Chair of Digital Image Processing

Single Image Super-Resolution for Car Plates Using Generative Adversarial Networks

Master Thesis by

IHEB CHHIBI

1. Examiner	2. Examiner
Prof. Dr. Tomas Sauer	Prof. Dr. Michael Granitzer

September 2, 2021

Contents

Abstract

Despite the advance realized by recent studies in the field of image processing, and Super-Resolution more specifically, researchers focus their investment on models and architectures capable of generating the finest texture details while adapting their findings both to single image and multiple image Super-Resolution. The datasets used for training the models based on Neural Networks are chosen according to their high quality, randomness, and feature richness. The super-resolved images acquire remarkable details, yet are often unpredictable and hardly fit for monotonous tasks.

In this thesis we offer to test a state-of-the-art architecture, that is Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) [Tan18], on a custom-generated dataset applicable to car license plates.

The adopted model enhances the standard SRGAN generator [Twi17] using a recursive Residual-in-Residual Dense Block (RRDB) featuring only convolutional and Rectified Linear Activation (ReLU) layers. It also applies a relativistic aspect to the conventional discriminator through Relativistic average GAN or RaGAN. The model also upgrades the adversarial loss with two additional terms, namely the perceptual loss using VGG-extracted features, and a standard pixel loss. The trained Neural Network outperforms on the first generated dataset that adopts a trivial structure, delivering remarkable Peak Signal-to-Noise Ratio (PSNR) values of over 40 dB. Hence the need to create more robust data samples that resemble real license plates. We generate a second dataset with images deriving from the German license plate structure. The solution was successful in producing perceptually satisfying images with relatively high PSNR. It also proved scalable if trained on high-quality datasets.

Acknowledgments

First and foremost, I would like express my deepest gratitude to my supervisor, Professor Dr. Tomas Sauer, for giving me the opportunity to execute and achieve my master thesis, and for his valuable expertise and insightful feedback.

I would also like to acknowledge Mr. Ruben Fischer for his technical support and guidance throughout the model training phase, as well as Dr. Wiem Fekih Hassan for providing the knowledge and advice I needed to successfully write my dissertation.

Finally, I would like to thank all my family and friends for their unwavering support and wise counseling during my master thesis and overall education.

List of Figures

List of Tables

1 Introduction

1.1 Motivation

For the past two to three decades, a relatively large amount of human and material resources was dedicated to studying the feasibility of certain challenges that were once perceived as fictional. The introduction of the term Deep Learning (DL) to the Machine Learning (ML) community was a big step forward into the newly introduced paradigm shift. Researchers started viewing challenges with theoretically unachievable results from wider angles. With that being said, discussions about the starting point and the most promising approach evolved into discussions about finding the best architecture and the most suitable dataset. Within the introduced context comes the topic to be discussed in this thesis, namely Super-Resolution (SR) and Single Image Super-Resolution (SISR) to be more specific, which has been an attractive topic during the last decade. SISR basically aims to recover High-Resolution (HR) images from Low-Resolution (LR) input images [Twi17]. This issue, also known as *zoom in and enhance* as referred to by Crime Scene Investigation (CSI) movies, states that in order to render an image more appealing to the human eye, it needs to either regain missing information lost due to limited hardware performance or have its details adapted to its dimensions. This contradicts the data processing inequality concept stating that *"no clever manipulation of the data can improve inference"* [Xie], or in our case, the image resolution.

The SISR challenge started to gain attention in the early 2000s and was then considered one of the most attractive research topics [Yu18]. The idea of restoring high frequency details proved to be useful and was exploited in many fields such as medical image processing (reconstruction of cardiac Magnetic Resonance Imaging (MRI)) [Rue14], satellite image processing [AN02], and face recognition [Kak11], where high edge-precision is most wanted.

The primary SISR algorithms used smoothing interpolation methods, mainly the Bicubic Interpolation [Twi17]. Being the target method for non-trainable SISR due to the appealing nature of its results, it is still used as a reference to measure the recent methods' outperformance. Following the data processing inequality previously stated, a higher image resolution can be achieved through an additional source of information. This is where Neural Networks (NNs) come to pass. A pre-trained NN can learn to add new details into a LR image using pattern similarity based on knowledge acquired from a dataset of non-specific images.

In the recent years, Research performed by Chao Dong *et al.* [HT15] on Super-Resolution Convolutional Neural Network (SRCNN) proved the efficiency of pre-trained NNs over traditional methods that include interpolation [Asi08], internal similarities [X14], or single image example based SR [GM12]. This brought prosperity to the ongoing topic upgrading the challenge from seeking the best algorithm to developing the finest NN architecture.

Numerous architectures and different training strategies were also developed, all with the same goal, and that is to outperform the traditional 3-layer SRCNN. In that regard, better results have been achieved, yet these results tend to be over-smoothed and usually lack high texture details. The SR problem as implemented using NNs usually comes with a high downscaling factor. A considerable amount of information is lost upon downscaling a HR image, and the reconstruction of the SR counterpart is examined and optimized through pixel-wise difference using the Mean Squared Error (MSE). The challenge then comes to minimizing the MSE, which is also convenient since it results in a maximization of the Peak Signal-to-Noise Ratio (PSNR) [Twi17]. Referring to the MSE proved to be mathematically clear, convenient, and easy, therefore its appeal as an assessment function [Sim04]. However, pixel-wise difference-based methods usually struggle to generate high texture details essential to upgrade the perceptual quality of the SR image, therefore, they do not necessarily match the human eye evaluation.

Further research has been conducted to propose new perceptual-driven methods to either replace or enhance the previously established MSE functions [Tan18]. The resulted perceptual loss takes advantage of the work proposed by Andrew Zisserman *et al.* [ZS15] to optimize SR algorithms in a feature space instead of a pixel space [Tan18] using a pre-trained Visual Geometry Group (VGG) model.

Overcoming training limitations encountered by SRCNNs was the next step. The primal focus was to enhance the CNN architecture using pixel dependency [NS17]. Later, skip connections were added to the prime Neural Network, making the training less consuming and allowing the model to deepen its layers, hence the replacement of SRCNNs with Super-Resolution Residual Networks (SRResNets). Ledig *et al.* [Twi17] proposed a new paradigm, that is to produce images that look real and appealing to the eye instead of the focus on real images. This is when Super-Resolution Generative Adversarial Networks (SRGANs) were introduced [Twi17]. The architecture used two trainable neural networks instead of one and added a second term to the perceptual loss, namely the adversarial loss.

1.2 Structure of the Thesis

In this thesis, we adopt the model ***Enhanced Super-Resolution Generative Adversarial Networks (ESRGANs)*** first introduced by Wang. *et al.* [Tan18], and derived from the pioneer work presented by Ledig *et al.* in the SRGAN paper [Twi17]. We train the model on a custom-created dataset that fits the proposed problematic.

Chapter 2 is divided into two main sections. In Section 2.1, we thoroughly go through the knowledge background needed to apprehend the presented solution. We combine the mathematical background corresponding to the Super-Resolution topic applied in image processing, along with the knowledge related to the field of Artificial Intelligence (AI) and Machine Learning. In Section 2.2, we give an insight into the models, architectures, and research previously established in the Super-Resolution study discipline.

Chapter 3 gives a detailed description of the used method. We further explain the specifications already mentioned in the ESRGAN research paper [Tan18], with emphasis on the architectural upgrades in comparison to the model built by Ledig *et al.* [Twi17]. Each independent section will be accompanied with explanatory elements [1] adapted to our custom dataset.

Chapter 4 is divided into three main sections. Section 4.1 thoroughly describes the custom datasets. In this section, we go through the context of creation for each dataset and explain the different elements utilized as common parts for the image samples.

[1] Figures and equations

In Section 4.2, we provide further training details, including the hardware and software properties, and the values for the training parameters. The last section delivers the results found during both the training phase and the testing phase. The training phase is illustrated through graphs that depict different loss functions and their evolution throughout the whole process. The testing phase is assessed through a perceptual comparison between the different model outputs.

In Chapter 5, we consider all pre-established speculations and obtained results into a thorough evaluation of the utilized model. The analysis contains an assessment of the model regarding both used datasets, chosen training parameters, and network interpolation. We also compare our model to a loaded pre-trained ESRGAN and discuss the major differences between these solutions.

Finally, in Chapter 6, we conclude with a brief review of the thesis and a summary of the final evaluation. We also discuss the next step to further improve our model training and open space for potential future development.

2 Background

This chapter provides the necessary knowledge to comprehend the thesis content.

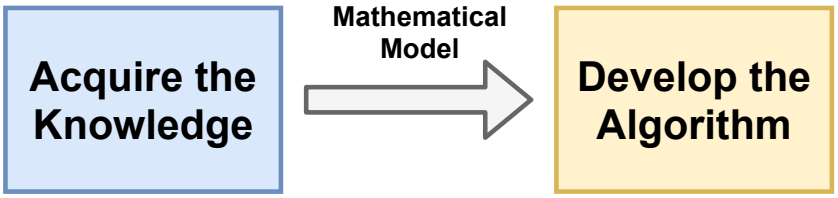
2.1 Definition of Terms

2.1.1 Machine Learning

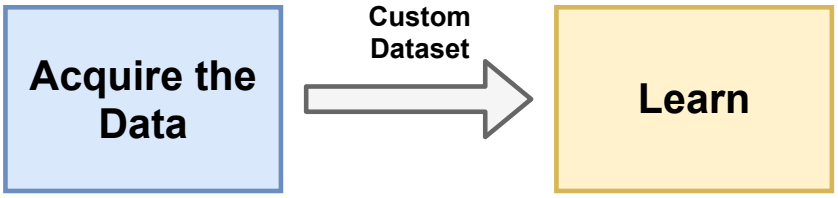
Before diving into a basic definition of Machine Learning, it is important to highlight the fact it gained its popularity over the past few decades for numerous reasons. It is only natural that the abundance of data becomes an attractive factor to why ML usage is important, especially with the exponential growth of the information records and the inability of the human effort to adapt to it. ML comes with the promise of acquiring meaning from all that data.

We introduce ML as a replacement for algorithmic methods traditionally used in engineering approaches [Sim18]. In Figure 2.1a, the standard procedure describes the necessity of collecting the knowledge to build an explicit algorithm that directly solves a pre-defined challenge using a designed mathematical model. ML can be presented as a branch of computer science arising from the study of pattern recognition and computational learning theory [VB16]. The diagram in Figure 2.1b defines basic ML as a data-driven algorithm able to learn and refine certain decision-making parameters, without explicit design, through a pre-processed set of examples. Therefore, a basic ML process is a replacement of the knowledge acquisition step with a relevant dataset collection step that is an easier procedure in most cases.

Hence the purpose behind building ML models is to reduce human interaction bringing the dawn of self-automated machines with a self-built set of rules generated from learned examples.



(a) Basic Algorithmic Process Diagram



(b) Machine Learning Process Diagram

Figure 2.1: Comparison between basic and machine learning algorithms

2.1.2 Deep Learning and Neural Networks

Deep Learning represents a specific sub-branch of Machine Learning. Although this practice came as a response to much more complicated challenges that needed better understanding of the data at hand to provide a more accurate set of results, the *"Deep"* in *Deep Learning* rather refers to the amount of layers used in the architecture [Cho18]. The more layers a DL model has, the *deeper* it is. Compared to other *shallow* ML techniques, DL models feature multiple layers (up to hundreds)[Cho18]. Such stacked up layers are referred to as *Neural Networks*.

The terminology *Neural Network* refers to neurobiology. Although such model structures are not an honest representation to how the human brain functions, they are strongly inspired by the neurons' way of communication within the nerve tissue. Figure 2.2 presents a simplified architecture of a basic NN prototype. The macroscopic hierarchy of the model is described as a 3-level hierarchy featuring an input layer with n input nodes, l hidden layers, and an output layer with m output nodes. All these nodes are connected via weighted links. The sole purpose behind the model training is to adjust these weights (or parameters) initially randomly assigned, such that the provided network appropriately allocates its inputs to their desired targets [Cho18]. This process requires a factor allowing the model performance observation, meaning a loss function that measures how close is the set of parameters to the target set providing an ideal or close to ideal result, and that is by computing the absolute distance between the pre-

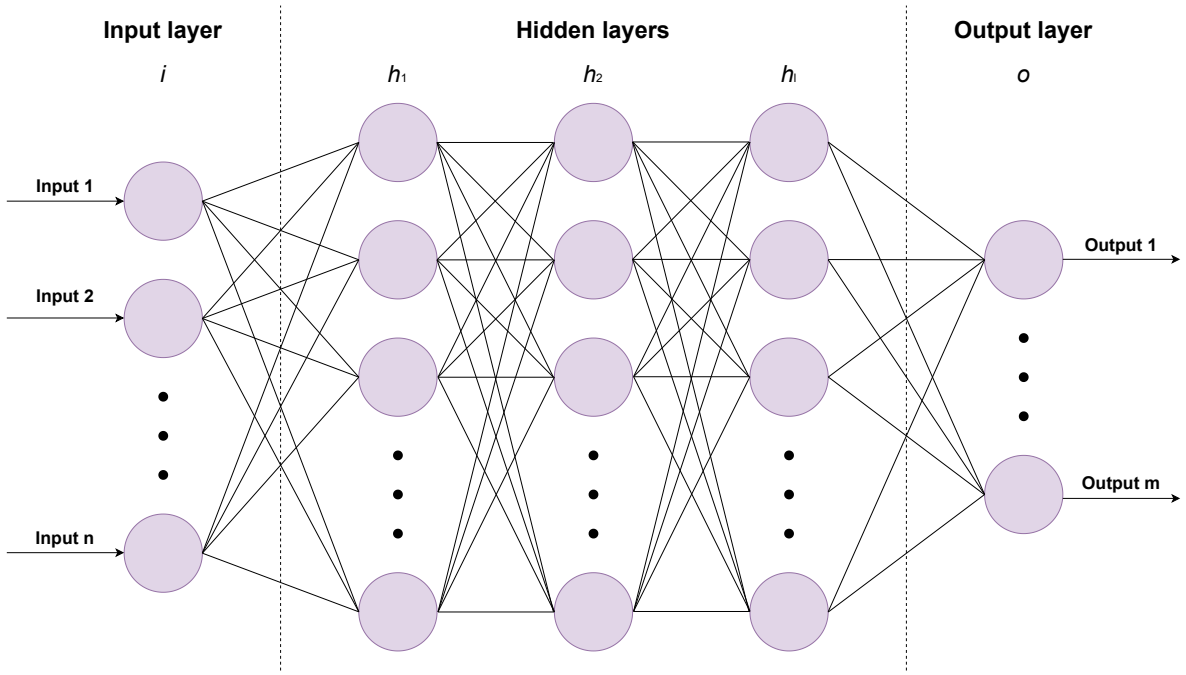


Figure 2.2: Graphical representation of a NN with n inputs, n outputs and 3 hidden layers

diction and the true value [Cho18]. Updating the weights accordingly is the *optimizer's* task. This is done through an algorithm called *backpropagation* which is considered the core algorithm when it comes to DL [Cho18].

