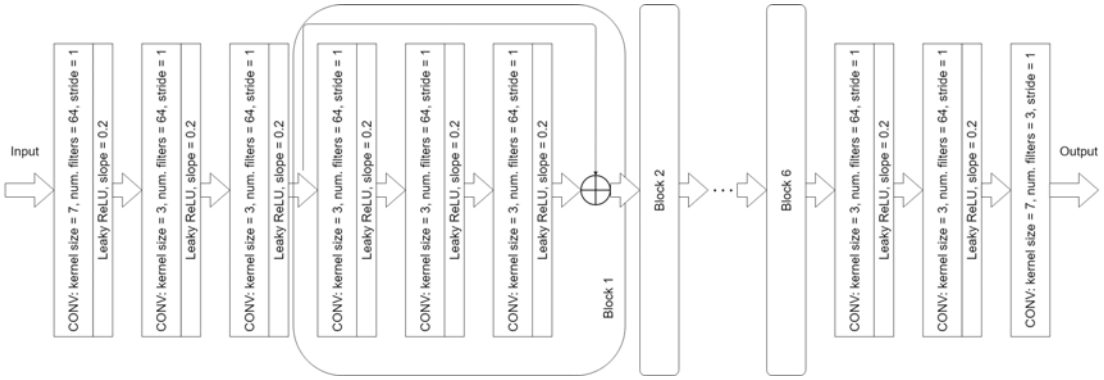


Figure 3.3: Architecture of the generative networks of the denoising model.

The networks start with 3 initial convolutions, each followed by a Leaky ReLU activation with slope of 0.2. The initial convolutions are followed by 6 Residual Blocks, each one performing a 3-layers skip. The network ends with 2 standard convolutional layers and a final acrivation-free layer outputting 3 channels. The number of variables which have to be optimized in order to train this type of network is 609731 and they are mainly localized in the Resolution Blocks.
Since the task of the discriminator is simpler, a less complex architecture is chosen, as Figure 3.4 shows.

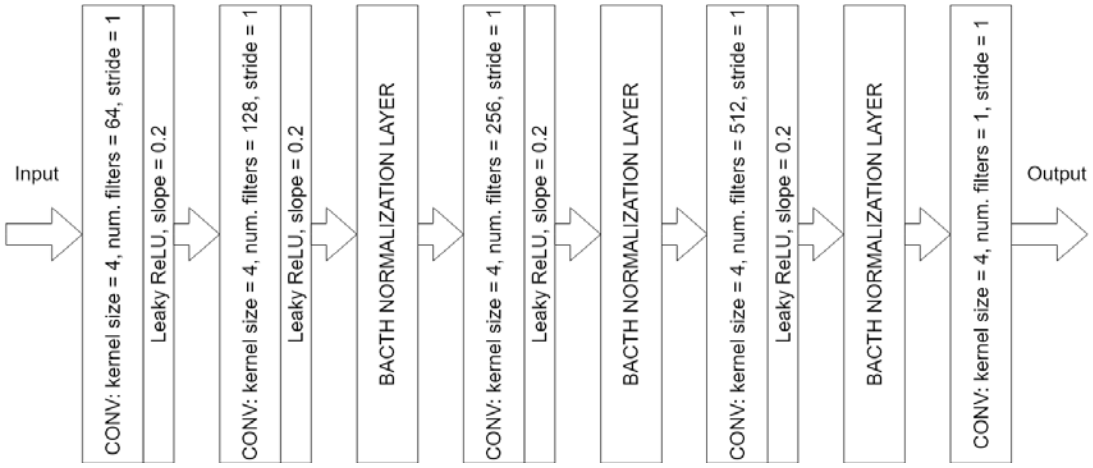


Figure 3.4: Architecture of the discriminative network of the Denoising model.

Only 5 convolutional layers and 3 Batch Normalization are employed. Its output should be an empty matrix if the input sample is real and a matrix containing only the value 0.9 if the input sample is artificial. Despite having only 8 layers, the discriminator counts 2768321 variables. This is due to the large number of filters per convolutional layer (64, 128, 256 and 512).
In total, the denoising block counts 3987783 network parameters.

3.2 Super-Resolution Module

In order to tackle the task of super resolution of denoised samples we chose to start from the *Enhanced Deep Super-Resolution* (EDSR) model [5]. EDSR performs Super-Resolution by improving the ResNet [4] model. In particular, it employs a CNN consisting in 32 improved Resolution Blocks, whose representation can be seen in Figure 3.5. In this type of architecture, where tens of layers are stacked subsequently, the residual learning framework is essential to prevent the vanishing gradient problem. The structure of the blocks employed by EDSR differs from the one proposed in [4]. As it can be seen in Figure 3.6, the Batch Normalization layers and the final convolution have been removed. This, according to [5], saves GPU memory usage and improves the overall performances. In our cycle-in-cycle approach, EDSR is placed after the denoising block and aims to increase the resolution of the images by a factor $\times 4$. EDSR is the only network in the model which is trained in a supervised way by taking a high resolution image, downsampling it by a factor 4 using the bicubic method, feeding the downsampled version to the

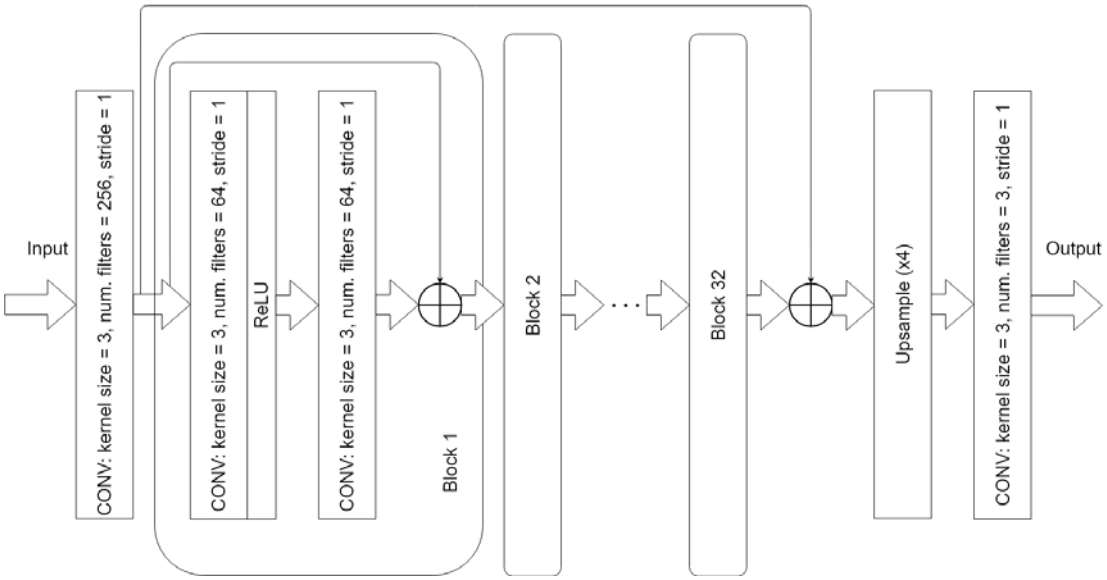
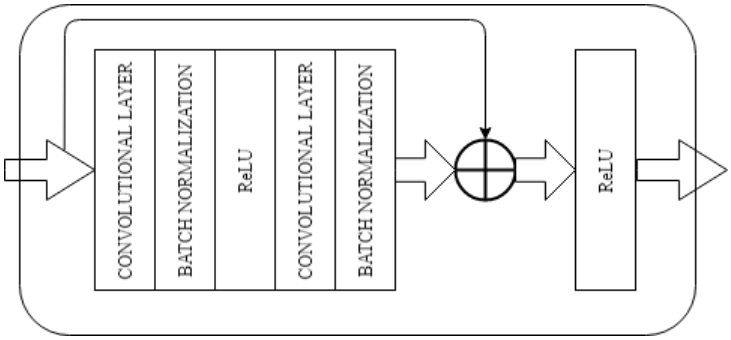


Figure 3.5: Architecture of the EDSR model.

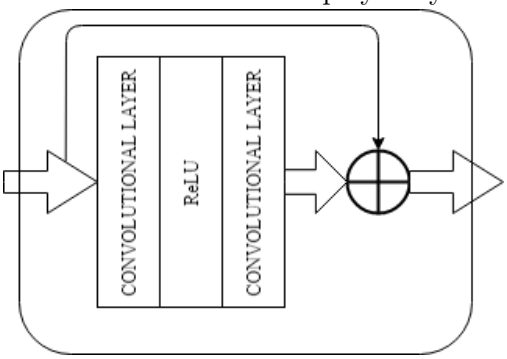
CNN and then computing the L1 loss for optimization purposes as

$$\mathcal{L}_{EDSR} = \frac{1}{N}\sum_{i=1}^{N} \|EDSR(\mathbf{z}_i') - \mathbf{z}_i\|_1, \tag{3.7}$$

where we denote as $\mathbf{z}_i'$ the bicubic downsampled version of $\mathbf{z}_i$.
The EDSR model is the largest network employed in this work, counting 3248245 optimizable parameters. Its supervised training is long and more than a week of training on a NVIDIA GTX 1080Ti GPU is needed before reaching convergence.



(a) Classic Residual Block employed by ResNET.



(b) Residual Block used by EDSR.

Figure 3.6: Comparison between Resolution Blocks.

3.3 CinCGAN

The main cycle-in-cycle structure presented at the beginning of the chapter considers the single image super-resolution problem in a general case, in the sense that couples of images at high and low resolution are unavailable and the downsampling method is unknown. Furthermore, the given low resolution images are blurred and degraded by noise of unknown nature.

This unfavourable setup translates into the need of a denoising strategy and an unsupervised learning model. Since the input images are degraded from some noise and the characteristics of this noise cannot be known or extrapolated from the images, it is necessary to include a denoising block in the super-resolution model. Moreover, since the training set consists of unpaired high/low resolution images, the learning shall be unsupervised.

Using Generative Adversarial Networks [2] and the CycleGAN [6] framework, we implement a cycle-in-cycle model to tackle the problem of single image super-resolution. The structure of the cycle-in-cycle model can be seen in Figure 3.7. The inner cycle maps a low resolution image $\mathbf{x}$ to a clean low resolution image $\tilde{\mathbf{y}}$

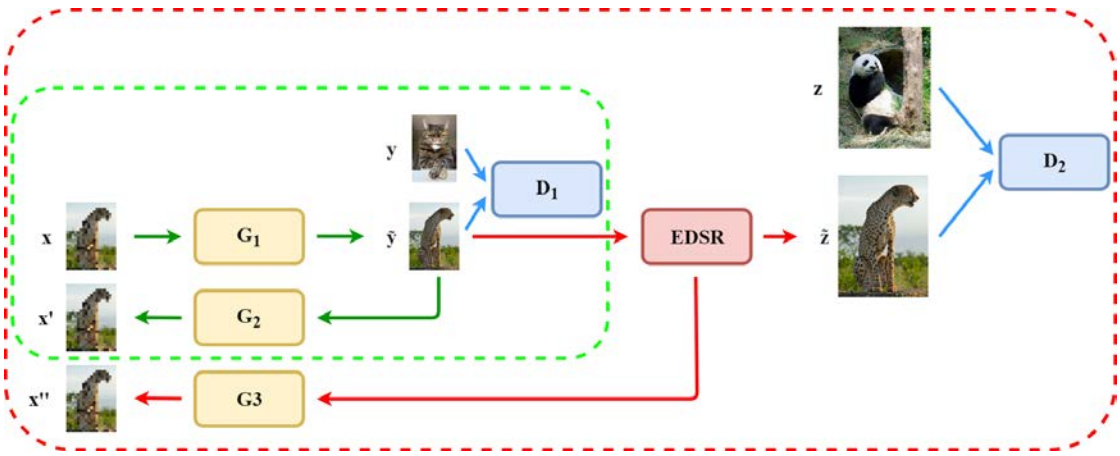


Figure 3.7: Cycle-in-CycleGAN model.

in a unsupervised way. As explained in Section 3.1, the task needs 3 convolutional neural networks:

- a generator G_1 to bring the image from the noisy low resolution domain to the noise-free low resolution domain;
- a second generator G_2 to perform the inverse task of the first one, in order to assure consistency between the input images and the produced images;
- a discriminator D_1 for adversarial training.

Then, it is necessary to investigate how to super-resolve the intermediate image $\tilde{\mathbf{y}}$ to the desired size. We stack an *EDSR* model, described in Section 3.2, directly after G_1. Then we use a discriminator D_2 in order to perform adversarial training and a generator G_3 to maintain consistency between the input $\mathbf{x}$ and the output $\tilde{\mathbf{z}}$.
The losses of the model are:

- the *generator adversarial loss*, used to perform adversarial training by fooling the discriminator D_2:

$$\mathcal{L}_{GAN}^{HR} = \frac{1}{N}\sum_{i=1}^{N} \|D_2(EDSR(G_1(\mathbf{x}_i))) - 1\|_2 \tag{3.8}$$

- the *cycle consistency loss*, used to maintain consistency between the input x and the output $\tilde{z}$

$$\mathcal{L}_{cyc}^{HR} = \frac{1}{N}\sum_{i=1}^{N} \|G_3(EDSR(G_1(\mathbf{x}_i))) - \mathbf{x}_i\|_2; \tag{3.9}$$

- the *identity loss*, to maintain color consistency between z and its super-resolved bicubic-downsampled version z'

$$\mathcal{L}_{idt}^{HR} = \frac{1}{N}\sum_{i=1}^{N} \|EDSR(\mathbf{z}_i') - \mathbf{z}_i\|_1 \tag{3.10}$$

- *total variation* loss, to impose spatial smoothness

$$\mathcal{L}_{TV}^{HR} = \frac{1}{N}\sum_{i=1}^{N} \Big(\|\nabla_h EDSR(G_1(\mathbf{x}_i))\|_2 + \|\nabla_w EDSR(G_1(\mathbf{x}_i))\|_2\Big). \tag{3.11}$$

These losses are combined in the weighted sum

$$\mathcal{L}_{total}^{HR} = \mathcal{L}_{GAN}^{HR} + \lambda_1 \mathcal{L}_{cyc}^{HR} + \lambda_2 \mathcal{L}_{idt}^{HR} + \lambda_3 \mathcal{L}_{TV}^{HR}, \tag{3.12}$$

where $\lambda_1, \lambda_2, \lambda_3$ are the weight parameters and it is minimized in the training procedure.
The discriminator is instead trained to distinguish between real high resolution samples and artificial Super-Resolution examples and therefore by minimizing the loss

$$\mathcal{L}_D^{HR} = \frac{1}{2}\left[\frac{1}{N}\sum_{i=1}^{N} \|D_2(\mathbf{z}_i) - 1\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|D_2(EDSR(G_1(\mathbf{x}_i)))\|_2\right]. \tag{3.13}$$

3.3.1 Networks' Architecture

As shown in Figure 3.7, the cycle-in-cycle structure employs a total of 6 CNNs. The inner cycle is identical to the one described in 3.1, while the outer one employs and EDSR model (as the one described in 3.2), a discriminator and a feedback generator.
The architecture of the discriminator is shown in Figure 3.8. It is identical to the architecture of the discriminator of the denoising block, Figure 3.4, except for the strides of the first 3 layers, which is set to 2. This modification does not change the number of trainable parameters, which remains 2768321. Similarly, the architecture of the feedback generator G_3 shown if Figure 3.9 is identical to the one of the generators employed in the denoising block, except for the filter sizes, set from 3 to 4, and the strides of the second and the third convolution blocks, which are set to 2 in order to perform a $\times 4$ downsampling and hence to bring the

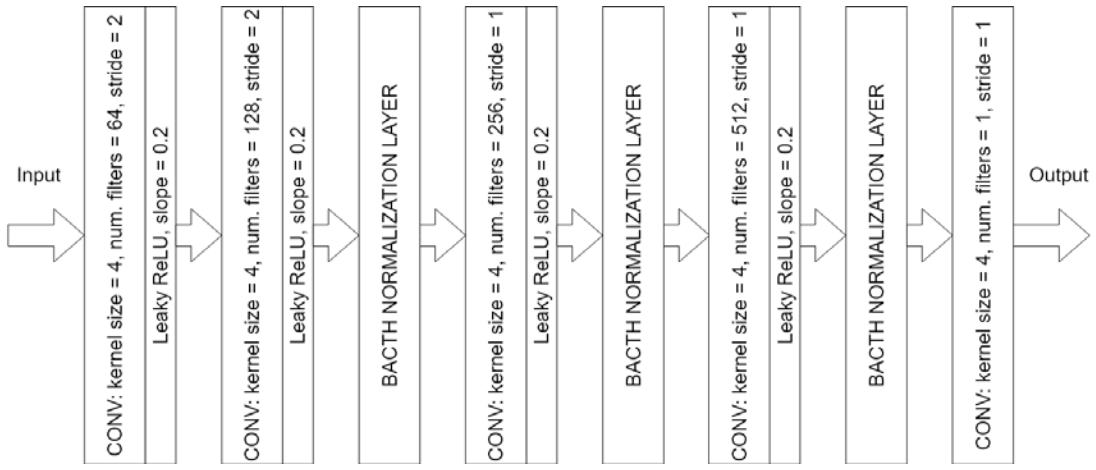


Figure 3.8: Architecture of the discriminative network of the CinCGAN model.

samples back to the original size. The total number of parameters to be optimized

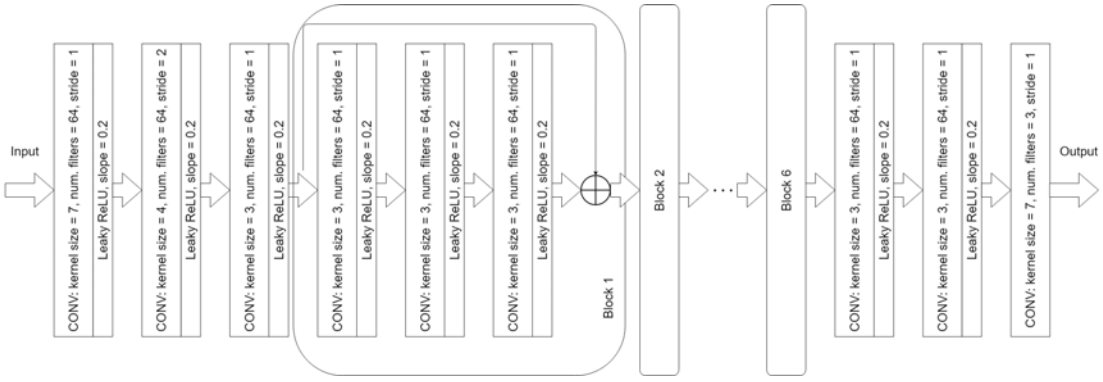


Figure 3.9: Architecture of the feedback network of the CinCGAN model.

in the whole model is 10671424. The training details are going to be explained in Chapter 4.

3.4 Enhanced Discriminator

We propose a novel architecture for the denoising cycle which aims to strengthen the feedback generator G_2. This first strategy aims to correct the behavior of G_2 by forcing it to produce samples closest to the ones in domain X without focusing too much on the removal of GAN artifacts. Since G_2 is originally trained just by a *cycle consistency loss* focusing on the cycle $X \to Y \to X$, we upgraded the loss in order to include also the cycle $Y \to X \to Y$ as

$$\mathcal{L}_{cyc}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|G_2(G_1(\mathbf{x}_i)) - \mathbf{x}_i\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|G_1(G_2(\mathbf{y}_i)) - \mathbf{y}_i\|_2. \tag{3.14}$$

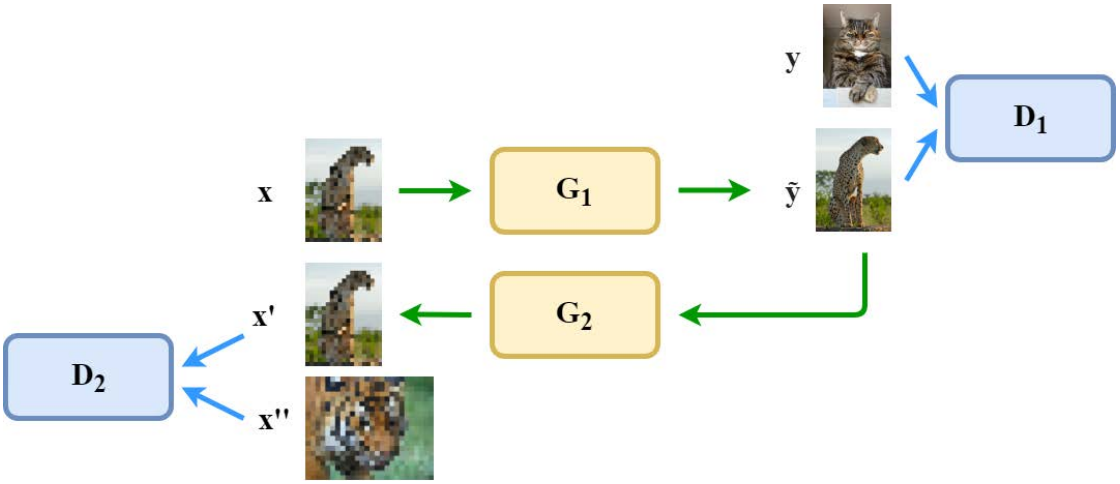


Figure 3.10: BD-GAN upgrades the baseline denoiser by adding a discriminator D_2 for the domain X. D_2 is used to perform adversarial training on G_2, bringing balance and stability to the whole structure.