2.1.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a form of deep learning models that is specifically designed to solve computer vision problems. The hidden layers in CNNs (referenced to in subsection 2.1.2) are mainly convolutional layers able to detect and recognize different patterns. Although CNNs were recently able to migrate towards solving more generic challenges such as Natural Language Processing (NLP), this pattern detection is what makes it primarily used for image analysis. The convolution operation is a basic mathematical operation involving an input and a pre-defined filter to produce an activation [Kre19]. In the case of patterns recognition for image processing tasks, the input is a 3-dimensional tensor with a height, a width, and a depth dimension (or channel dimension) [ON15]. For RGB images (implemented in the dataset used for this thesis), the depth is a 3-dimensional axis (red, green and blue channels) [Cho18].

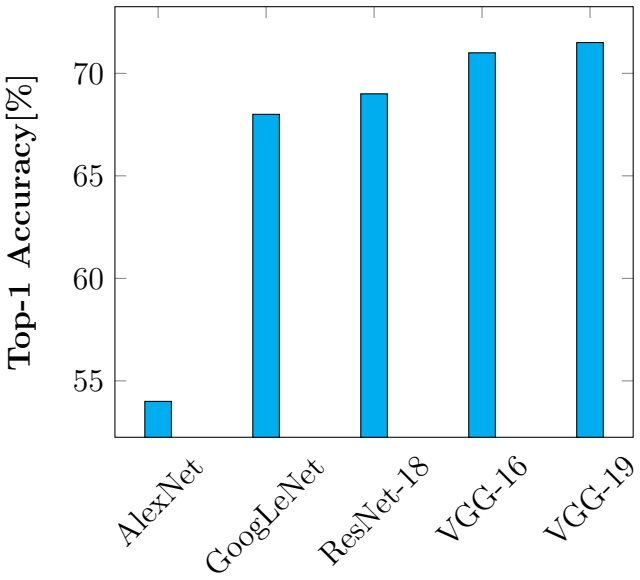


Figure 2.3: Representation of iconic CNNs and their recorded accuracy in the ILSVRC contest [Das17]

Within the context of visual pattern recognition, a number of iconic CNN architectures has risen above others, delivering remarkable accuracy scores in the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) held by the ImageNet project [Das17]. Figure 2.3 exhibits a bar chart with the results achieved by some of these exemplary CNNs. In the following two subsections, we further describe the architectures we adopted in our work (among those mentioned in the bar chart).

2.1.4 VGG Networks

VGG is an iconic deep Convolutional Neural Network designed by Visual Geometry Group of Oxford University (hence the name) [ZS15]. The NN comprises up to 19 layers and is known for outperforming traditional architectures in the field of image recognition and classification. Numerous modern image processing architectures are built on top of VGG networks. Authors often exploit a pre-trained VGG model to produce a feature space replacing the conventional pixel space.

Figure 2.4 depicts both VGG16 and VGG19 architectures. Each model features a number of convolutional layers with multiple kernels of size 3x3, max-pooling layers of kernel size 2x2, and fully connected layers.

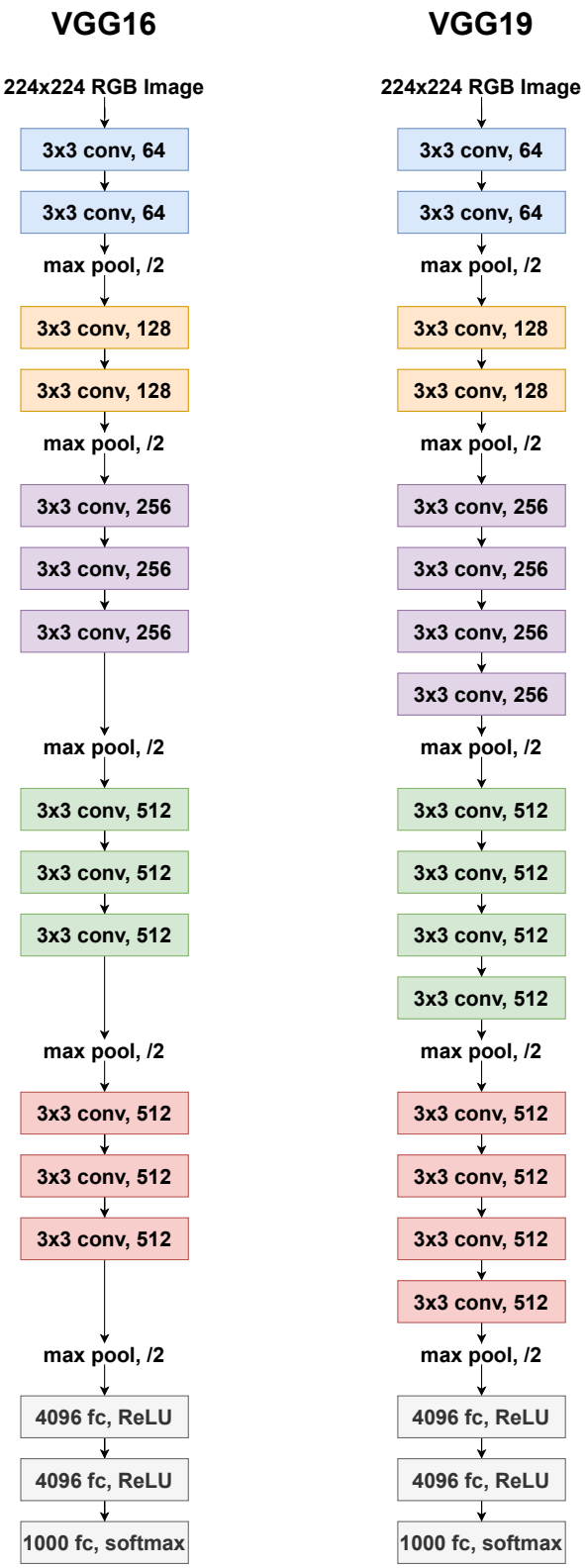


Figure 2.4: VGG16 and VGG19 model architectures

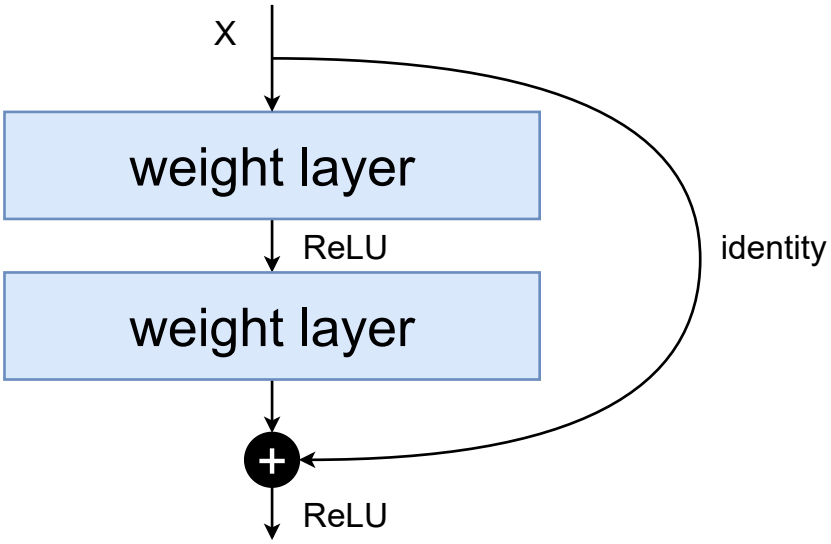


Figure 2.5: Residual block: a representation of a skip connection

2.1.5 Residual Networks

Enhancing a Neural Network to learn high-level features requires deeper architectures, meaning adding more layers. Although deep NNs often suffer from a major problem, that is vanishing and exploding gradients. A solution would be to exploit residual networks, which is a type of NNs implementing skip connections [Sun15].

Residual networks are built over residual blocks (seen in Figure 2.5). Each residual block implements a *skip connection* or a *shortcut* that takes an activation from one layer feeding it to another layer deeper in the NN, thus making identity functions easier to learn.

Basic residual networks are constructed by stacking up residual blocks. More sophisticated models are referred to as Dense Convolutional Networks (DenseNets) [Maa18]. Such NNs feature a different unit than residual blocks, namely dense blocks, where all layers are fully connected, meaning that a layer's activation is forwarded to all subsequent layers within the same dense block (see Figure 2.6).

For the sake of further simplification, we give the mathematical models for each residual network:

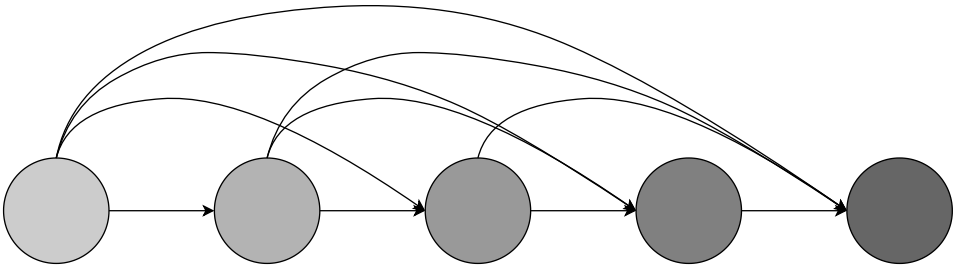


Figure 2.6: Graph representing a dense block; Nodes refer to network layers, arcs refer to skip connections; A deeper node is modeled by a darker color

Consider a residual network (or a dense block) of K layers and a single input image x_0. We denote as x_k the output of the k^{th} layer and H_k its non-linear transformation [Maa18].

Feed-forward Networks. Traditional NNs connect the k^{th} layer output directly to the $(k+1)^{th}$ layer input [Maa18]. The transformation is modeled by the following equation:

$$x_k = H_k(x_{k-1}) \tag{2.1}$$

Residual Networks. ResNets add a second connection that bypasses the transformation layer (skip-connection) adding an identity-function term to the previous formula [Maa18]. The transformation is then modeled by the following equation:

$$x_k = H_k(x_{k-1}) + x_{k-1} \tag{2.2}$$

Dense Convolutional Networks. DenseNets connect the output of the k^{th} layer to all $(k+i)^{th}$ subsequent layer inputs [Maa18]. Each layer k within a dense block then receives concatenation of previous activations as input. The transformation is modeled by the following equation:

$$x_k = H_k([x_0, x_1, ..., x_{k-1}]) \tag{2.3}$$

Where $[x_0, x_1, ..., x_{k-1}]$ denotes the tensor of feature-maps forwarded to the k^{th} layer by the layers $0, .., k-1$.

2.1.6 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are complex NNs introduced by Goodfellow *et al.* [CB14] and based on two sub-neural networks, namely the generator and the discriminator, trained simultaneously. In the context of image processing, it has the task of generating real looking images by recursive comparison to a set of authentic input examples [Cho18]. In order to understand the mechanism of GAN training, countless analogies could be referred to. In the book *Deep learning with python* [Cho18], this mechanism is modeled by the act of faking real paintings. The generator is simulated as a forger and the discriminator as the art dealer. At the start of training, the the forger delivers obvious fakes that the dealer easily spots from the real paintings, the latter is then bound to give feedback about the differences between real and fake paintings. As the training continues, the forger becomes better at faking authentic artwork, and the dealer improves his ability to spot differences. The goal then is to have real-looking fake results. In practice, real images are samples from the provided set of data. Fake images on the other hand are forwarded by the generator to match those samples (Figure 2.7).

As a result of this dual composition, training a GAN can be both difficult in terms it needs substantial tuning of the model architecture and parameters [Cho18], and tricky in terms of convergence given its dynamic nature. In theory, a NN convergence is achieved through a gradient descent algorithm which, to put simply, is driven by a mechanism of back propagation to seek the global minimum. GAN training, on the other hand, seeks a balance between two agents rather than a predetermined extremum [Cho18].

GAN training can be defined using the following functions [Jol18], i.e. the discriminator loss and the adversarial loss:

$$L_D^{GAN} = \mathbb{E}_{x_r \sim P}[\tilde{f}_1(D(x_r))] + \mathbb{E}_{x_f \sim Q}[\tilde{f}_2(D(G(x_f)))] \tag{2.4}$$

$$L_G^{GAN} = \mathbb{E}_{x_r \sim P}[\tilde{g}_1(D(x_r))] + \mathbb{E}_{x_f \sim Q}[\tilde{g}_2(D(G(x_f)))] \tag{2.5}$$

Table 2.1 describes the used terms.

Term	Description
$\tilde{f}_1, \tilde{f}_2, \tilde{g}_1, \tilde{g}_2$	Scalar-to-scalar functions
P	Distribution of real data
Q	Distribution of fake data
$D(x)$	Discriminator evaluated at x
$G(x)$	Generator evaluated at x
x_r	Reference to real data
x_f	Reference to fake data
L_D	Discriminator loss
L_G	Adversarial loss

Table 2.1: Brief description of the terms used in the GAN loss function equations

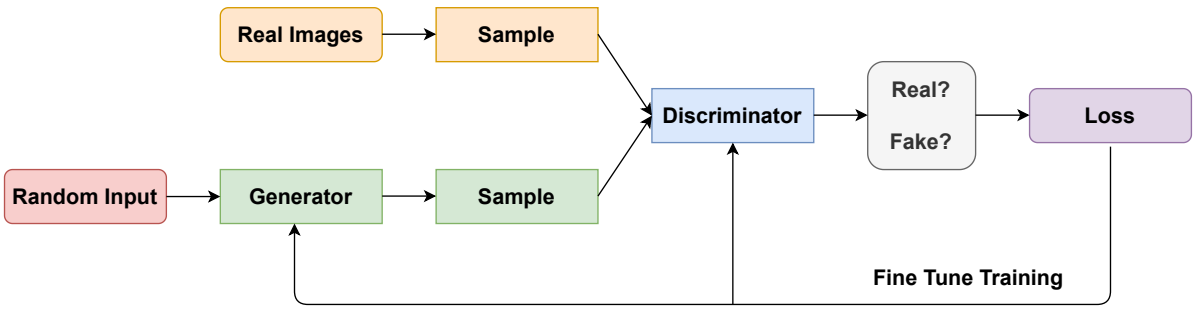


Figure 2.7: Graphical representation of GAN training process

2.1.7 Batch Normalization (BN)

When using large datasets for training a NN, we often come across an imbalance in different dimensions of the data, which calls for data pre-processing, a very important step that precedes the training phase [Dee18]. Specifically, the imbalance in numerical scales requires a prior normalization[1] and standardization [2] that prevents problems during training, such as long training processes and exploding gradients.

NNs often use Stochastic Gradient Descent (SGD) to learn and update weights after each epoch. This method has proven to be simple and effective, yet it might occur that one of the weights becomes significantly larger than the others and cascades as the networks go deeper to cause instability. In their paper entitled Accelerating Deep Network Training by Reducing Internal Covariate Shift [IS15], Ioffe *et al.* proposed to apply BN on specific layers within the network.

Applying a BN simply refers to normalizing each dimension of each batch's output vector [IS15]. If we denote a d-dimensional layer's vector $x = (x^{(1)}, ..., x^{(d)})$, a basic normalization would be:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{2.6}$$

Such normalization can sometimes change the layer's representation. The countermeasure proposed by Ioffe *et al.* [IS15] is that *"the transformation should represent the identity function"*. This is accomplished by introducing two new learnable parameters for each activation, namely $\gamma^{(k)}$ and $\beta^{(k)}$, respectively used for scaling and shifting as follows:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)} \tag{2.7}$$

We then denote the normalized output for each batch element i as $y_i = BN_{\gamma,\beta}(x_i)$

To sum up, we can say that instead of normalizing the data before passing it to the input layer, what BN does is apply batch norm to the output data for the activation function in single layers, therefore speeding up the training process and avoiding the over-influence of large values.

[1] A typical normalization practice is to transform all numerical values down to a scale of 0 to 1

[2] Basically subtracting the mean of the dataset from each point and then dividing the difference by the standard deviation

Figure 2.8: Representation of three different interpolation techniques [Coo21]

2.1.8 Bicubic Interpolation

Interpolation is considered a central technique when it comes to processing images [San13]. Its use cases involve restoring images upon simple modifications such as rotation or scaling and adapting to new hardware or channel upon display or printing. Such frequent tasks require fast response where the quality is often overlooked. Image interpolation is considered a SR algorithm that requires no training. Instead, it calculates an empty pixel's value from nearby deterministic pixels using a predefined mathematical function. This function has as variable the distance d_k between the considered pixel and the k closest pixels. It is described by the following formula:

$$f(x) = \sum_k a_k u(d_k) \tag{2.8}$$

Where $u()$ is the interpolation kernel and a_k are the interpolation coefficients [San13]. The kernel $u()$ is explicitly written according to the chosen interpolation method. In this context, three techniques in particular, namely nearest neighbor, bilinear and bicubic, are often used due to their highly adaptive nature and ease of computation (Figure 2.8).

When a smooth result is desired and speed of computation can be overlooked, bicubic interpolation is often preferred over bilinear or nearest neighbor in image resembling [San13]. In exchange for a minor time lag, the output is smoother in comparison to other methods. In SISR, bicubic interpolation is usually the default reference for more sophisticated algorithms to build a thorough PSNR comparative study.

Bicubic interpolation takes into consideration 16 pixels (4x4) instead of 4 pixels (2x2)

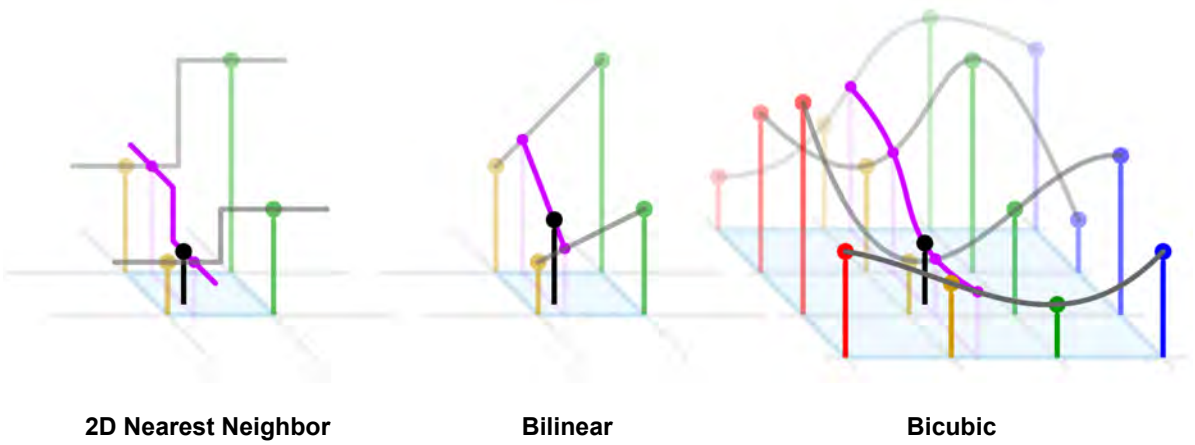


Figure 2.9: Comparison between 2D nearest neighbor, bilinear and bicubic interpolation techniques [Cmg21]

as used in 2D nearest neighbor and bilinear techniques (shown in Figure 2.9), which explains the time lag leveraged in a more complex calculation [San13]. Therefore, the bicubic kernel uses the corner pixel coordinates as well as the derivatives computed at these pixels. The bicubic interpolation function is given by the following formula:

$$f(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j \tag{2.9}$$

and the problem is reduced to calculating the coefficients a_{ij}.

2.2 State of the Art

2.2.1 Prediction-Based Algorithms

An introduction to SISR were prediction-based algorithms. Although SR and fine-textured details were not the goals of such methods, the outcome was remarkably more appealing than the basic nearest neighbor process. Techniques such as bilinear and bicubic interpolations were used, to later become a basic reference for comparison to newly developed formulas [Twi17]. Further advanced refinements such as *Cubic B-spline* functions by Hou and Andrews [HA78] and *Curvature Based* techniques by Ghuatam *et al.* [Gop15] then came to adjust basic interpolation flaws [Asi08]. Aftab *et al.* also proposed a fast hybrid method for transitioning between covariance-based interpolation techniques and curvature-based interpolation techniques to test on aerial images [Asi08]. To further emphasize texture details, Xin Li and Michael T. Orchard developed an edge-directed interpolation algorithm based on the geometric duality between the LR covariance and the HR covariance [LO04].

2.2.2 Learning-Based Algorithms

Although the previously mentioned algorithms had a more appealing output than the original LR input, they only delivered over-smoothed results and the solutions didn't quite add any information that wasn't already in the image.

The research performed by Hong Chang *et al.* [Xio04] was one of the first steps into implementing learning-based methods in SR. The algorithm used Locally Linear Embedding (LLE) to train an ML model on a set of small image patches in two different feature spaces [Xio04]. Later on, other ML algorithms such as random tree [SP15] and random forest [LB15] were used to overcome minor regression flaws discovered in the process [Twi17].

It wasn't until the introduction of NNs to the SISR practices that we could expect drastic changes on high-frequency-detail levels. The pioneer work performed by Dong *et al.* [HT15] in the SRCNN paper delivered exceptional results and a significant improvement to the state-of-the-art. The model used a 3-layer deep SRCNN to learn an end-to-end mapping between the LR and HR images. The SRCNN paper set a new direction to

solving the SISR problem. Research authors began investing in different methods to either enhance or deepen the basic CNN architecture. The work proposed by Andrew Zisserman *et al.* [ZS15] features a developed CNN for large-scale image recognition. Jiwon Kim *et al.* [LL16] developed a Deeply-Recursive Convolutional Network (DRCN) with up to 16 recursions. The model used recursive supervision and skip-connections to avoid eventual exploding and vanishing gradient issues that may be caused by the recursion depth. Yet exceeding the three layers still came with a high price of both performance and training.

The introduction of GANs was a remarkable milestone in NN architectures. The success of the 2-tier model in achieving subjectively appealing results combined with the network scalability made it the perfect candidate to follow the trail of pioneer work in SISR. In this context, Ledig *et al.* [Twi17] developed a photo-realistic SISR model using SRGAN. The model uses two NNs, namely a generator and a discriminator, and implements residual blocks with skip connections. The discriminator is then trained to differentiate between real images and super-resolved images [Twi17], while the generator is trained to output results that highly resemble the HR input.

Following the model of Ledig *et al.* [Twi17] to leverage residual network architectures in SISR, Bee Lim *et al.* proposed a modified deep residual network (SRResNet) in the paper Enhanced Deep Residual Networks for Single Image Super-Resolution (EDSR) [NL17]. The research used a similar model construction, featuring minor modifications to the different residual blocks by removing BN layers.

In their recent paper Enhanced Super-Resolution Generative Adversarial Network (ESRGAN)[Tan18], Wang *et al.* proposed structural improvements to the SRGAN model key components. They removed BN layers as proposed by the EDSR paper [NL17]. They also added more layers to SRGAN architecture and used residual scaling to stabilize the model training, resulting in a replacement of primal residual blocks with Residual-in-Residual Dense Block (RRDB) [Tan18]. The discriminator also received further enhancement using Relativistic average Generative Adversarial Network (RaGAN) which, instead of differentiating between real and fake images, learns to detect whether the output looks more realistic than fake data or less realistic than real data.

2.2.3 Loss Function

One of the known struggles faced by the CNN architecture in the SRCNN paper [HT15] alongside its limited scalability, is the use of pixel-wise loss functions, the Mean Squared Error (MSE) to be more specific. Pixel-dependent metrics usually attract research authors due to their mathematical simplicity and clear physical meaning [Sim04]. During the SRCNN training, minimizing the MSE as a loss function results in a direct maximization of the PSNR, a convenient compromise to enhance the model efficiency. However, such metrics usually fail to match perceived visual quality and tend to overlook fine texture details rendered as high-frequency image features. In Figure 2.10 a representation of the MSE/SSIM[3] hypersphere is shown, where all distorted images on the circle are equally distant from the original image in terms of MSE, yet they do not look equally appealing. Various methods have been proposed as an upgrade to the classic MSE loss function. In their paper pixel recursive super resolution [NS17], R. Dahl *et al.* introduce an extension to the pixelCNN using a probabilistic deep network architecture [NS17].

Later on, pixel-dependent loss functions have been replaced by perceptual-driven approaches. Inspired by perceptual similarity for object recognition [Bet15], Johnson *et al.* [Fei16] define the perceptual loss as a combined feature reconstruction loss and a style reconstruction loss. The result is used for training feed-forward networks for image transformation tasks. For SR tasks in particular, the perceptual loss is reduced to the feature reconstruction, and minimized in a feature space instead of a pixel space. In the SRGAN paper [Twi17], Ledig *et al.* formulate the perceptual loss as the weighted sum of a content loss calculated on a VGG19-extracted feature space, and an adversarial loss. While the content loss trains the model to generate more appealing images, the adversarial loss is trained to generate realistic super-resolved images based on a set of HR-LR samples. To further enhance the SRGAN model, Wang *et al.* [Tan18] propose to add the weighted MSE (either L1 or L2 loss) to the equation, in order to set a compromise between perceptually pleasing output and a high PSNR. A further improvement in the ESRGAN architecture is modifying the VGG loss to compare feature maps before activation, as these feature maps contain information that is essential to the training and assessment of the model, which is being removed by the activation functions.

[3]Structural Similarity Index

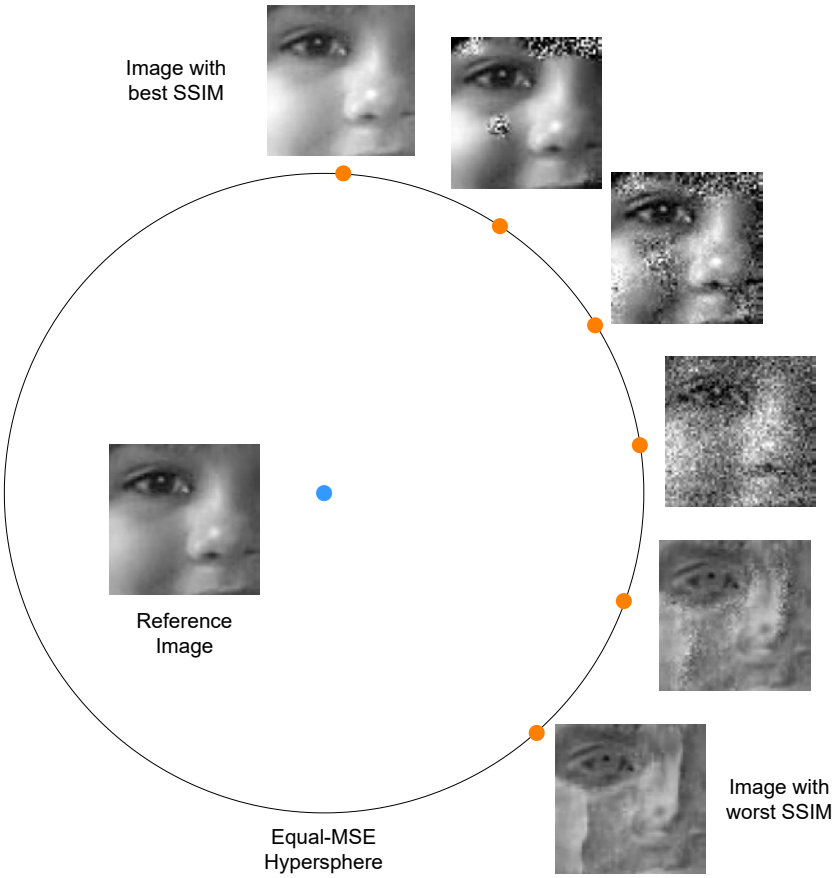


Figure 2.10: Image MSE/SSIM Hypersphere [Sim03]

3 Methods

Our main goal is to train a Neural Network to estimate a High-Resolution German license-plate image, from its Low-Resolution counterpart, by forwarding an output image that we denote SR and that the network assumes is real.