Furthermore, we feed to the discriminator D_1 a stack of two images from the domains X and Y. In this setup, D_1 shall distinguish between *real* examples of the form $[G_2(\mathbf{y}); \mathbf{y}]$ from fake examples of the form $[\mathbf{x}; G_1(\mathbf{x})]$, $\mathbf{x} \in X$ and $\mathbf{y} \in Y$. This upgrade on the generator influences also the computation of the adversarial loss, which now sees also the impact of G_2, as

$$\mathcal{L}_{GAN}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|D_1([\mathbf{x}_i; G_1(\mathbf{x}_i)]) - 1\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|D_1([G_2(\mathbf{y}_i), \mathbf{y}_i])\|_2 \tag{3.15}$$

The loss of the discriminator becomes instead

$$\mathcal{L}_{D,2}^{LR} = \frac{1}{2}\left[\frac{1}{N}\sum_{i=1}^{N} \|D_1([G_2(\mathbf{y}_i); \mathbf{y}_i]) - 1\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|D_1([\mathbf{x}_i, G_1(\mathbf{x}_i)])\|_2\right]. \tag{3.16}$$

3.5 Bi-Directional GAN

This second strategy upgrades the CinCGAN denoiser by supporting it with an additional discriminator. In this way we build *BD-GAN*, a *Bi-Directional GAN* that shares the structure of the classic CycleGAN but uses different loss functions to perform his training for domain shift. The new discriminator D_2 is in charge of distinguishing between samples coming from the domain X and examples of the domain Y which have been degraded by G_2, as Figure 3.10 shows. This architectural change comes with some modifications with regards to the loss functions.

Now the *generator adversarial loss* has to be modified as

$$\mathcal{L}_{GAN}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|D_1(G_1(\mathbf{x}_i)) - 1\|_2 + \frac{1}{N}\sum_{i=1}^{N} \|D_2(G_2(\mathbf{y}_i)) - 1\|_2. \tag{3.17}$$

The parameters of G_1 are tuned to minimize the first term of the loss, while the second will be optimized by the parameters of G_2. The *cycle consistency loss* becomes the same as Equation 3.14, while the *identity loss* is modified as

$$\mathcal{L}_{idt}^{LR} = \frac{1}{N}\sum_{i=1}^{N} \|G_1(\mathbf{y}_i) - \mathbf{y}_i\|_1 + \frac{1}{N}\sum_{i=1}^{N} \|G_2(\mathbf{x}_i) - \mathbf{x}_i\|_1. \tag{3.18}$$

In this way, also G_2 is forced to maintain the color informations at his output.

3.6 EDSRv2

Driven by the need to reduce the training time of EDSR and to produce samples which better respect the color information of the input, we add to EDSR a network skip. We simply upsample the input and we add it at the output of the CNN, as Figure 3.11 shows.

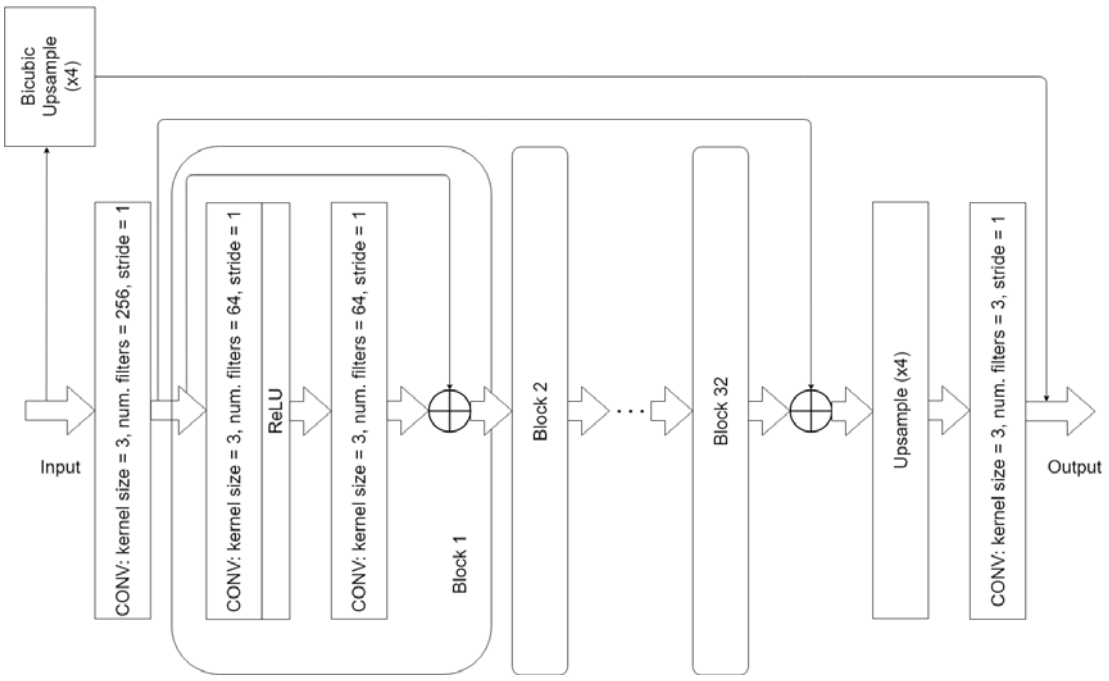


Figure 3.11: Proposed network skip on EDSR. The upsampled input is added to the output in order to force the network to correctly reproduce the input colors.

This setup, that is going to be referred as EDSRv2, performs better than the classic interpolation approaches and the implemented EDSR network in terms of PSNR and SSIM, as we are going to see in Chapter 5.

3.7 Joint Restoration and Super-Resolution

To jointly perform denoising and super-resolution on data degradated by unknown noise and blur operators, we chose to work on a model that differs from the one proposed by Yuan *et al.* [8].
Starting from the denoising BD-GAN model seen in the previous sections, we stack a Super Resolution module right after the denoiser G_1 and we add a discriminator in order to perform adversarial learning on the HR domain Z. Since the training is going to be unsupervised, we use a model to perform the domain shift $Z \rightarrow X$ which is going to be used in order to assure cycle-consistency between the input from the X domain and the output from the Z domain. We also employ a second discriminator on the X domain, having become aware of the superiority on the BD-GAN model with respect to the CinCGAN denoiser.
This model is represented in Figure 3.12. We choose EDSRv2 as super resolution

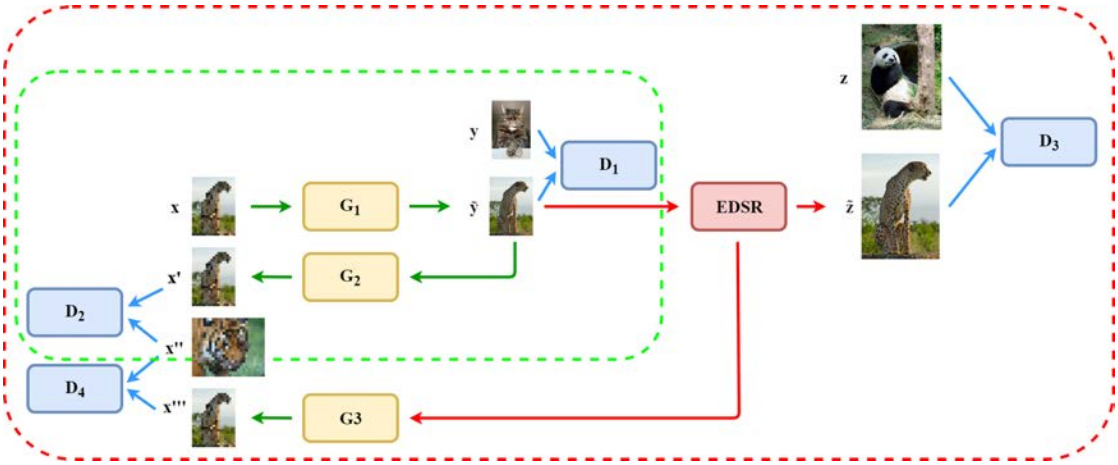


Figure 3.12: Proposed model for joint denoising and super-resolution.

module and we keep G_3 and D_3 as in Chapter 3.3. D_4 performs the same task of D_2 and thus we choose to employ the same architecture, i.e. the one described in Figure 3.4.

4

Training

This chapter is going to describe the dataset generation and to explain in detail the training procedures of the models presented in the previous chapters.
In Section 4.1 we are going to introduce DIV2K, the dataset chosen to train the models described before. It gathers 1000 images at 2K resolution, along with their downsampled and degraded copies. This dataset is used to build the training sets and the validation and test sets for the training of the proposed models.
Section 4.2 is going to reveal the details of the models presented in Chapter 3. In particular, we are going to see how the denoising cycle and the EDSR model are pre-trained and how jointly restoration and super-resolution work.

4.1 Training Data Generation

In order to train our models we choose DIV2K, the dataset used for the Super-Resolution challenges NTIRE (CVPR 2017 and 2018) and PRIM (ECCV 2018).
In [7] a novel dataset with DIVerse 2K (DIV2K) resolution high quality images is introduced. The dataset authors collected from the internet 1000 RGB color images. All the 1000 images are 2K resolution, that is they have 2000 pixels on at least one of the axes (vertical or horizontal). To ensure the diversity between the data, images have been collected among dozens of different websites. DIV2K covers a large diversity of contents, ranging from people, handmade objects and environments (cities, villages), to flora and fauna, and natural scenes, including underwater and dim light conditions.
Each image is then downsampled by a scale $\times 2$, $\times 3$ and $\times 4$ in two different tracks:

- Track 1, the first set of downsampled images, is produced using a known bicubic interpolation (function `imresize` of MATLAB). This set has been used to feed to the discriminative model *clean* examples.

- Track 2 is the result of unknown downsampling algorithm, blur and decima-

tion. This challenging subset is the one that has been used in the denoising experiments.

Furthermore, each track is split in Training set (80%), Validation set (10%) and Test set (10%). Since the ground truth was absent in the Test set for domain Z, we tested our models on the Validation set.
In order to train the Super-Resolution models described in Chapter 3, different sets of images had to be taken from the DIV2K Tracks:

- a set of 400 degraded and $\times 4$ downsampled images, called X *training set*, is used to feed the generator G_1.

- a set of 400 $\times 4$ downsampled images, called Y *training set*, is used to train the discriminator D_1. The discriminator D_1 is trained to distinguish between the images of this set and the examples coming from the X *training set* cleaned by the generator G_1. The images of this training set are not the same of the X *training set*, since we want unsupervised training.

- a set of 400 high resolution images (the same of Y *training set*), called Z *training set*, is used to train the discriminator D_2. D_2 has to distinguish between the samples of Z *training set* and samples enhanced by the generative branch and coming from the X *training set*.

Moreover, these sets are augmented through 90 and 270 degree rotation, and flipping, bringing the size of each training set from 400 to 1200 samples. If we also take into account the fact that the networks are trained with squared patches randomly cropped from the images of the training sets, we see that the size of the training sets grows further.

4.2 Training Details

The models presented in Chapter 3 have been implemented in Python 3.7.1 with the support of the Tensorflow 1.10 library. This Section will not linger on the code[1] and the implementation details, but it will focus instead in the training of the models, the choice of the parameters and the learning techniques.

[1] All the code can be found at `https://github.com/boemd/CinCGAN`.

4.2.1 Training of the Denoising Cycle

In order to optimize the parameters of the inner cycle we chose the Adam optimizer. Adam [3] is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Given a loss function $\mathcal{L}_t(\theta)$, where θ is the vector of optimization parameters and t is the time index, and given the learning rate α and the exponential decay rates $\beta_1, \beta_2 \in [0, 1)$, the parameters' updates at time t proceeds as follows:

- The gradients of the loss function at time t are computed as

$$g_t = \nabla_\theta f_t(\theta_{t-1}). \tag{4.1}$$

- The bias corrected first and second moment estimates are updated using moving average filters of the first order

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad \hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{4.2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{4.3}$$

- The weight are updated as

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{4.4}$$

The learning parameters have been chosen as $\beta_1 = 0.5$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The learning rate α is initialized as 2×10^{-4} and then decreased by a factor 2 every 40000. The filters of the convolutional layers are initialized using a normal distribution. The parameters used to weight the losses in Equation 3.5 are chosen as $w_1 = 10$, $w_2 = 5$ and $w_3 = 0.5$ after a process of hyperparameters tuning.
The model is trained for 400000 iterations. In each iteration a batch of 16 image patches of size $32 \times 32 \times 3$ are fed to the denoising cycle, and the losses are computed. Two optimizers using the Adam algorithm optimize the parameters of the generators G_1 and G_2 by respectively minimizing Equations 3.5 and 3.2, and another Adam optimizer optimizes the parameters of the discriminator D_1 by minimizing Equation 3.6. In this way, adversarial training is performed.

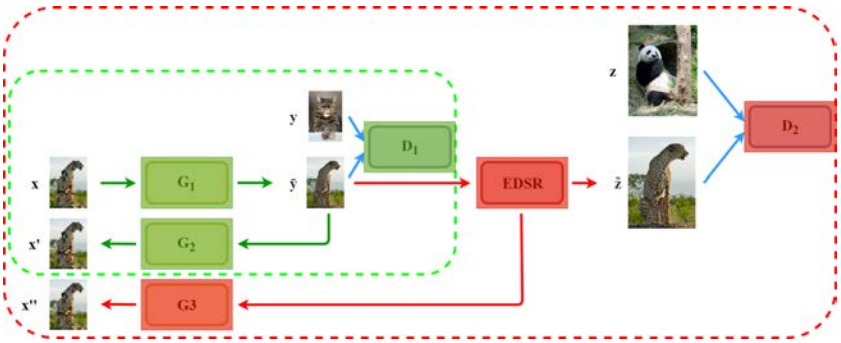
4.2.2 Training of EDSR

In order to train the EDSR network we randomly crop RGB patches of size $192 \times 192 \times 3$ from the *Z training set* and we downscale them by a factor 4. At this point we have $48 \times 48 \times 3$ patches to feed to the network and its ground truth, which will be used to compute the L1 norm with the produced examples. As before, the batch size is set as 16. The network is trained by an Adam optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The learning rate is initialized as 10^{-8} and then halved after every 2×10^5 iterations. The network is trained for 10^6 iterations.

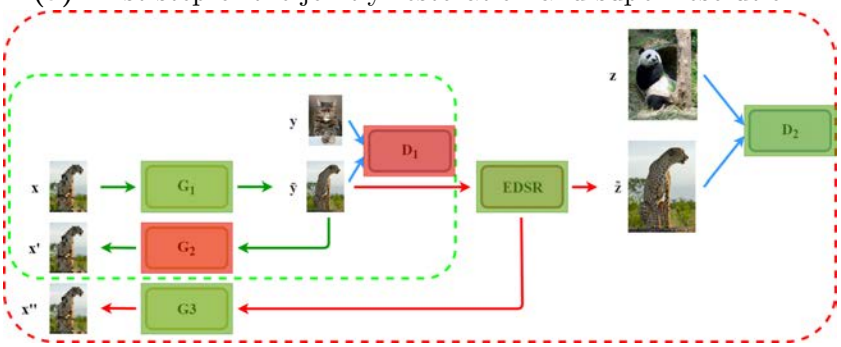
4.2.3 Training of CinCGAN

The training of the cycle-in-cycle model is a complex operation which has been divided in two steps. In the first step we firstly pre-train the denoising cycle and the EDSR network and then, in the final step, we jointly fine tune the whole cycle-in-cycle model.
In the fine tuning step we initialize G_1, G_2, D_1 and $EDSR$ with the weights computed before and then in each training iteration we firstly minimize Equation 3.5 and then Equation 3.12. This alternation is better represented in Figure 4.1: at first the networks of the outer cycle are frozen and $\mathcal{L}_{total}^{LR}$ is minimized as shown in the previous sections and then, after G_2 and D_1 are frozen, the loss $\mathcal{L}_{total}^{HR}$ gets minimized.

Adam optimizers are chosen as in Section 4.2.1 and the learning rate is set as 10^{-4}. The losses' weights are set as $w_1 = 10$, $w_2 = 1$, $w_3 = 0.5$, $\lambda_1 = 10$, $\lambda_2 = 5$ and $\lambda_3 = 2$, as the authors recommend in [8].



(a) First step of the jointly restoration and super-resolution.



(b) Second step of the jointly restoration and super-resolution.

Figure 4.1: Jointly restoration and super-resolution.

5

Results

In this section the results of the experiments performed starting from the models presented in Chapter 3 will be presented along with the metrics used to evaluate them.
The produced examples are evaluated by comparing them with their ground truth by means of *Peak Signal-to-Noise Ratio* and *Structural Similarity*.
The experiments have been performed on the denoising model, on the Super-Resolution module and on the cycle-in-cycle combination of the previous models.

5.1 Evaluation metrics

The images produced by the models described in Sections 3.1, 3.2 and 3.3 have been evaluated with two functions, *Peak Signal-to-Noise Ratio (PSNR)* and *Structural Similarity (SSIM)* [1]. Both functions evaluate a metric between an artificial image and its ground truth.
The PSNR is the ratio, usually expressed in decibel, between the maximum possible power of a signal and the power of the noise that corrupts the the quality of the image. The power of the noise between a color image I_x and its ground truth I_y is often expressed in terms of Mean Squared Error

$$MSE(I_x, I_y) = \frac{1}{3MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{c=0}^{2} [I_x(m,n,c) - I_y(m,n,c)]^2 \tag{5.1}$$

where $(M, N, 3)$ is the shape of the images and $I(m, n, c)$ indicates the value of the intensity of the pixel (m, n) and color channel c for the image I.
When the mean squared error is known, the peak signal-to-noise ratio is expressed as

$$PSNR(I, I_{gt}) = 10 \log_{10} \left(\frac{I_{max}^2}{MSE(I, I_{gt})} \right) \tag{5.2}$$

$$= 20 \log_{10} \left(I_{max} \right) - 10 \log_{10} \left(MSE(I, I_{gt}) \right) \tag{5.3}$$

where I_{max} is the maximum value that the images can take, for example 255 if the value of each channel of a pixel is represented with an unsigned 8-bits integer.
Usually the peak signal-to-noise ratio is used to measure the quality of the reconstruction of lossy compression algorithms, but in our scenario it is useful to measure how the artificial image is close to the ground truth.
The PSNR is an appealing metric, since it is simple to compute, has a clear physical meaning and it is mathematically convenient when optimizing, since the minimization of the MSE is equivalent to the maximization of the PSNR. However, PSNR is not a good measure to perceive visual quality. The Structural Similarity is another measure for the similarity of two images but, differently from the PSNR, the SSIM index does not measure absolute differences between intensity levels. SSIM takes into account perceptual phenomena such as luminance, contrast and structure, as explained in [1].

An example of the difference between PSNR and SSIM can be seen in Figure 5.1.

(a) I_g: image "0810" from DIV2K blurred by a gaussian filter with $\sigma^2 = 0.5$.

(b) I_a: dummy alteration on the corners of the blurred image I_g.

Figure 5.1: Alteration of Image "0810" from DIV2K. A small change in the image causes a huge loss on the PSNR, while the SSIM remains almost unchanged.

We have taken an image from DIV2K, namely I_0, and blurred it with a gaussian filter, producing I_g (Figure 5.1a). The PSNR and SSIM between I_0 and I_g are $44.38dB$ and 0.9915. If we alter the corners of I_g, as in image I_a (Figure 5.1b), we

observe a drop of $8dB$ in the PSNR, which falls to 35.98, while the SSIM remains almost unchanged at 0.9898.
This dummy example shows the how the PSNR metric is inadequate in measuring the visual quality of an image, while the SSIM behaves better in this task.

The SSIM index is the result of the combination of 3 components, i.e.

$$SSIM(I_x, I_y) = l(I_x, I_y) \cdot c(I_x, I_y) \cdot s(I_x, I_y), \tag{5.4}$$

which respectively compare luminance, contrast and structure. The function $l(I_x, I_y)$ compares the luminance of the two images and it is defined as

$$l(I_x, I_y) = \frac{(2\mu_x\mu_y + c_1)}{(\mu_x^2 + \mu_y^2 + c_1)}, \tag{5.5}$$

where:

- μ_x, μ_y are the averages of I_x and I_y, i.e. the mean intensities;
- $c_1 = (k_1 I_{max})^2$;
- $k_1 = 0.01$ and is fixed.

It assumes values in the interval $[0, 1]$ and touches his maximum when the input images share the same mean, i.e. when $\mu_x = \mu_y$.
The contrast comparison function $c(I_x, I_y)$ takes a similar form:

$$c(I_x, I_y) = \frac{(2\sigma_x\sigma_y + c_2)}{(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{5.6}$$

where:

- σ_x^2, σ_y^2 are the variances of I_x and I_y, which estimates the squares of the signals' contrast;
- $c_2 = (k_2 I_{max})^2$;
- $k_2 = 0.03$ and is fixed.

The structure comparison function instead uses the correlation (inner product) between I_x and I_y as a simple and effective measure to quantify the structural similarity:

$$s(I_x, I_y) = \frac{(2\sigma_{x,y} + c_2)}{(2\sigma_x\sigma_y + c_2)}, \tag{5.7}$$

where $\sigma^2_{x,y}$ is the covariance of I_x and I_y.
The *Structural Similarity* is hence defined as

$$SSIM(I_x, I_y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.8}$$

To evaluate the experiments, we chose the MATLAB scoring functions [1] used in *NITRE 2018 challenge* for the evaluation of the proposed solutions. The PSNR/SSIM validation curves have been computed using a Python transposition of those MATLAB scripts.