In this chapter, we describe the method ESRGAN used in this thesis, first implemented by Wang *et al* and described in their paper [Tan18] as an enhanced version of the iconic SRGAN paper [Twi17]. We decompose the chapter into four sections according to the model's highlighted enhancements, in comparison to previous algorithms. We describe in the first section the network architecture with a focus on the generator (A basic architecture is modeled in Figure 3.1). In the second section, we explain the concept of RaGAN and relativistic discriminator. Finally, the third and fourth sections both discuss the effect of combining old and new approaches linearly to optimize both the perceptual loss and the network result's visual properties.

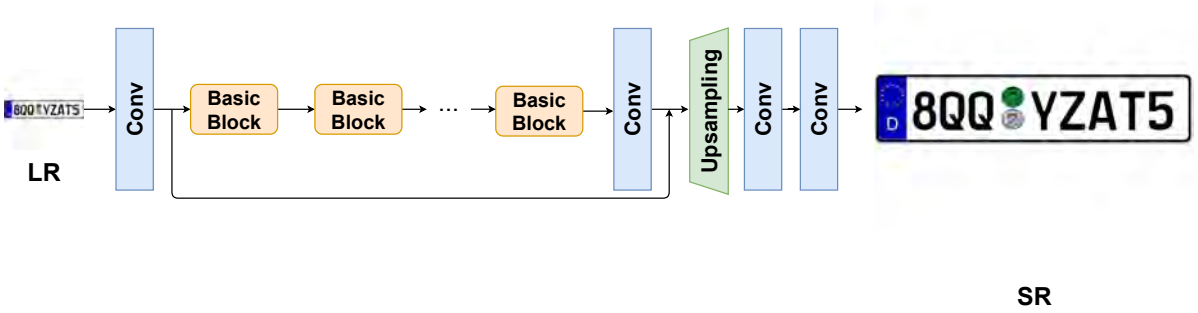


Figure 3.1: The implemented architecture is ESRGAN [Tan18], similar to the SRGAN network [Twi17] with a customizable "basic block".

3.1 Network Architecture

The general architecture is a dual CNN trained as a GAN (referenced in 2.1.6) where the generator learns to trick the discriminator into forwarding super-resolved images[1] as real images. Using this algorithm, the model will not only forward perceptually superior and pleasant license plates[2], but will also output legit German license plates with matching flags and registration seals.

Following Ledig *et al.* in the SRGAN paper [Twi17], the generator features a number of stacked *basic blocks* (Figure 3.3a). Each basic block should comprise a subnetwork implementing a residual learning technique (referenced in subsection 2.1.5).

In this section, we will describe the two modifications applied to the generator's structure in the ESRGAN method [Tan18]: First is the removal of all BN layers, and second is the application of RRDB depicted in Figure 3.3, which combines both major residual learning techniques (subsection 2.1.5) in a single basic block.

3.1.1 Batch Normalization Layer Removal

For most NNs, including SR and other PSNR-related tasks, BN layers proved to be efficient for speeding up the training process and avoiding exploding gradients (see subsection 2.1.7), therefore increasing the network's performance.
However, when it comes to deeper networks, including GANs, BN layers usually introduce displeasing artifacts hindering the stability of the training performance [Tan18]. As previously mentioned, BN layers use the mean $\mathbb{E}(x)$ and the variance $Var(x)$ over a mini-batch for normalization. Furthermore, these values are approximated over the training dataset when applied for testing. When such measures have uncorrelated values, Batch Normalization tends to introduce unpleasant artifacts reducing the model's ability for generalization, a valuable characteristic of SR.

In the case of performing SISR for German license plates, the provided dataset has a limited diversity range[3]. However, the choice of removing BN layers is based on the statement that they perform badly on deeper networks and GANs in particular [Tan18].

[1] For the rest of the thesis, we will refer to *license-plate images* simply by *images*
[2] Meaning clear, nonblurry letters and digits
[3] Only the used characters and their distribution is different from one image to another, whereas the general structure is fixed

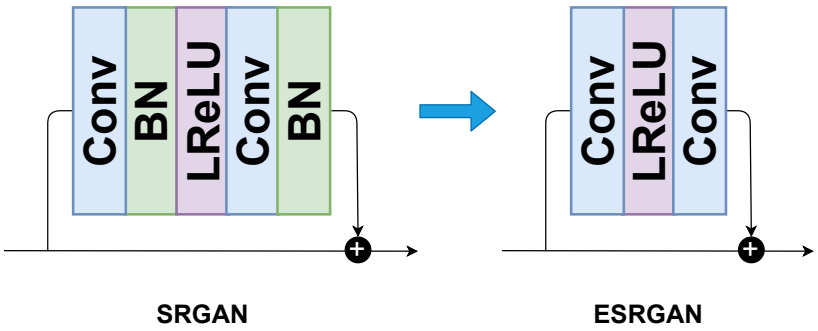


Figure 3.2: Comparison between SRGAN residual block and ESRGAN residual block (without BN)

3.1.2 Residual-in-Residual Dense Block (RRDB)

In an attempt to follow the study speculating that adding more connections can always improve the model's performance [NL17], We implement a 3-tier residual architecture, i.e. a residual-in-residual dense architecture, while keeping the basic generator structure used in the SRGAN paper (Figure 3.1) [Twi17; Tan18]. All levels of the executed structure are modeled in the figure 3.3. Each one of these levels implements a different residual learning technique:

- **(a) First level:** Fundamental generator architecture mentioned in the SRGAN paper [Twi17] and featuring a number of cascading basic blocks. These blocks are customized using two deeper residual levels.
- **(b) Second level:** A basic block is in fact a residual block featuring a number of dense blocks with skip connections (see Figure 3.3b). In order to further enhance fundamental residual blocks, we use residual scaling [NL17], meaning that we scale down each block output or residual by multiplying a decimal constant $\beta \in [0, 1]$ before adding the identity function to ensure stability.
- **(c) Third level:** As already mentioned in the subsection 2.1.5, dense blocks connect each irreducible block's output to all subsequent blocks (see Figure 3.3c). Each irreducible block comprises a convolutional layer and a LReLU layer (see Figure 3.4a and Figure 3.4b).

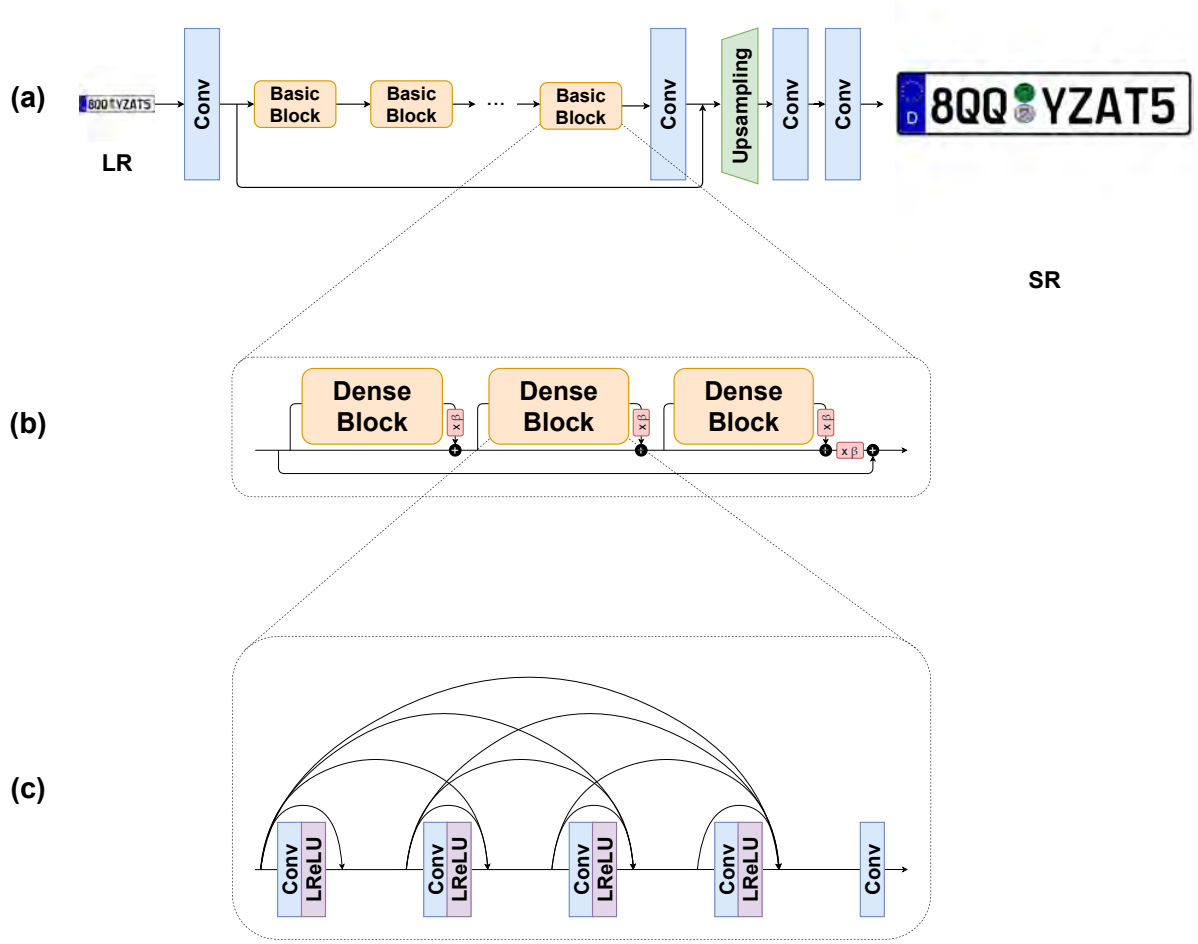
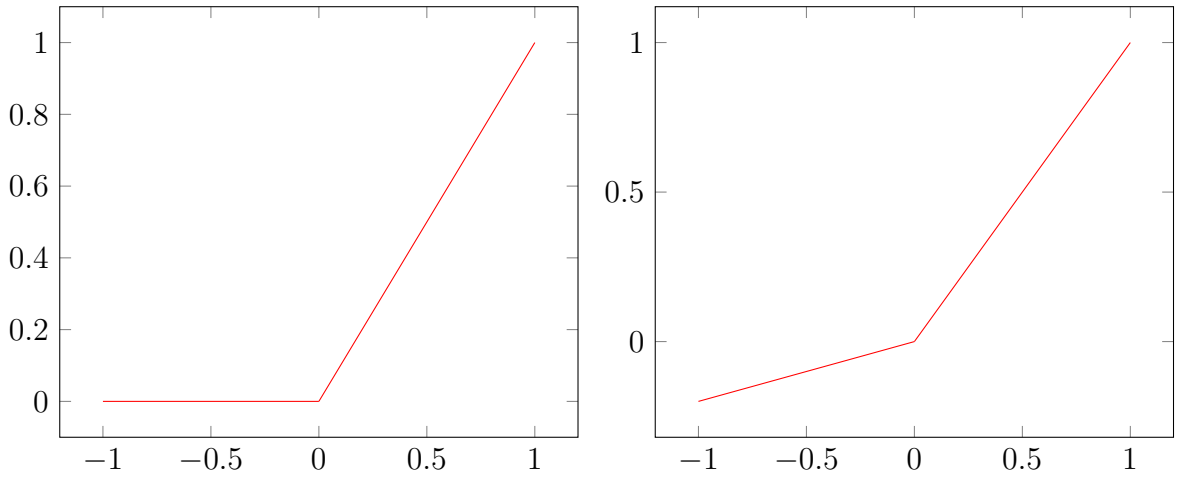


Figure 3.3: Representation of a Residual in Residual Dense Block (RRDB)
(a) The basic architecture for the ESRGAN generator
(b) First zoom in: residual block
(c) Second zoom in: dense block



(a) ReLU

(b) LReLU with a slope of 0.2 for $x < 0$

Figure 3.4: Representation of ReLU and LReLU functions for $x \in [0, 1]$

3.2 Relativistic Discriminator

Aside from the generator, we further improve the discriminator following the research proposed by Jolicoeur-Martineau [Jol18]. We therefore implement a Relativistic Generative Adversarial Network (RGAN), an enhanced version of the Standard Generative Adversarial Network or SGAN. In the case of SGAN, the discriminator D calculates the likelihood that the input image is real (see subsection 2.1.6). This deterministic assumption usually lacks precision, unless the model concurrently reduces the probability that real data is real [Jol18]. RGAN on the other hand uses a discriminator that estimates the probability that the real image x_r is more realistic than a fake image x_f [Jol18; Tan18]. RGAN can then be defined by the following functions:

$$L_D^{RGAN} = \mathbb{E}_{(x_r,x_f)\sim(P,Q)}[f_1(C(x_r) - C(x_f))] + \mathbb{E}_{(x_r,x_f)\sim(P,Q)}[f_2(C(x_f) - C(x_r))], \quad (3.1)$$

$$L_G^{RGAN} = \mathbb{E}_{(x_r,x_f)\sim(P,Q)}[g_1(C(x_r) - C(x_f))] + \mathbb{E}_{(x_r,x_f)\sim(P,Q)}[g_2(C(x_f) - C(x_r))]. \quad (3.2)$$

All used terms are described in the Table 3.1.