5.2 Denoising Experiments

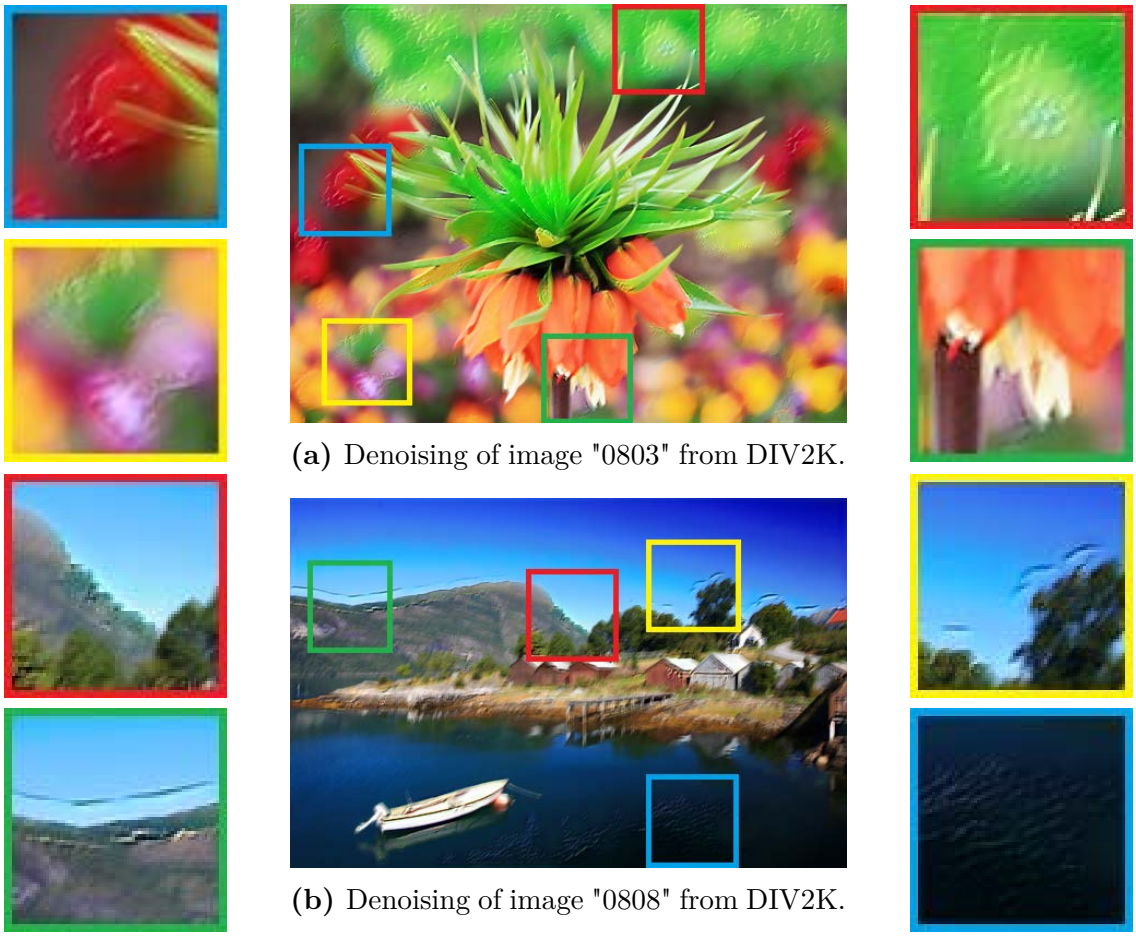
In this section we are going to report some of the tests and experiments that have been carried out using the CinCGAN framework and its sub-components.

5.2.1 Baseline Denoising

The first experiments that have been performed involve the inner cycle of the CinCGAN model, which has been presented in Chapter 3.1. As it was explained, this module performs unsupervised denoising of low resolution image patches by using a quasi-CycleGAN structure (Figure 3.2) and exploiting some ideal characteristics of the cleaned LR images during the training phase, such as cycle consistency, spatial smoothness and color preservation.
The first experiments, as depicted in Chapter 4.2.1, did not give good results both in terms of evaluation metrics and visual perception, as it is possible to see in Figure 5.2.

From the training statistics and the learning curves it was possible to see that that there were no problems in the adversarial part of the training. The discriminator is trained to output the value 0.9 when the example is recognized as *real*. When the adversarial training is balanced and at regime, the discriminator should output $\frac{1}{2} \times 0.9 = 0.45$ and hence its loss should float around $0.45^2 \approx 0.2$, since we are using a least squares GAN. From the plot of the discriminator loss, red line of Figure 5.3, it is possible to see that we almost fall on this case.

[1] MATLAB Scoring functions: `https://competitions.codalab.org/my/datasets/download/ebe960d8-0ec8-4846-a1a2-7c4a586a7378`



(a) Denoising of image "0803" from DIV2K.

(b) Denoising of image "0808" from DIV2K.

Figure 5.2: Two of the examples generated by the denoising cycle during the first experiments containing the largest number of artifacts.

The average value of the loss is a bit lower than the desired one. This behavior is justified, since the task of the discriminator D_1 is simpler than the one of the generator G_1.

The most evident issue in this configuration is the vast presence of artifacts in the output images. This kind of artifacts is a well known problem in the field of adversarial learning and we tried to limit their presence with a simple modification of the total loss. We added a fourth weight on the GAN component of Equation 3.5 in order to better control the impact of the adversarial loss on the total one. The resulting loss is

$$\mathcal{L}_{total}^{LR} = w_0\mathcal{L}_{GAN}^{LR} + w_1\mathcal{L}_{cyc}^{LR} + w_2\mathcal{L}_{idt}^{LR} + w_3\mathcal{L}_{TV}^{LR}. \tag{5.9}$$

We tuned the new parameter w_0 by trying different values and observing the trend of the training losses and metrics.

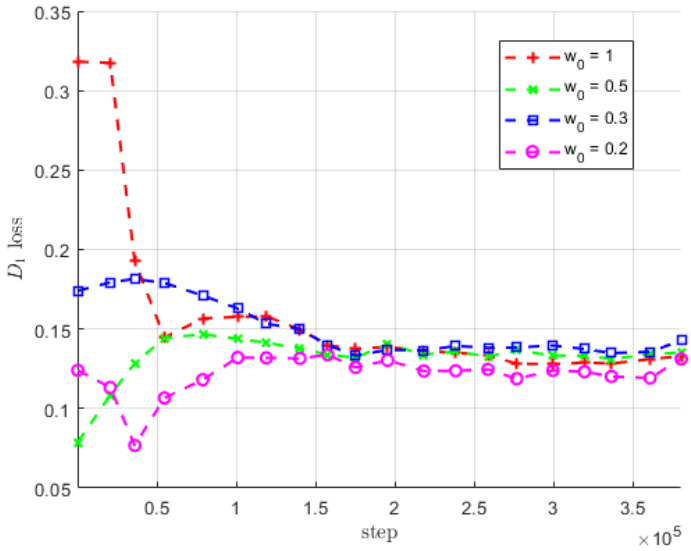


Figure 5.3: Discriminator losses while performing parameter tuning on w_0.

In Figures 5.3 and 5.4 the values of the discriminator's loss and the PSNR measured during training are represented for some values of the parameter w_0. When values higher than 1 or lower than 0.2 are assigned to w_0, the training degenerates. From Figure 5.3 we see that the adversarial learning is quite stable on all the scenarios.

From the comparisons of the PSNRs obtained by tuning w_0, we can see that the highest PSNR is given by the lowest parameter, i.e. 0.2. This setup allows us to obtain an average PSNR of $22.88dB$ and SSIM of 0.6912 on the validation set, along with a rarefication of the adversarial artifacts, as Figure 5.5 shows.

Despite the gain in the metrics value, this approach did not led to an acceptable solution because of 3 reasons:

- the PSNR and SSIM values were still too low, in comparison to the ones of the noisy input images (24.07/0.7180);
- some artifacts were still present;
- the generator G_2 did not learn the expected task, i.e. to map LR images to noisy-LR images, but it learnt to remove the artifacts due to the adversarial training. This is evident in Figure 5.6. When we feed to G_2 an image denoised by G_1 (Figure 5.6a) it learns to correct the GAN artifacts, as Figure 5.6c shows. When instead we try to shift an image from the clean LR domain to the noisy LR domain, we see in Figure 5.6d that G_2 tries to remove some non-existing artifacts, resulting in degrading the image in a way that is different from the expected one.

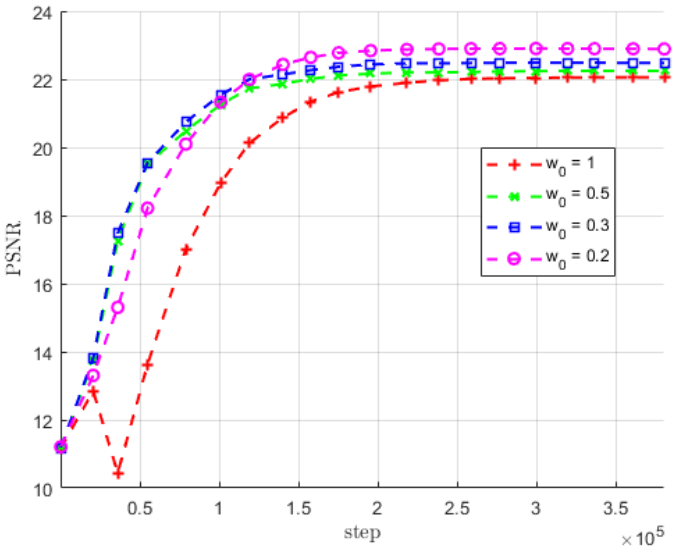


Figure 5.4: PSNR comparison on the tuning of the parameter w_0.

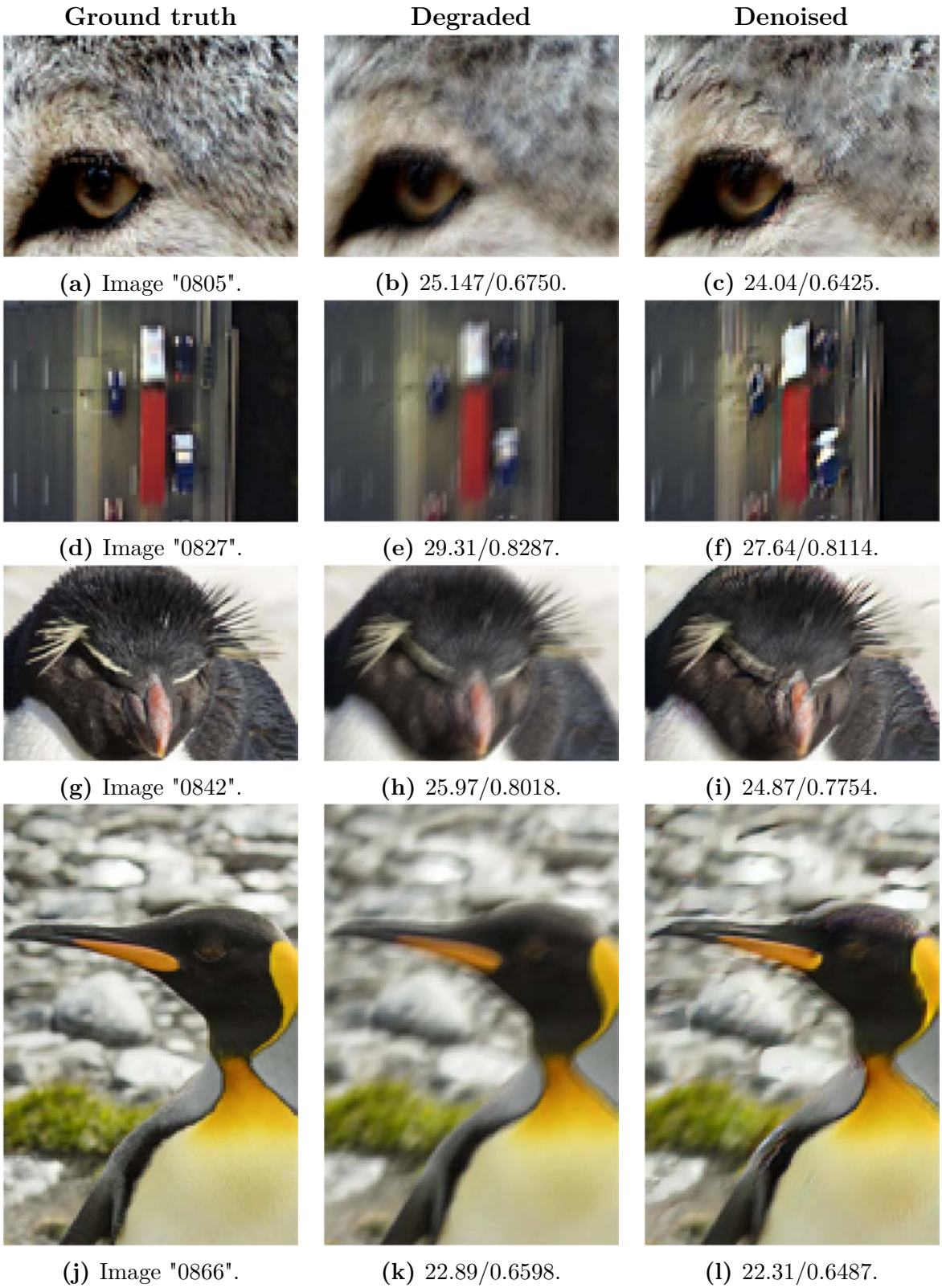
To solve this issue, it is necessary to further reduce the artifacts produced by G_1 and improve the performances of G_2. Two solutions have been thought. In the first one, no architectural modification is added to the framework. We just work on the loss functions and on the input of the discriminator. The second solution establishes a major update, since a new discriminator is added.

5.2.2 Enhanced Discriminator Denoising

This first strategy aims to correct the behavior of G_2 by forcing it to produce samples closest to the ones in domain X without focusing too much on the removal of GAN artifacts. We tested this approach, presented in Section 3.4 on the usual dataset, obtaining the prefixed results. As Figure 5.7 shows, G_2 has stopped to produce artifacts and started to perform the domain shift $Y \rightarrow X$.

5.2.3 BD-GAN

This second strategy, presented in Section 3.5, upgrades the CinCGAN denoiser by supporting it with an additional discriminator. In this way we build *BD-GAN*, a *Bi-Directional GAN* that shares the structure of the classic CycleGAN but uses different loss functions to perform his training for domain shift. The new discriminator D_2 is in charge of distinguishing between samples coming from the domain X and examples of the domain Y which have been degraded by G_2. In



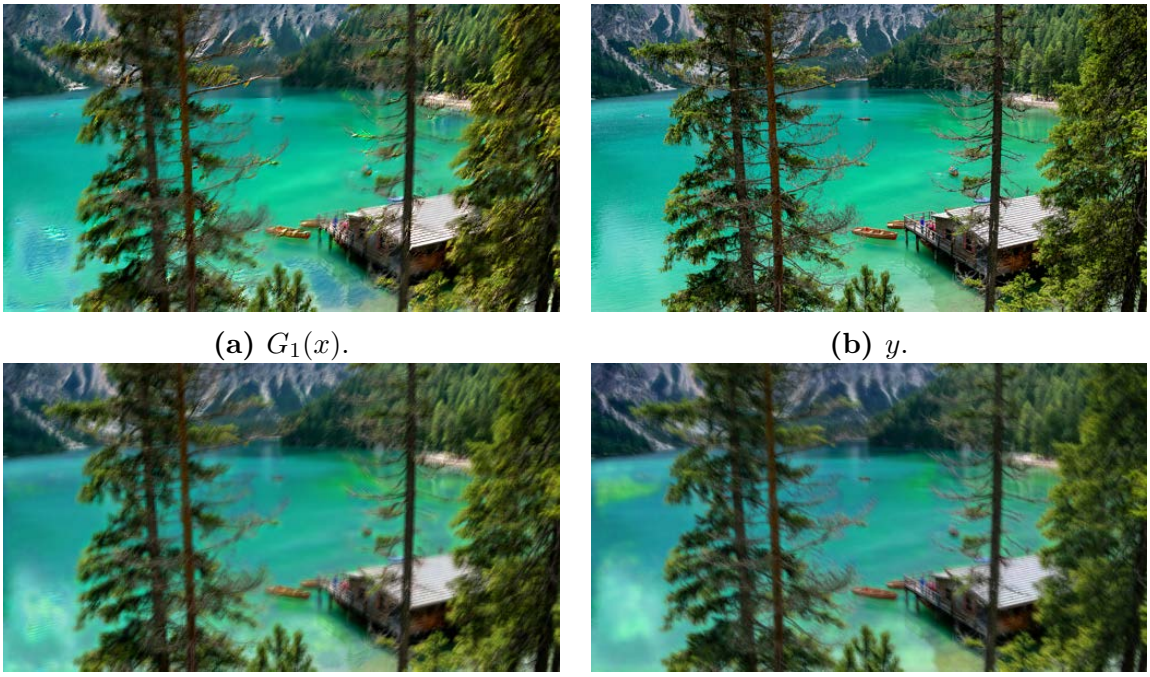
(a) Image "0805". **(b)** 25.147/0.6750. **(c)** 24.04/0.6425.

(d) Image "0827". **(e)** 29.31/0.8287. **(f)** 27.64/0.8114.

(g) Image "0842". **(h)** 25.97/0.8018. **(i)** 24.87/0.7754.

(j) Image "0866". **(k)** 22.89/0.6598. **(l)** 22.31/0.6487.

Figure 5.5: Examples of baseline denoising with a reduced impact of the *adversarial loss*.



(a) $G_1(x)$. **(b)** y. **(c)** $G_2(G_1(x))$ **(d)** $G_2(y)$.

Figure 5.6: Example of misfunctioning of G_2 on the image 807 of the validation split. We refer as **x** to the blurred and degraded image and to **y** to its clean ground truth.

this way, also G_2 is forced to maintain the color informations at his output.

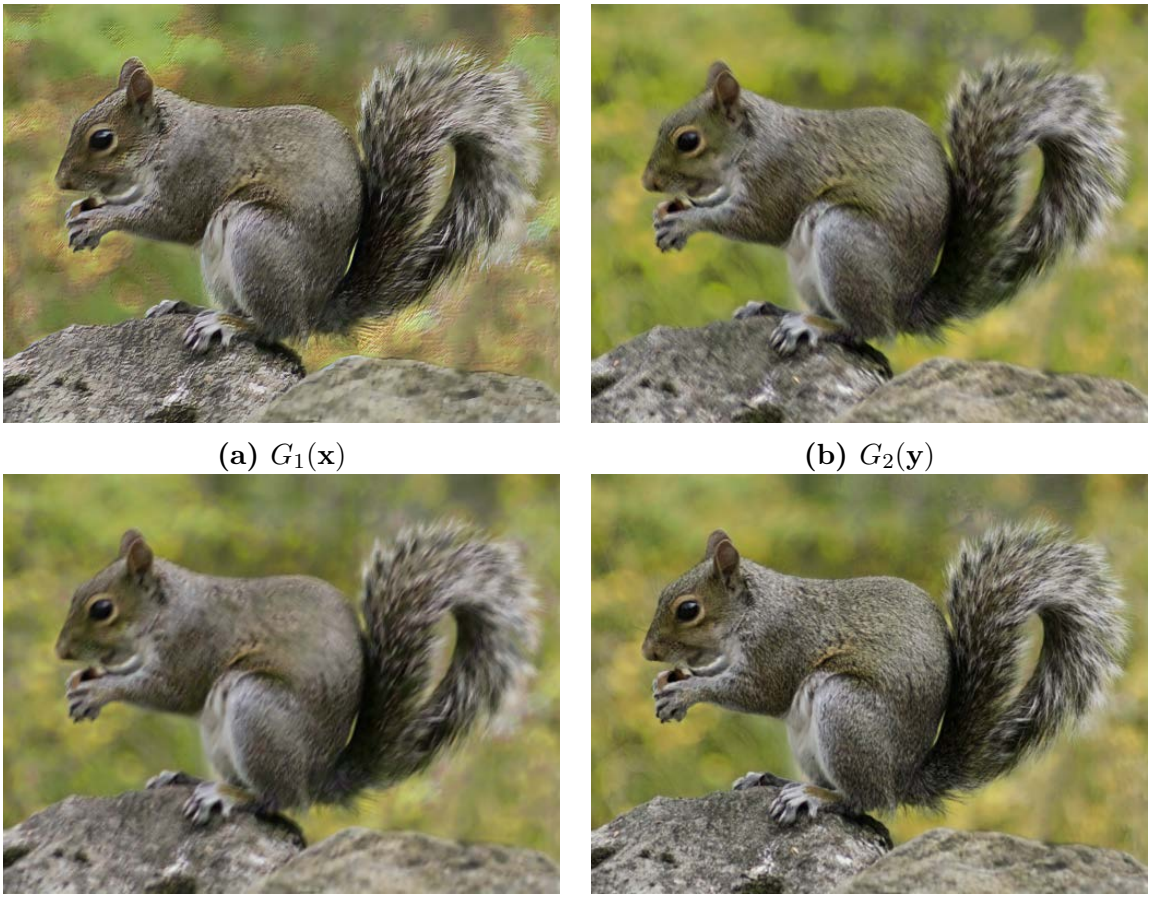
5.2.4 Denoising comparisons

The implemented denoising strategies have been compared both in terms of evaluation metrics and visual perception. The values of PSNR and SSIM are collected in Table 5.1, where we can see that the two new solutions outperform the baseline at least in terms of SSIM.

method	PSNR (dB)	SSIM
Baseline	22.85	0.6854
Baseline, $w_0 = 0.2$	22.88	0.6912
Enhanced discriminator	22.54	0.7183
BD-GAN	23.26	0.6929

Table 5.1: Comparisons between the implemented denoising techniques in terms of evaluation metrics PSNR and SSIM.

By forcing the discriminator to work on a stack of *noisy-clean* images, we obtain a large increment on the Structural Similarity, but we are not able to minimize the presence of adversarial artifacts, as we can see in Figure 5.8e.