Term	Description
f_1, f_2, g_1, g_2	Scalar-to-scalar functions
P	Distribution of real data
Q	Distribution of fake data
$C(x)$	Discriminator before activation, evaluated at x
x_r	Reference to real data
x_f	Reference to fake data
L_D	Discriminator loss
L_G	Adversarial loss

Table 3.1: Brief description of the terms used in the RGAN and RaGAN loss function equations

Precisely, we implement a variant of RGAN called Relativistic average Generative Adversarial Network or RaGAN [Jol18] established as estimating the probability that real images x_r are more realistic than fake images x_f on average[4]. RaGAN can then be defined by the following functions:

$$L_D^{RaGAN} = \mathbb{E}_{x_r \sim P}[f_1(C(x_r) - \mathbb{E}_{x_f \sim Q} C(x_f))] + \mathbb{E}_{x_f \sim Q}[f_2(C(x_f) - \mathbb{E}_{x_r \sim P} C(x_r))], \quad (3.3)$$

$$L_G^{RaGAN} = \mathbb{E}_{x_r \sim P}[g_1(C(x_r) - \mathbb{E}_{x_f \sim Q} C(x_f))] + \mathbb{E}_{x_f \sim Q}[g_2(C(x_f) - \mathbb{E}_{x_r \sim P} C(x_r))]. \quad (3.4)$$

All used terms are described in the Table 3.1.

Furthermore, we consider the following equations as additional information:

$$D(x) = \begin{cases} sigmoid(C(x) - \mathbb{E}_{x_f \sim Q} C(x_f), & \text{if } x \text{ is real} \\ sigmoid(C(x) - \mathbb{E}_{x_r \sim Q} C(x_r), & \text{if } x \text{ is fake} \end{cases} \quad (3.5)$$

As a result, both real and fake image gradients are used for training the RaGAN generator, as opposed to the standard GAN generator that only uses fake data.

[4]Unlike RGAN that uses randomly sampled data

3.3 Perceptual Loss Function

Aside from the adversarial loss mentioned in section 3.2, we use a more efficient perceptual loss L_p proposed by Johnson *et al.* [Fei16], implemented by Ledig *et al.* [Twi17], and further enhanced by Wang *et al.* [Tan18].

Up until the recent years, many SR state-of-the-art approaches applied methods based on pixel-wise MSE loss that can be defined as follows:

$$L_{MSE} = \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} (I_{x,y}^{HR} - G(I^{LR})_{x,y})^2 \tag{3.6}$$

The aim was to target a distinctly high PSNR[5].
What Ledig *et al.* did in their SRGAN paper [Twi17] was trivialize the importance of a high PSNR compared to sharp edges and high frequency content, hence the definition of perceptual loss or VGG loss (see subsection 2.1.4).

The VGG loss is extracted from a pre-trained VGG19 network and defined by the following formula:

$$L_{VGG/i,j} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G(I^{LR}))_{x,y})^2 \tag{3.7}$$

Similar to 3.6, L_{VGG} is established as the euclidean distance between a generated image $G(I^{LR})$ and its HR counterpart I^{HR} (All used terms are described in the Table 3.2.)

Furthermore, we adopt the version applied by Wang *et al.* in ESRGAN [Tan18], where the VGG features are used before activation[6] as shown in Figure 3.5. The reason behind this update is to overcome feature sparsity caused by the activation function in deep NNs [Tan18] and leading to weak performances. Another reason is to avoid odd brightness added by the use of activated features in comparison to the original image.

In conclusion, the total loss function used for GAN training is:

$$L_{GAN} = L_p + \lambda L_G^{RaGAN} + \eta L_1, \tag{3.8}$$

[5]The very definition of Super-Resolution
[6]Instead of after activation as implemented in SRGAN

Term	Description
W, H	Image width and height in pixels
$W_{i,j}, H_{i,j}$	Feature map width and height within the VGG network
I^{HR}	High resolution image
I^{LR}	Low resolution image
$G(x)$	Generator evaluated at x
$\phi_{i,j}$	Feature map obtained between the j-th convolution and the i-th maxpooling layer

Table 3.2: Brief description of the terms used in the MSE and VGG loss function equations

where $L_1 = \mathbb{E}_{x_i}||G(x_i) - y||_1$ represents the 1-norm distance between the generated image and its HR counterpart (ground-truth), and λ, η represent stabilizing coefficients.

3.4 Network Interpolation

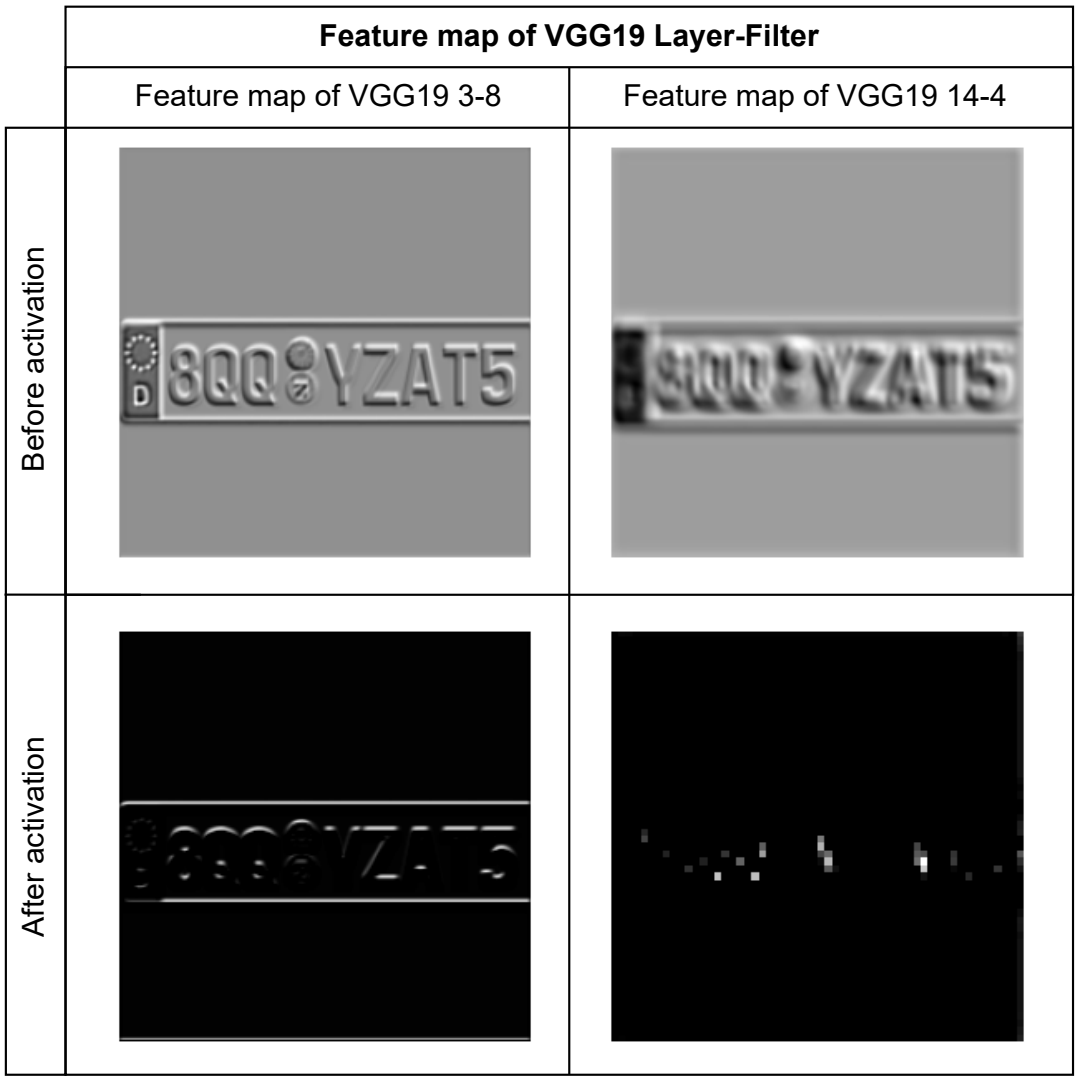
GAN-based networks occasionally introduce a distinctive unpleasant noise that can be remarkable to a certain degree. In order to eliminate or reduce such noise, we leverage PSNR-oriented methods abilities for minor artifact removal by using network interpolation, a flexible and efficient technique implemented by Wang *et al.* in an attempt to enhance the SRGAN structure [Tan18]. We first train a PSNR-oriented model and then use it for GAN training and fine-tuning. The derivative model G_{INT} is an interpolation between both networks G_{PSNR} and G_{GAN}, defined by the following parameters:

$$\theta_G^{INT} = (1 - \alpha)\, \theta_G^{PSNR} + \alpha\, \theta_G^{GAN}, \tag{3.9}$$

where θ_G^{INT}, θ_G^{PSNR} and θ_G^{GAN} represent the respective parameters of G_{INT}, G_{PSNR} and G_{GAN}, and $\alpha \in [0, 1]$ represents the interpolation parameter.

Network interpolation can be advantageous so far as using any meaningful value $\alpha \in]0, 1[$ can generate relatively noise-reduced results. Additionally, the parameter α can be continuously evaluated for quality assessment without re-training the models involved.

(a) Original image



(b) Feature maps

Figure 3.5: Feature maps before and after activation for a sample license plate image extracted from different VGG19 layers

4 Results and Technical Details

4.1 Dataset

The training data consists of a set of `png` images generated using the python imaging library `Pillow`[1]. We train two separate complex models[2] using two different sets of data, a number dataset and a German license plate dataset.
In the next subsections, we go through the specifics of each dataset.

4.1.1 Number Dataset

For a first dataset, we use a fundamental imagery template. The data samples produced are basic RGB images[3] of size 256x256. Each sample features a white background with a centered text (as depicted in Figure 4.1). The texts are randomly generated number sequences with a length of 10, a font size of 30, a font color black, and a font style `Charles Wright Bold`.

LR images are obtained from HR images using bicubic for down-sampling with a factor of 4, i.e. LR images are of size 64x64.

We generate 5000 HR samples for training and 500 HR samples for testing (LR image generation is part of the ESRGAN code). We also provide additional data samples with different properties for further testing purposes, which we discuss in section 4.3.

[1] Check `https://pillow.readthedocs.io/`
[2] Each complex model is an interpolation of a PSNR model and a ESRGAN model
[3] HR images

Figure 4.1: Random number sequence generated using the Charles Wright font

4.1.2 German License Plate Dataset

For a second dataset, we use a regular German license plate structure as a more complex template (Depicted in Figure 4.2). A regular car plate has five main components described as follows[4]:

1. **Flag**, we display the European Union flag. This component is part of a fixed background and is static throughout the whole dataset.

2. **Country**, the D underneath the EU flag stands for "Deutschland" or Germany. This component is also part of a fixed background and is static throughout the whole dataset.

3. **City or region**, by definition, this part contains 1 to 3 letters, a prefix representing the city or region. It also reflects the region size, as larger cities require fewer letters, and therefore more alphanumerics in the unique section of the plate. For our data samples, we utilize a randomly generated sequence of alphanumerics of length 1-3.

4. **Registration and safety seals**, two seals aligned vertically. The lower seal identifies the respective German state. The upper seal is the safety inspection

[4]Check `https://www.customeuropeanplates.com/german-license-plate-codes/`

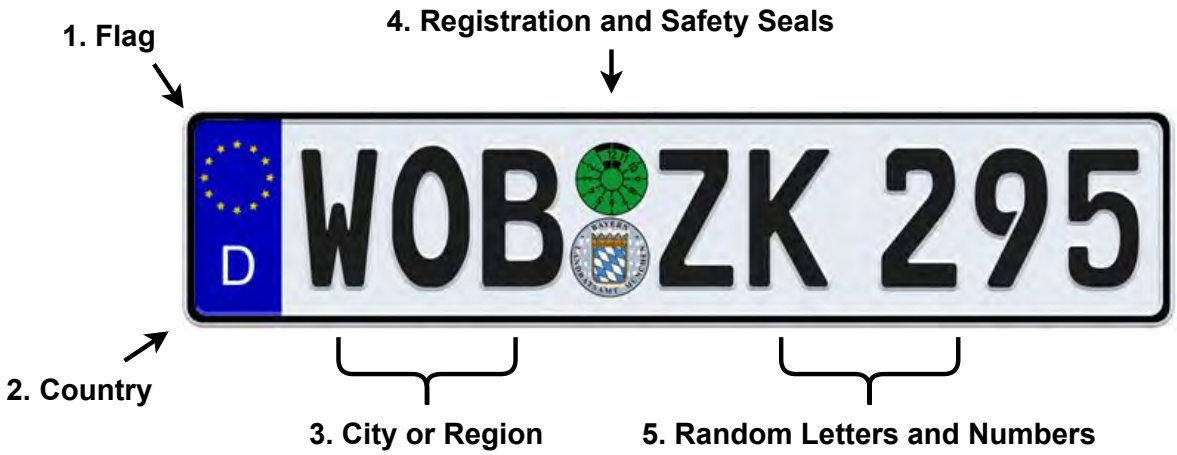


Figure 4.2: German license plate format

sticker. We use the green sticker (see Figure 4.3r) as the upper seal throughout the entire dataset. As lower part, we randomly select one of the existent 17 seals depicted in Figure 4.3

5. **Random letters and numbers**, the unique identifier for the vehicle. It usually contains 1 or 2 random letters followed by 1 to 4 digits. We use a randomly generated sequence of alphanumerics for our dataset. The length of the sequence depends on the length of the previously generated sequence in (3), in a way that the total length of the text equals 8.

Furthermore, the data samples produced are RGB images[5] of size 512x512. Each sample features a basic empty license plate of height 116 as background (parts 1 and 2 as depicted in Figure 4.2) with changing text and images for seals. The texts are randomly generated alphanumeric sequences with a length of 8, a font size of 70, a font color black, and a font style `Fälschungserschwerende Schrift`.

LR images are obtained from HR images using bicubic for down-sampling with a factor of 4, i.e. LR images are of size 128x128.

We also generate 5000 HR samples for training and 500 HR samples for testing (LR image generation is part of the ESRGAN code).

[5]HR images

(a) Bayern

(b) Baden

(c) Berlin

(d) German Government

(e) Hamburg

(f) Niedersachsen

(g) Rheinland Pfalz

(h) Saarland

(i) Sachsen Anhalt

(j) Sachsen Leipzig

(k) Schleswig

(l) German flag

(m) Thuringen

(n) US military seal

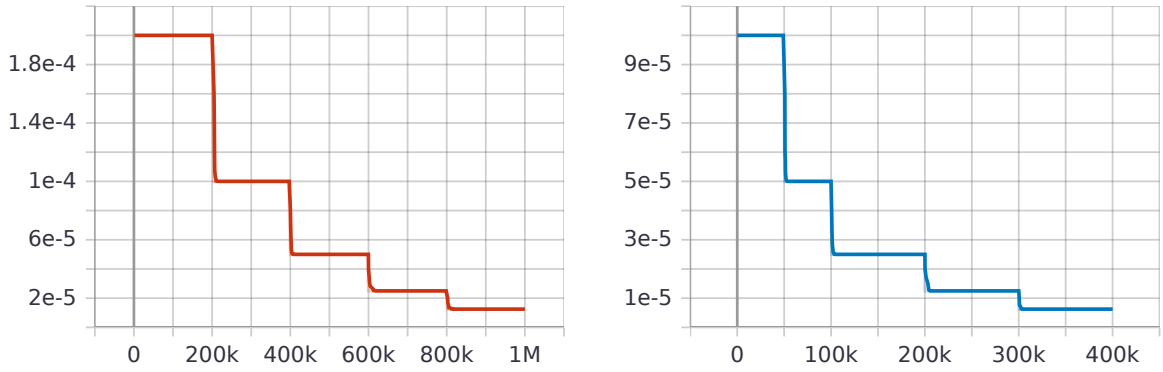
(o) Bundesfinanzverwaltung blue

(p) Bundesfinanzverwaltung red

(q) Bundesfinanzverwaltung white

(r) safety seal

Figure 4.3: German registration and safety seals



(a) Learning rate for PSNR model (b) Learning rate for ESRGAN model

Figure 4.4: Learning rate evolution during training

4.2 Training Details

We perform all our network trainings on a `GeForce RTX 3090` GPU with a version 11.2 `CUDA`. For network implementation, we use `Python 3.8.10` and `tensorflow v2.5.0`. We train four models in total (a PSNR model and an ESRGAN model for each dataset mentioned in section 4.1) and use the same values for common training parameters[6] as practiced in regular ESRGAN [Tan18].

We set the mini-batch size to 16 throughout the entire training phase. As a first step, we train a PSNR-based model with L_1 as loss function and set the number of iterations to $1 * 10^6$. The learning rate is initialized as $2 * 10^{-4}$ and halved every $2 * 10^5$ iterations (see Figure 4.4a).

The trained PSNR model is then utilized to initialize the GAN training as a second step. The number of iterations is set to $4 * 10^5$ and the learning rate is initialized as $1 * 10^{-4}$. The latter is then decreased by half every $5 * 10^4$ iterations (see Figure 4.4b). We use the loss function in the equation 3.8 to train the generator, and set the parameters to $\lambda = 5 * 10^{-3}$ and $\eta = 1 * 10^{-2}$.

Finally we apply Adam [Ba17], a technique for stochastic optimization, and set the associated parameters to $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

[6]Same parameter values for both datasets

4.3 Experimental Results

This section presents both qualitative and quantitative results obtained during our models' training and testing. We mainly focus on the final super-resolved outcome, i.e. all previously discussed enhancement methods are taken into consideration.

In a first part, we describe the results achieved in the first two models (PSNR and GAN) associated with the number dataset specified in subsection 4.1.1. This includes training results depicted in `Tensorboard`, testing results as generated SR images in comparison to the HR counterpart and the bicubic scaling, and interpolation results using a list of values for α (see section 3.4).

In a second part, we describe the results achieved in the second two models (PSNR and GAN) associated with the German license plate dataset specified in subsection 4.1.2. Similar to the first part, we include training results as `Tensorboard` charts, testing results as generated SR images in comparison to the HR counterpart and the bicubic scaling, and interpolation results using a list of α values (see section 3.4). We also compare our custom-model-generated license plates to a sample generated using a pre-trained ESRGAN [7].

4.3.1 Number-Dataset-Based Model

Training Results

We initially evaluate the PSNR-oriented model during training by plotting the loss function w.r.t. the processed mini-batches (see Figure 4.5). The obtained graph is a quasi-exponential decay, referencing a basic loss function behavior. The training starts with a loss value of ~ 0.17. The function then keeps on decreasing on average within the interval $[0, 200k]$. We then note a quasi-constant evolution modeled by *noisy levels or plateaux* within intervals $[\mu 200k, (\mu+1)200k], \mu \in \{1, 2, 3, 4\}$. The loss value at the end of the training is $2.4672e-4$.

In a second step, we commence the GAN training and depict the discriminator loss function (Figure 4.6). We observe after a transition phase within the interval $[0, 50k]$, a

[7] Check `https://github.com/peteryuX/esrgan-tf2`

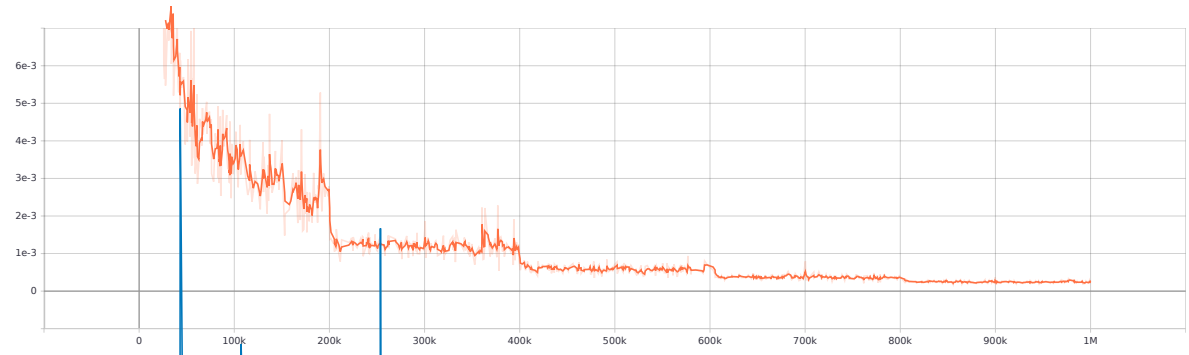


Figure 4.5: PSNR model total loss

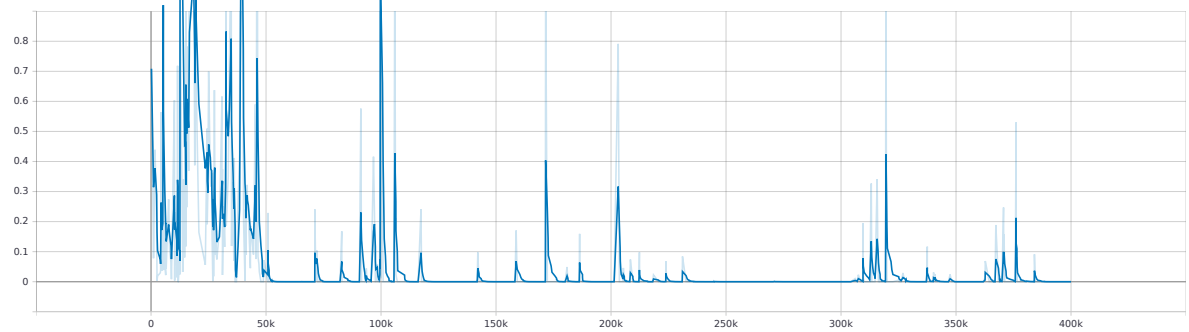


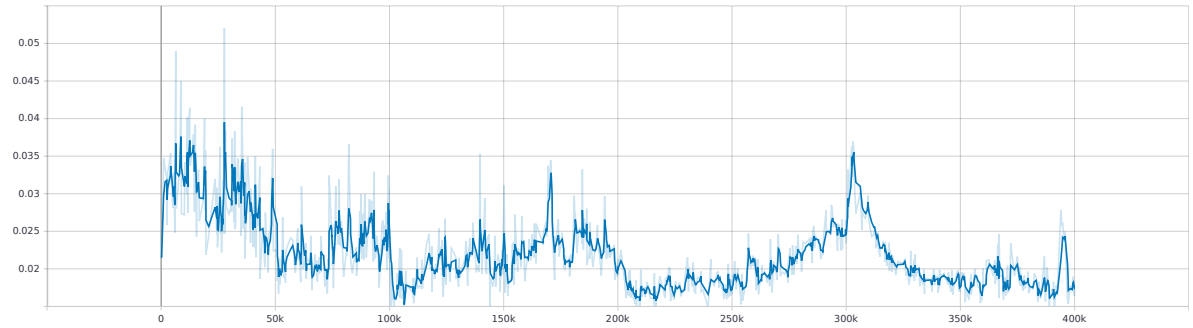
Figure 4.6: ESRGAN model discriminator loss

value stabilization featuring two main stages: a first stage where the loss is null or quasi-null and a second stage where the loss undergoes sudden jumps in value (e.g. 0.4237). As a matter of fact, the discriminator is used to classify both real and fake images. In case it labels a fake instance as real or a real instance as fake, the loss function receives a penalty, hence the noted local maxima.

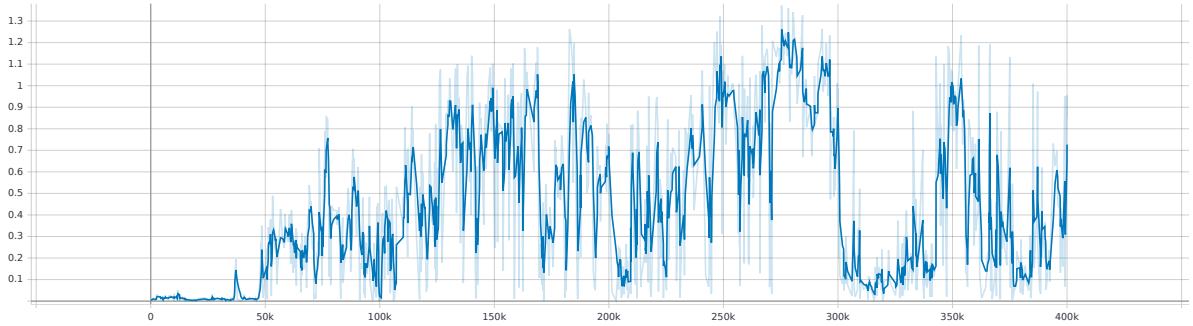
The generator performance is represented by four different plots as seen in Figure 4.7 (a plot for each term in the equation 3.8). We observe that each of the represented loss functions has a different scale with a domination of the adversarial loss (Figure 4.7b), hence the high similarity between the adversarial loss graph and the total loss graph[8].

The perceptual loss (Figure 4.7a) is set to a scale that gives it a support role without over-influencing the total loss. As for the pixel loss (Figure 4.7c), it has a scale of minimal influence, translating its secondary role.

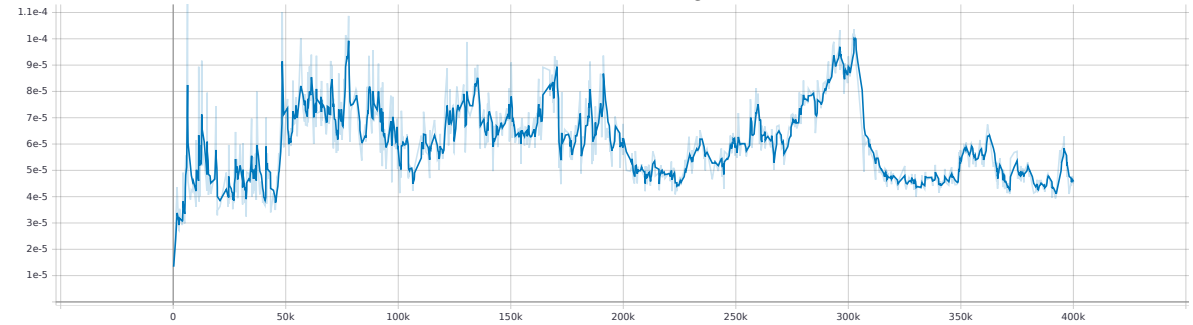
[8]Note that although the loss function isn't decreasing, this doesn't deny the model convergence.



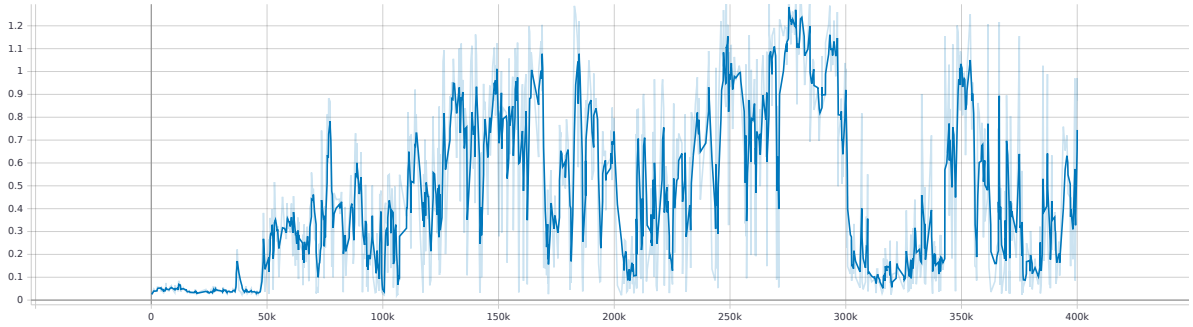
(a) Perceptual loss L_p



(b) Adversarial loss L_G^{RaGAN}



(c) Pixel loss L_1



(d) Total loss L_{GAN}

Figure 4.7: ESRGAN model generator loss

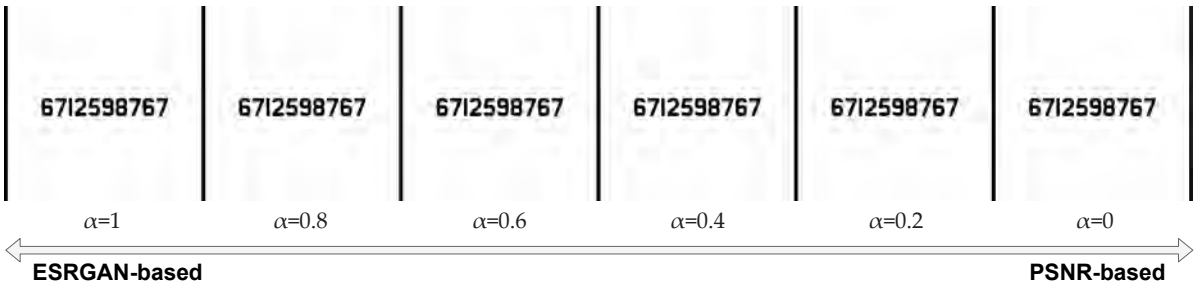


Figure 4.8: Representation of the network interpolation result for different α values

Testing Results

We represent the final result for the first set of data in the summarizing table 4.1, where we depict the bicubic interpolation, the PSNR-oriented model output, the ESRGAN model output, and the original HR image respectively. For the model assessment, we use the metrics PSNR and SSIM discussed in subsection 2.2.3. We observe for this example that the PSNR-oriented model output and the ESRGAN model output (second and third columns) are indistinguishable from the ground truth (fourth column). We also note that the pre-trained model slightly outperforms the ESRGAN model w.r.t. the measured PSNR, which is intuitive since it's mainly trained to maximize this value. Furthermore, we obtain the highest SSIM value for both models.