(a) $G_1(\mathbf{x})$ **(b)** $G_2(\mathbf{y})$

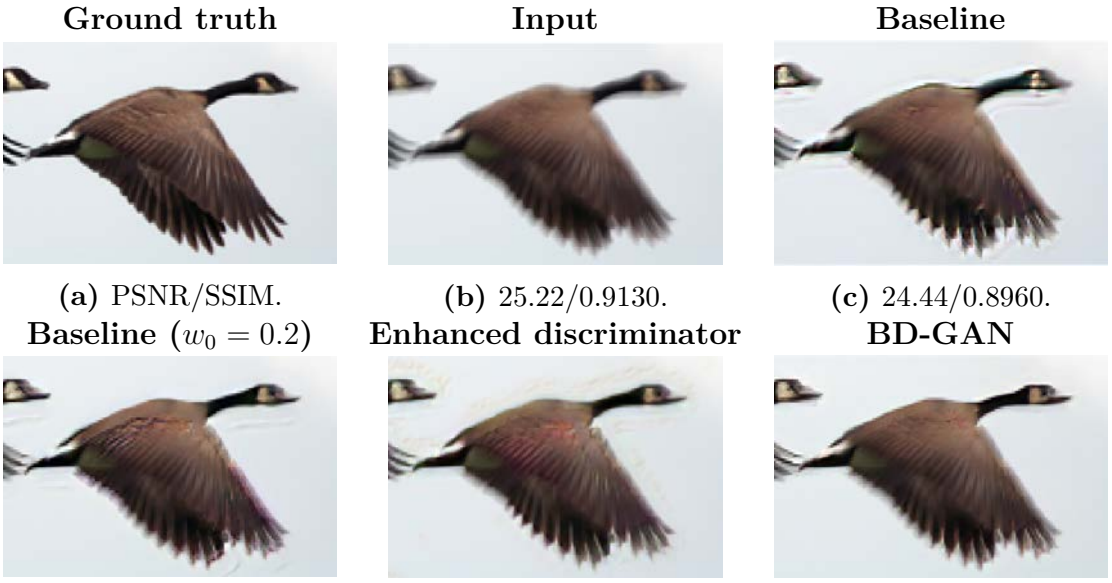
(c) $G_2(G_1(\mathbf{x}))$ **(d)** $G_1(G_2(\mathbf{y}))$

Figure 5.7: Action of the CNNs G_1 and G_2 on the image "0810" of DIV2K dataset. We denote the blurred and degraded image as $\mathbf{x}$ and the clean ground truth as $\mathbf{y}$.

By adopting a BD-GAN to perform denoising, we improve both the evaluation metrics and we reduce the amount of artifacts introduced in the cleaned images. For this reasons we decided to implement this model also in the SR step.

5.3 Super-Resolution Experiments

The SR-module has been pre-trained in a supervised way, as explained in Chapter 4.2.1, by downsampling HR images with a scale factor equal to 4 with a bicubic downsampling and feeding them to the CNN. Since we computed a metric between the reconstructed images and their ground truth, the training cannot be labeled as *unsupervised*. However, we did not need any HR-LR pair from any dataset, since we generated them by ourselves.



(a) PSNR/SSIM. **(b)** 25.22/0.9130. **(c)** 24.44/0.8960.

(d) 24.68/0.9040. **(e)** 24.31/0.8950. **(f)** 24.86/0.9120.

Figure 5.8: Comparison of the implemented denoising strategies with ground truth and input in Image "0896" of DIV2K.

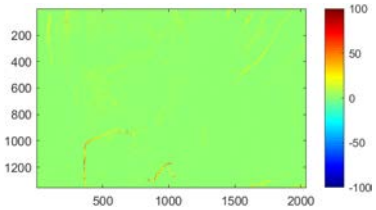
With this setup we trained a network able to obtain an average PSNR of 25.98 and an average SSIM of 0.7643. These results are lower than the ones obtained with the classical interpolation techniques because of a small color shift on the green channel that the CNN was not able to correct. Figure 5.9 shows an example of SR performed with EDSR, along with the colormaps of the error in the RGB channels. It is possible to deduce both from visual inspection and from MSE analysis that the error is gathered in the green channel.

Visual inspection shows also that EDSR can better super-resolve the images with respect to the classic interpolation methods. Figure 5.10 shows some comparisons between HR images, EDSR super-resolution and bilinear and bicubic upsampling. EDSR boasts a greater capability of reconstructing the edges and reproducing textures, such as animal fur in Figures 5.10f and 5.11p, avoiding the blocking artifacts present in the upsamplings.

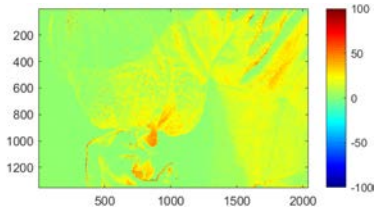
An approach to solve this color problem stands in slightly modifying the EDSR architecture by upsampling the input and then adding it at the end of the network's operations, as depicted in Section 3.6. EDSRv2 performs better than the classic interpolation approaches and the implemented EDSR network in terms of PSNR and SSIM, as Table 5.2 shows.

(a) Image "0838" from DIV2K Dataset [7]. PSNR/SSIM.

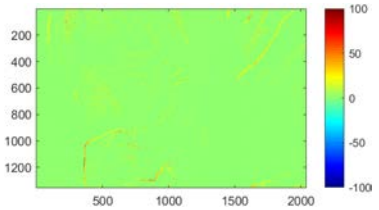
(b) EDSR reconstruction of image 838. 28.70/0.8536.



(c) R channel. MSE: 7.14.



(d) G channel. MSE: 110.43.



(e) B channel. MSE: 9.24.

Figure 5.9: Example of color failure in the reconstrucion of Image 0838 (DIV2K). The colormaps show the RGB split of the difference of the ground truth and the SR result. We see that the green channel is the one deviating from the ground truth.

method	bilinear	bicubic	EDSR	EDSRv2
PSNR	26.02	26.59	25.80	27.27
SSIM	0.7323	0.7521	0.7517	0.7804

Table 5.2: PSNR and SSIM comparisons of the implemented SR techniques with the classic interpolation methods.

5.4 Joint Denoising and Super-Resolution Experiments

To jointly perform denoising and super-resolution on data degradated by unknown noise and blur operators, we chose to work on a model that differs from the one proposed by Yuan *et al.* [8]. We employ the model presented in Section 3.7 and we divide each training iteration in two steps:

1. In the first step we update the denoising module, i.e the networks G_1, G_2, D_1 and D_2, by minimizing the losses seen in Section 5.2.3.

2. The second step sees the optimization of G_1, G_3, $EDSR$, D_3 and D_4.

From the results of table 5.3 we firstly see that when we increase the resolution

method	PSNR	SSIM
bicubic	21.75	0.5737
EDSRv2	21.56	0.5643
BD-GAN + bicubic	21.32	0.5599
BD-GAN + EDSRv2	19.79	0.5141
Joint denoising and SR	20.60	0.5444

Table 5.3: Comparisons between the implemented SR techniques in terms of evaluation metrics PSNR and SSIM.

of a noisy input, the classic bicubic upsampling works better than EDSRv2. At first, this result may seem counter-intuitive since we saw in Table 5.2 that EDSRv2 performed better Super-Resolution than the upsampling. This is one of the typical problems of Machine Learning: a model tested on a distribution different than the one it was trained on, sees a dramatic decrease on the performance. In our case, EDSRv2 was trained on clean input data coming from domain Y and tested on domain X. This shift in the input domain causes the PSNR to lose $\approx 5.5dB$.
When we directly apply EDSRv2 to data denoised by BD-GAN, we face a similar problem: the bicubic interpolation still outperforms our CNN. This happens because BD-GAN has still some issues and doesn't actually perform the $X \to Y$ with sufficient accuracy, so the input domain is not adequate.
When instead we jointly fine-tune the BD-GAN denoiser and EDSRv2 super-resolutor as depicted in Figure 3.12, we improve the PSNR of the produced examples of $\approx 1dB$.
In Figure 5.11 we can compare the reconstruction techniques implemented in this thesis. We see that while keeping an high PSNR, the simple SR on noisy data does not produce samples with acceptable visual quality. In particular, this approach amplifies the blur and the degradation of the input images, as it was reasonable to expect. When we stack BD-GAN and EDSRv2, we improve the visual quality but we also magnify the artifacts produced by the denoiser, if they are present. This issue induces a loss on the evaluation metrics which is partially recovered when jointly tuning the denoiser and the SR module, as we can see in Figures 5.11e and 5.11j. The joint optimization of the two modules not only performs SR on the denoised samples, but also acts as an artifacts mitigator, as we can see in the last example: the artifacts produced by BD-GAN and amplified by EDSRv2, Figure 5.11s, are not present in the output of the joint optimization, Figure 5.11t.
However, much room for improvement is still present both in the detail reconstruction and in the color preservation, as we can see in Figure 5.11r.

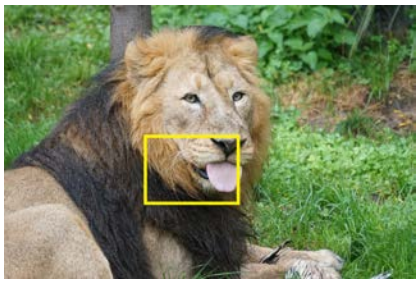
(a) Image "0803 from DIV2K Dataset [7], validation split.

(b) GT: PSNR/SSIM

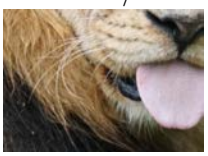
(d) BIL: 29.63/0.8454

(c) EDSR: 24.63/0.9202

(e) BIC: 30.83/0.8658



(f) Image "0809" from DIV2K Dataset [7], validation split.

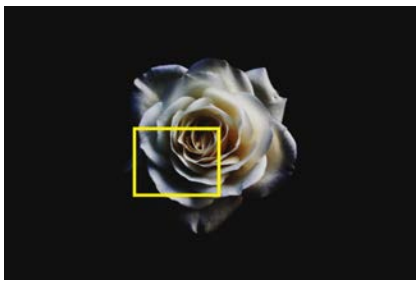


(g) GT: PSNR/SSIM

(i) BIL: 29.10/0.7698

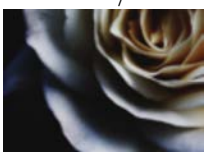
(h) EDSR: 29.47/0.6980

(j) BIC: 30.02/0.8011



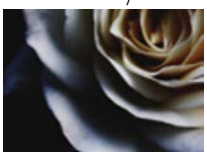
(k) Image "0843" from DIV2K Dataset [7], validation split.

(l) GT: PSNR/SSIM

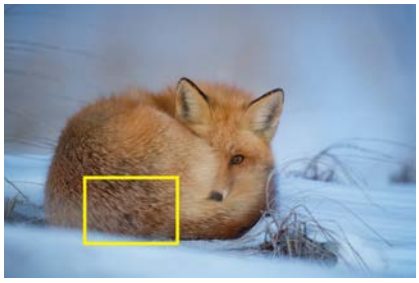


(n) BIL: 26.72/0.8618

(m) EDSR: 38.660/0.9726



(o) BIC: 27.43/0.8704



(p) Image "0862" from DIV2K Dataset [7], validation split.

(q) GT: PSNR/SSIM

(s) BIL: 27.45/0.6034

(r) EDSR: 31.59/0.8008

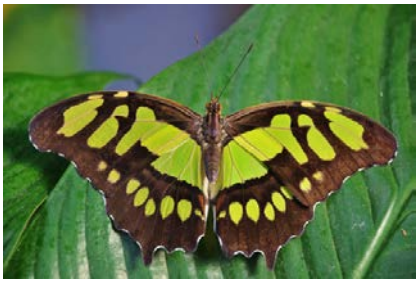
(t) BIC: 27.96/0.6389

Figure 5.10: Comparison between a HR patch (GT), its EDSR reconstruction (EDSR) and its bilinear (BIL) and bicubic (BIC) interpolation.

(a) Image "0803 from DIV2K Dataset [7], validation split.

(f) Image "0810" from DIV2K Dataset [7], validation split.

(k) Image "0816" from DIV2K Dataset [7], validation split.



(p) Image "0829" from DIV2K Dataset [7], validation split.

(b) GT: PSNR/SSIM

(d) BD+SR: 23.03/0.7680

(c) EDSRv2: 26.09/0.8540

(e) JOINT: 22.28/0.8142

(g) GT: PSNR/SSIM

(i) BD+SR: 20.89/0.4680

(h) SR: 22.720/0.5750

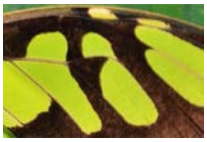
(j) JOINT: 21.77/0.5480

(l) GT: PSNR/SSIM

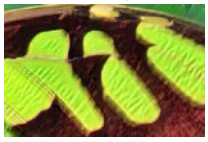
(n) BD+SR: 22.49/0.5781

(m) SR: 24.47/0.6300

(o) JOINT: 21.08/0.575



(q) GT: PSNR/SSIM



(s) BD+SR: 20.59/0.3490

(r) SR: 22.94/0.4040

(t) JOINT: 20.65/0.3800

Figure 5.11: Comparison between a HR patch (GT), the EDSRv2 Super-Resolution of the noisy input (SR), BD-GAN + EDSRv2 of the noisy input (BD+SR) and the joint restoration and Super-Resolution of the input (JOINT).

6

Conclusions

In this thesis I faced the problem of single-image super-resolution, i.e. the task of generating a high-resolution image from a low-resolution one. I chose a dataset where LR images were affected by blur, degradation and random shift of unknown nature. Furthermore, LR-HR resolution pairs of images were not available at training time, causing the need of an unsupervised kind of training.
Three image domains had hence to be taken in consideration:

1. Domain X of low resolution images affected by blur, noise and random shift.
2. Domain Y that included *clean* low resolution images.
3. Domain Z containing *clean* images at high resolution.

Given the complexity of the task, he problem had to be divided in sub-tasks:

- The first task consisted in learning a model to perform the shift $X \rightarrow Y$, i.e. to denoise images at low resolution.
- For the second task it was necessary to increase the resolution of clean LR images, i.e. to learn a model able to bring images from domain Y to domain Z.
- The final task consisted in jointly tune the previous models in order to perform the domain shift $X \rightarrow Z$.

To accomplish these tasks we chose to rely on Generative Adversarial Networks, more in particular on CycleGAN, a powerful framework for unsupervised image-to-image translation, and CinCGAN, a recent technique for unsupervised single-image super-resolution of noisy images.
The denoising of LR images was the most challenging of the sub-tasks in terms of model stability and output quality and required the investigation of different models. At first we implemented the CinCGAN denoiser, consisting in a generative model G_1 paired to a discriminator D_1 used to perform adversarial learning.

In order to perform the task in an unsupervised way, another generative model G_3 performing the inverse mapping of G_1 was added. The inverse mapping sets a harder constraint by forcing $G_2(G_1(\mathbf{x})) \approx \mathbf{x}$, the so-called *cycle consistency*. To improve the mediocre performances of this setup, we taught of two alternatives. We firstly tried to feed the discriminator with a stack of a two images from the domain X and Y. This allowed to perform adversarial training also on G_2, improving the reliability of its output.
Then we tried to bring balance to the model by adding a second discriminator and upgrading the *identity loss* and the *cycle-consistency loss*, developing *BD-GAN*.
These two extentions to the CinCGAN denoiser, in particular BD-GAN, both improved the quality and the metrics of the produced examples by reducing the presence of artifacts and avoiding color shifts on the outputs.
In order to implement the super-resolution module, we slightly modified the EDSR model in order to reduce the training time and improve the reliability of the color information of the produced examples. We included a network skip in the EDSR architecture by upsampling the input image and adding it to the output.
This solution helped us to obtain a better image quality in terms of accuracy metrics with respect to the classic upsampling methods and also reduced the training time to the 20% of the time required to train classic EDSR from scratch.
In order to accomplish the final task, we chose to employ the BD-GAN denoiser and stacking the SR module to it. We also included an inverse mapping to maintain cycle-consistency between the input and the output and two discriminators for adversarial training. This setup saw the joint optimization of 8 neural networks and was quite challenging to stabilize.
We compared its performance with just the stack of the denoiser and the SR module and we verified an improvement in the evaluation metrics.
As regards future works, it is necessary to further improve the denoiser, since it is the starting point of the whole process. BD-GAN needs to improve both perceptually and in terms of accuracy metrics and these improvements may be achieved by exploring new kinds of loss functions to adjust or substitute the ones exploited in this thesis. In particular it could be interesting to work on a new color-preserving loss, since the *identity loss* implicitly enforces $G(\mathbf{x}) \approx x$.
It could be also beneficial to slim down the overall architecture, since the largest issues we had involved the stabilization of the model.
In the end, we could also address the multiscale Super-Resolution problem by adopting a multiscale SR module, for example MDSR.

Ringraziamenti

Vorrei ringraziare in primis il prof. Pietro Zanuttigh, relatore di questa tesi di laurea, per la sua disponibilità nell'accogliermi nel suo laboratorio di ricerca negli ultimi 7 mesi, per i preziosissimi consigli e per il grande supporto fornitomi.
Sono particolarmente grato ai miei correlatori, Gianluca Agresti e Umberto Michieli, per il loro sostegno e per le critiche e la motivazione datemi nel corso di questi mesi.
Ringrazio inoltre i miei colleghi Luca Rossi, Marco Toldo e Mazen Mel, per gli scambi di idee, i consigli e le numerose pause caffè.

Un ringraziamento va ai miei amici, in particolare *O.P.D.I.D.P.D.O.*[1], *Il mio viaggio a Betlemme*, *Meme Godz*, *C.C.C.P.*, *Machine Faybio* ed il *Team Carrà*.

Ringrazio Giada per questo ultimo anno assieme.

In fine ringrazio tutta la mia famiglia, in particolare i miei genitori Antonella e Maurizio e mio fratello Andrea, per avermi supportato, sopportato, finanziato ed ospitato. Non avrei raggiunto questo traguardo senza di voi. Grazie.

[1] *Oggi più di ieri, domani più di oggi.*

Bibliography

[1] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *Image Quality Assessment: From Error Visibility to Structural Similarity*, IEEE Transactions on Image Processing, vol. 13, no. 4, April 2004.

[2] Ian J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, *Generative Adversarial Networks*, Advances in Neural Information Processing Systems 27, 2014.

[3] D. P. Kingma, J. L. Ba, *Adam: A method for stochastic optimization*, In Proceedings of the 3rd International Conference for Learning Representations, 2015.

[4] K. He, X. Zhang, S. Ren, J. Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[5] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, *Enhanced deep residual networks for single image super-resolution*, Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on. IEEE, 2017.

[6] J. Y. Zhu, T. Park, P. Isola, A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, IEEE Computer Vision International Conference (ICCV), 2017.

[7] E. Agustsson, R. Timofte, *NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2017.

[8] Y. Yuan, S. Liu, J. Zhang, Y. Zhang, C. Dong, and L. Lin, *Unsupervised Image Super-Resolution using Cycle-in-Cycle Generative Adversarial Networks*, Computer Vision and Pattern Recognition Workshops (CVPRW), 2018.

[9] K. I. Kim and Y. Kwon, *Single-image super-resolution using sparse regression and natural image prior*, IEEE transactions on pattern analysis and machine intelligence, 2010.

[10] H. Zhang, J. Yang, Y. Zhang, T. S. Huang *Non-Local Kernel Regression for Image and Video Restoration*, IEEE transactions on image processing, 2011.

[11] J. Sun, J. Sun, Z. Xu, H. Shum, *Image Super-Resolution using Gradient Profile Prior*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2008.

[12] X. Li, M. T. Orchard, *New Edge-Directed Interpolation*, IEEE Transactions on image processing, 2001.

[13] L. Zhang and X. Wu, *An Edge-Guided Image Interpolation Algorithm via Directional Filtering and Data Fusion*, IEEE Transactions on image processing, 2006.

[14] J. Allebac and P. W. Wongt, *Edge-Directed Interpolation*, The IEEE International Conference on Image Processing (ICIP), 1996.

[15] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi- Morel, *Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding*, The British Machine Vision Conference (BMVC), 2012.

[16] H. Chang, D.-Y. Yeung, and Y. Xiong, *Super-resolution through neighbor embedding*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2004.

[17] X. Gao, K. Zhang, D. Tao, and X. Li, *Image super-resolution with sparse neighbor embedding*, IEEE Transactions on Image Processing, 2012.

[18] C. Dong, C. C. Loy, K. He, and X. Tang, *Learning a deep convolutional network for image super-resolution*, The European Conference on Computer Vision (ECCV), 2014.

[19] C. Dong, C. C. Loy, and X. Tang, *Accelerating the superresolution convolutional neural network*, The European Conference on Computer Vision (ECCV), 2016.

[20] J. Kim, J. Kwon Lee, and K. M. Lee, . *Accurate image superresolution using very deep convolutional networks*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2016.

[21] J. Kim, J. Kwon Lee, and K. M. Lee, *Deeply-recursive convolutional network for image super-resolution*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2016.

[22] C. Dong, C. C. Loy, and X. Tang, *Accelerating the superresolution convolutional neural network*, The European Conference on Computer Vision (ECCV), 2016.

[23] C. Leding *et al.*, *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, The IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR), 2016.

[24] T. Karras, S. Laine, T. Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, Neural and Evolutionary Computing, 2018.

[25] Y. Zhang, S. Liu, C. Dong, X. Zhang, Y. Yuan *Multiple Cycle-in-Cycle Generative Adversarial Networks for Unsupervised Image Super-Resolution*, The Conference for Telecom Innovation (TIP), 2019.