Network Interpolation Results

In order to evaluate the network interpolation method (see section 3.4), we depict the output of the combined models w.r.t. a set of α values. The figure 4.8 represents six different results for α varying in [1.0, 0.8, 0.6, 0.4, 0.2, 0.0]. We further discuss the results in chapter 5.

Additional testing Results

We further test the obtained model on data samples that slightly deviate from the number dataset specification. For this matter, we use two generated images having each exactly one modified property according to the dataset definition in subsection 4.1.1. The first example seen in the table 4.2 is a generated number sequence using the Arial font. The second example is a generated letter sequence using the Charles Wright font.

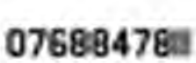
Bicubic	PSNR (Pretrain)	ESRGAN	Ground Truth
			
24.50dB/0.96	42.18dB/1.00	40.21dB/1.00	-
			
24.01dB/0.96	41.89dB/1.00	40.18dB/1.00	-
			
24.66dB/0.97	41.70dB/1.00	39.93dB/1.00	-
			
24.54dB/0.96	42.01dB/1.00	39.92dB/1.00	-

Table 4.1: Number-dataset result presentation: Comparison between different approaches and model assessment using the metrics PSNR/SSIM

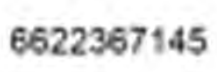
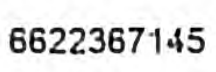
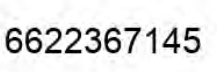
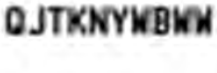
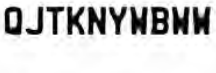
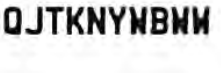
Bicubic	PSNR (Pretrain)	ESRGAN	Ground Truth
			
23.45dB/0.95	27.34dB/0.99	27.41dB/0.99	-
			
23.94dB/0.96	31.04dB/0.99	30.48dB/0.99	-

Table 4.2: Additional results featuring a number sequence in Arial font and a letter sequence in Charles Wright font: Comparison between different approaches and model assessment using the metrics PSNR/SSIM

Naturally, the noted PSNR and SSIM are lower than the prior testing result values. We also observe that the model achieves a better performance in the second example, since the obtained letters are both correct and distinguishable, and the output image is visually more appealing.

4.3.2 License-Plate-Based Model

Training Results

Similar to the previous dataset, we first assess the PSNR-oriented model during training by plotting the loss function against the processed mini-batches (see Figure 4.9). The resulting graph is a quasi-exponential decline, resembling the behavior of a simple loss function. The training begins with a loss value of ~ 0.18. The function then continues to decrease on average in the range $[0, 200k]$. We then note a quasi-constant evolution represented by *noisy levels or plateaux* within intervals $[\mu 200k, (\mu+1)200k], \mu \in \{1, 2, 3, 4\}$.

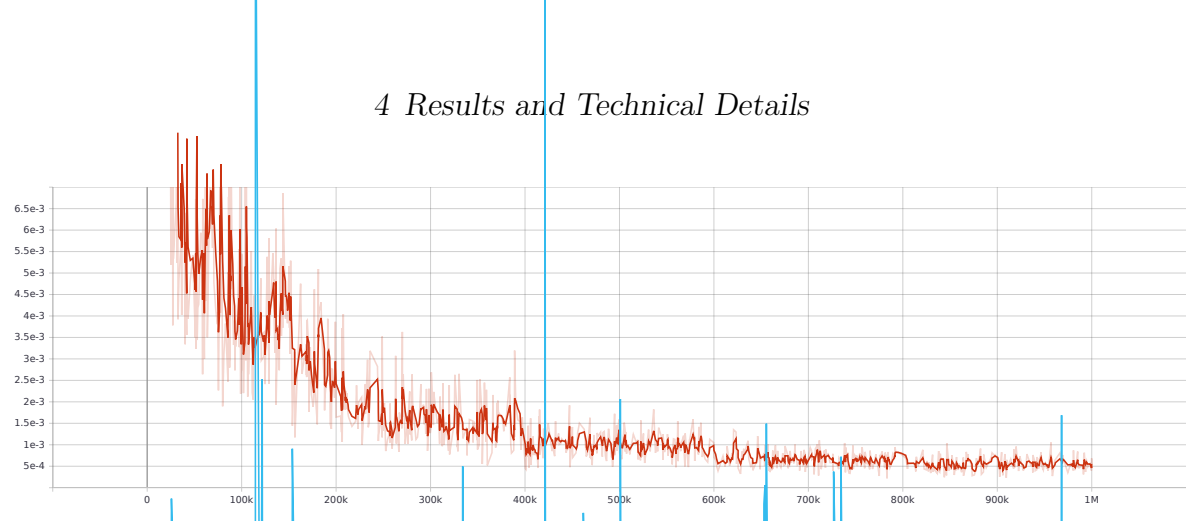


Figure 4.9: PSNR model total loss

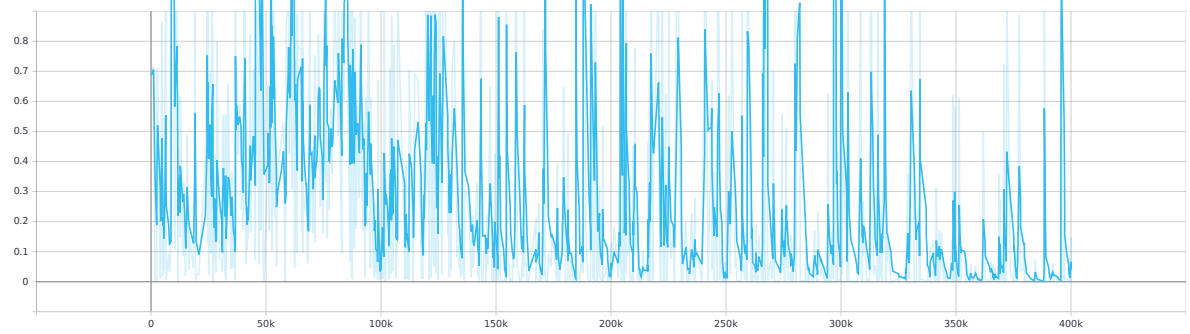


Figure 4.10: ESRGAN model discriminator loss

The loss value at the end of the training is $5.4344e-4$. We also observe slightly higher values on average compared to the previous PSNR model.

As soon as the PSNR model training is over, we can start training the GAN model. We represent in an independent plot the evolution of the discriminator loss w.r.t. the defined steps (see Figure 4.10). Compared to the previous discriminator chart (Figure 4.6), the current chart highlights remarkable distortions. A transition phase is still noted between steps 0 and $100k$. The minima and maxima then start to decrease, accompanied by a less prominent distortion. We note a convergence at the end of the training with a steadier state translated in more values closer to null and lower maxima (between steps $350k$ and $400k$).

We also depict the current generator performance in four different graphs seen in Figure 4.11 (a plot for each term in the equation 3.8). Unlike the previous model, the scales associated to both the perceptual loss L_p (Figure 4.11a) and the adversarial loss L_G^{RaGAN} (Figure 4.11b) are comparable. Furthermore, we note the decreasing aspect on average of the feature loss and the increasing aspect on average of the adversarial loss. Therefore, The total loss is over-influenced by L_p during the first steps of training, and by L_G^{RaGAN}

during the last steps. The pixel loss L_1 (Figure 4.11c) still has a scale of minimal influence by choice.

Testing Results

Similarly, the table 4.3 summarizes the final result for the second dataset. We also depict the bicubic interpolation, the PSNR-oriented model output, the ESRGAN model output, and the original HR image respectively, and use the metrics PSNR and SSIM discussed in subsection 2.2.3 for the model assessment. In this example, the PSNR-oriented model output and the ESRGAN model output (second and third columns) are rather distinguishable from the ground truth (fourth column) and from each other, where the difference appears in smaller details (state flags and safety seals). The pre-trained model still outperforms the ESRGAN model w.r.t. the measured PSNR, and we obtain high values for the SSIM in the given samples.

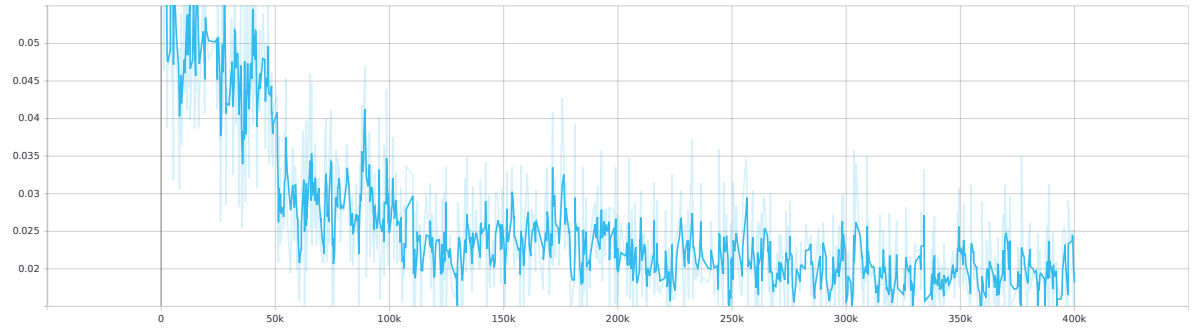
Network Interpolation Results

In parallel to the previous set of data, we depict the output of the combined models relevant to the license plate dataset w.r.t. different α values. The figure 4.12 represents six distinct outputs for α varying in [1.0, 0.8, 0.6, 0.4, 0.2, 0.0]. In Chapter 5, we go over the results in more detail.

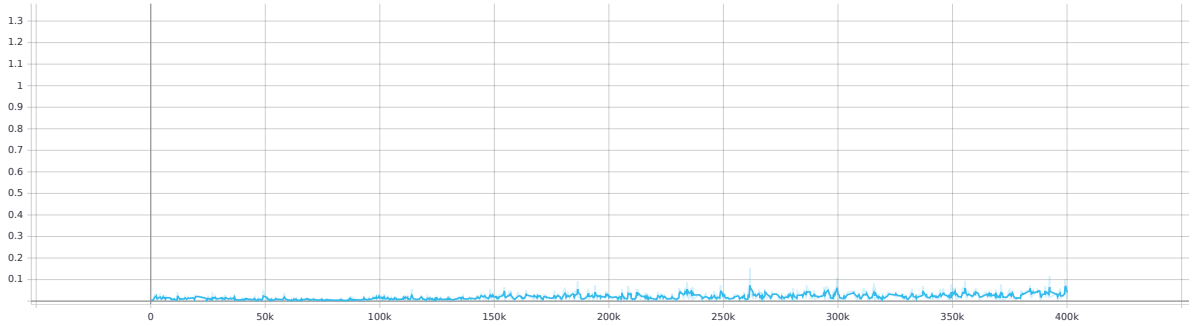
Pre-trained model Results

Finally, we illustrate in the table 4.4 the SR output license plates using a pre-trained ESRGAN. The loaded model is trained on a DIV2K[9] dataset, a famous high-quality dataset of 1000 images often used for Super-Resolution and image restoration purposes. We further discuss the result details in chapter 5.

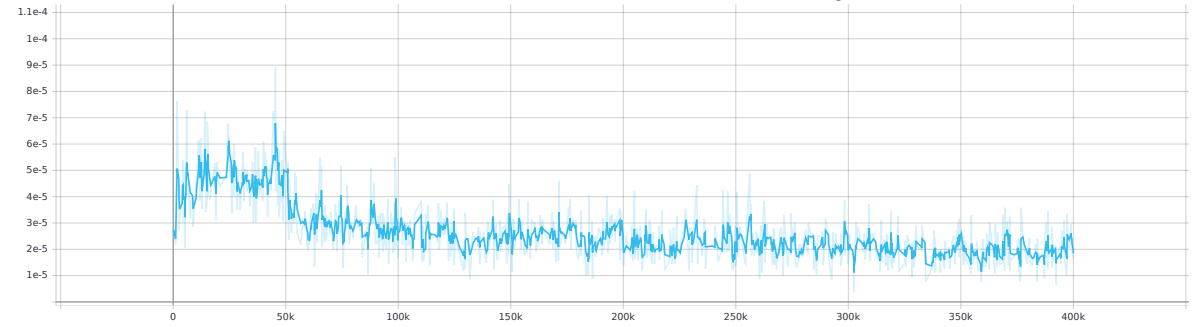
[9] `https://data.vision.ee.ethz.ch/cvl/DIV2K/`



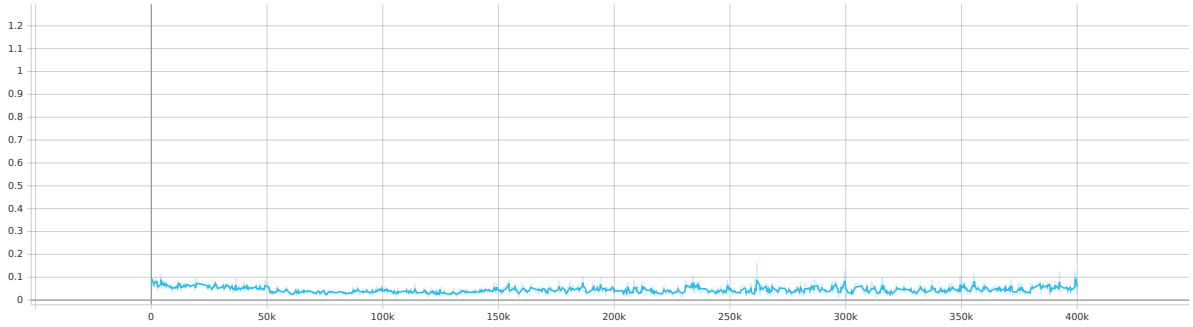
(a) Perceptual loss L_p



(b) Relativistic average discriminator loss L_G^{RaGAN}



(c) Pixel loss L_1



(d) Total loss L_{GAN}

Figure 4.11: ESRGAN model generator loss

Bicubic	PSNR (Pretrain)	ESRGAN	Ground Truth
			
24.51dB/0.94	35.67dB/0.99	35.51dB/0.99	-
			
25.00dB/0.94	36.78dB/0.99	36.30dB/0.99	-
			
24.68dB/0.94	36.37dB/0.99	36.00dB/0.99	-
			
24.75dB/0.94	36.20dB/0.99	35.87dB/0.99	-

Table 4.3: License-plate-dataset result presentation: Comparison between different approaches and model assessment using the metrics PSNR/SSIM

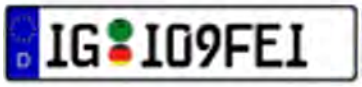
Bicubic	ESRGAN	Ground Truth
		
24.75dB/0.94	29.00dB/0.97	-
		
24.68dB/0.94	28.91dB/0.97	-
		
25.00dB/0.94	29.27dB/0.98	-
		
24.51dB/0.94	28.80dB/0.97	-

Table 4.4: Results using a pre-trained ESRGAN: Comparison between different approaches and model assessment using the metrics PSNR/SSIM

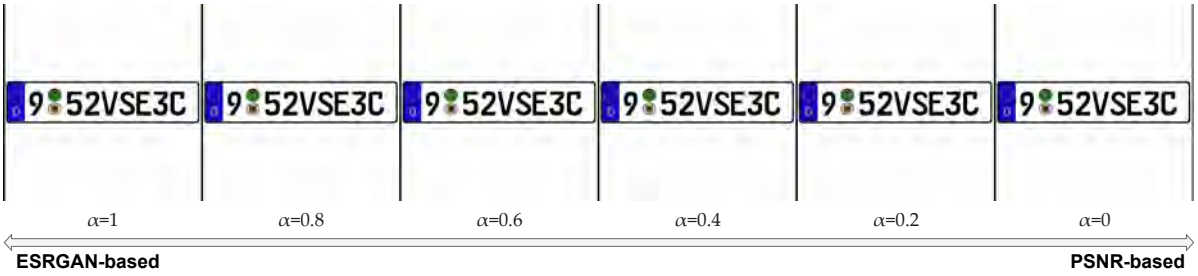


Figure 4.12: Representation of the network interpolation result for different α values

5 Discussion

In this chapter, we closely examine the outcome of our experiment (chapter 4). We navigate through the obtained result of each phase and formulate a comparative description between the models associated with the first and second datasets respectively. The model comparison will be accompanied by visual support in the form of magnified windows of previously inserted SR images. We also discuss the pre-trained model result and explain the returned behavior as well as the major difference in comparison to our custom-trained network. For simplicity purposes, we will refer to the *number-dataset-based model* as *number model* and to the *license-plate-based model* as *plate model*. Further visual support is found in the Appendix A.

5.1 Training Results

5.1.1 PSNR Model Training

During the PSNR model pre-training, we note a quasi-identical performance from both models. The comparative Figure A.1 displays a twin decreasing behavior of the L_1 loss, with the plate model returning slightly higher values. This is expected behavior since the second dataset is more complex, therefore has more features to learn.

5.1.2 ESRGAN Model Training

In this subsection, we focus on the training result of the defined ESRGAN model (see chapter 3). We first steer our discussion into the represented discriminator loss and adversarial loss in order to examine the GAN performance independently. In a second

steps we consider the *secondary* loss functions, i.e. the perceptual loss L_p and the pixel loss L_1.

First and foremost, we need to explain the GAN training process. Both the generator and discriminator are trained simultaneously. Their respective loss functions are derived from the same formula, but the loss values are complementary. In a scenario where the discriminator is penalized for misclassifying a data sample, its recorded loss is forwarded to the generator as compensation. Therefore, if the discriminator has a large loss rate, the generator loss should be low and vice versa. The final goal behind the training process is to minimize the sum of both losses.

In figures A.2 and A.3, we observe that the GAN behaviors for both models are opposite of each other. The number model presents low discriminator loss values, therefore high generator loss. This might lead to the conclusion that for such a trivial dataset, GAN training can be overlooked. The discriminator seems to converge after 1/8 of the training time, and the generator is bound to be unstable throughout the majority of the defined steps. Therefore, the PSNR model is enough to train such a model.

The plate model on the other hand displays a rather standard GAN behavior. The generator exhibits minimal and consistent loss values during the training procedure. Oppositely, the discriminator sets out a conventional learning pattern. We divide the discriminator training process into three different phases w.r.t. the specified steps:

- **Between steps 0 and 100*k*:** A totally distorted loss function with high maxima and high minima, a transition phase for the discriminator.
- **Between steps 100*k* and 300*k*:** The learning phase. The discriminator loss function displays high maxima and rather low minima.
- **Between steps 300*k* and 400*k*:** Low minima and maxima values indicate the model's quasi-convergence.

As a consequence, the model's convergence is represented by a quasi-steady generator loss and a decreasing discriminator loss on average.

As previously mentioned, the number model ESRGAN training is hardly affected by the terms appearing in the equation 3.8, including the feature loss L_p and the pixel loss L_1, since the PSNR pre-training already achieves the desired results. This can be reflected by non-decreasing perceptual and pixel losses. Whereas these functions may

be significant for the plate model, especially at the beginning of the GAN training. In Figures A.3a and A.3c, we note a diminishing loss, accentuated in the interval $[0, 100k]$. This leads to an increase of the total loss (Figure A.3d) resulting in a quicker transition phase.

5.2 Testing Results

To evaluate the ESRGAN plate model performance, we provide magnified windows of an acquired SR image and its HR counterpart in Figure 5.1. We select for the purpose a box containing both the safety and registration seals, as well as a single character or alphanumeric[1]. We note that the custom-trained model provides a close-to-identical output on the alphanumeric character level. Both letters *B* appearing in the magnified boxes are indistinguishable to the human eye. We also note that the acquired seals are blurry copies of the original. This is mainly a consequence of the limited resources and the relatively low resolution of the training dataset seals, to begin with. The generator didn't add any new outlines it didn't learn. Instead, it applied the predefined parameters to two plate components with different resolutions.

In light of the supplied dataset, we may infer that the developed model performs as intended. The super-resolution is mostly required for the combination of numerals and letters, [2], since it's the component that makes the license plate unique. The super-resolved seals might not be of high quality, yet they're still identifiable due to the reduced number of possibilities[3] (17 as depicted in Figure 4.3).

5.3 Interpolation Results

We further depict in Figure 5.2 magnified windows of the acquired interpolation result for the values $\alpha = 1$ (ESRGAN model), $\alpha = 0.6$ (hybrid model) and $\alpha = 0$ (PSNR model) respectively.

[1] Without loss of generality
[2] We count 36^8 combinations as defined
[3] We note a possibility to be a combination of the safety seal and a state/German seal

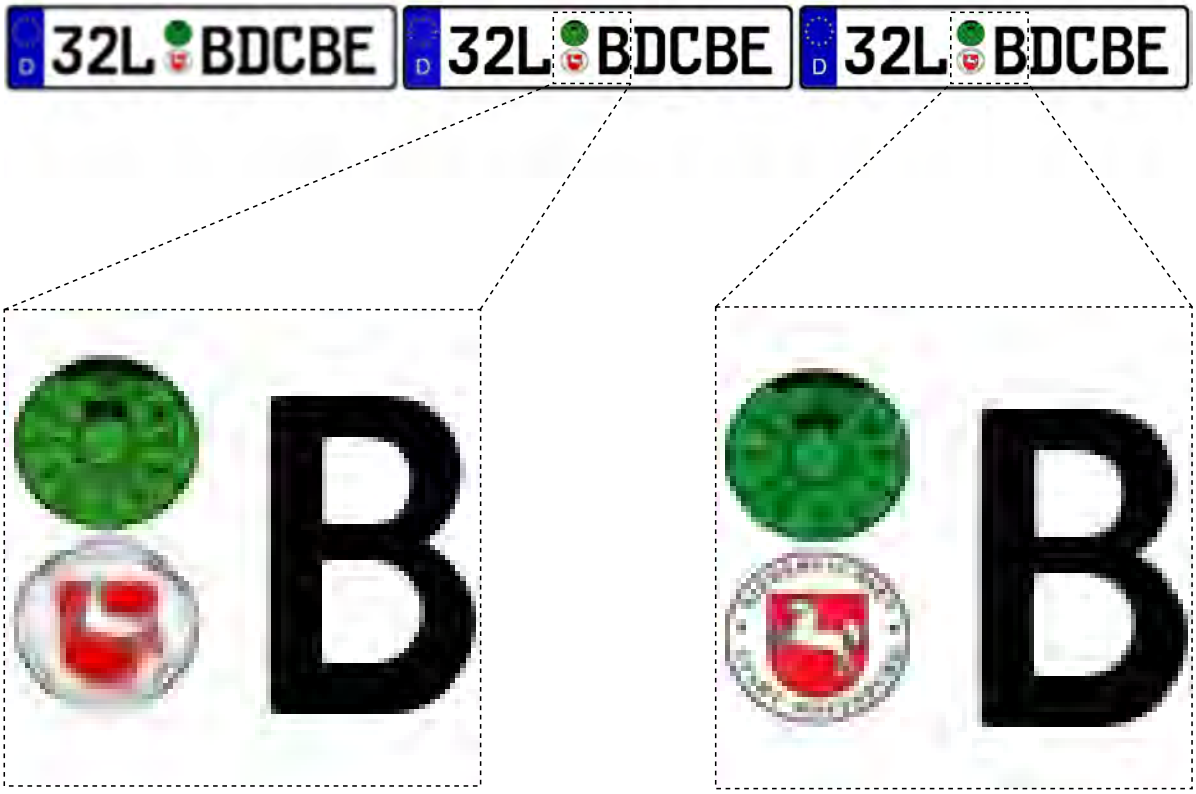


Figure 5.1: Magnified windows of the custom-trained ESRGAN model results

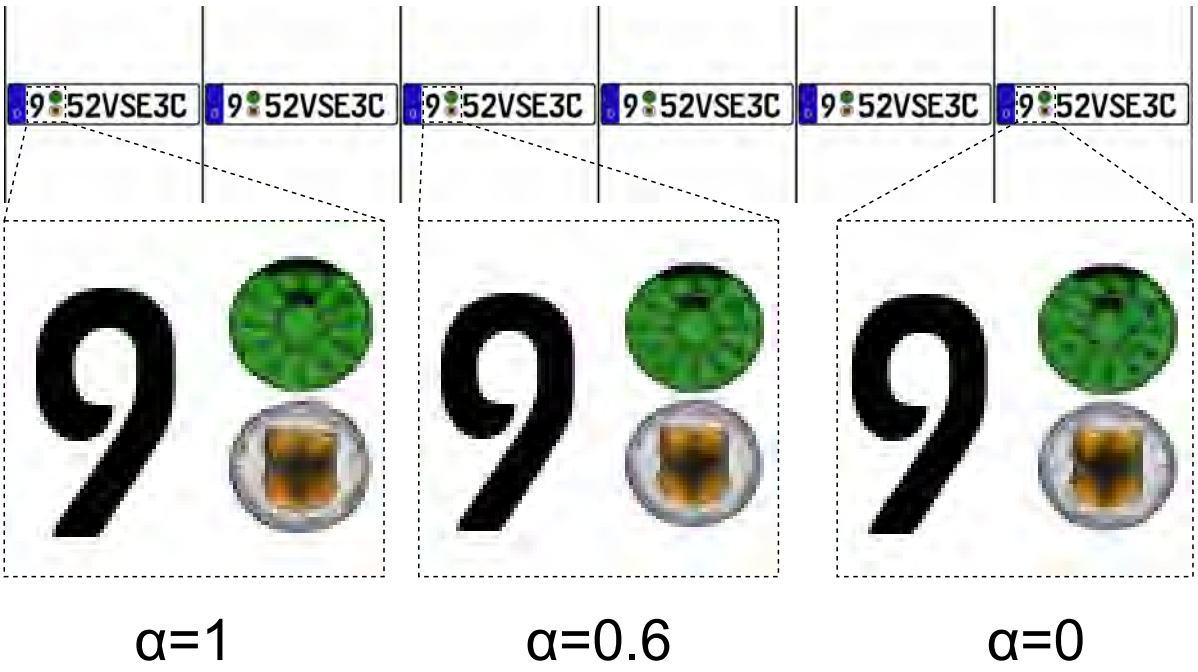


Figure 5.2: Magnified windows of the network interpolation result for different α values

A first trivial observation would be that the network interpolation does not effect the displayed digit. We therefore focus our attention on the safety and state seals.

The differences between the shown safety seals are minor and may be reduced to a change in color brightness (ESRGAN model delivers a brighter green color). The state seal on the other hand has more distinct features. The ESRGAN model ($\alpha = 1$) establishes over-the-top high-frequency edges that may as well be superfluous for a simple classification task with few classes at hand. For $\alpha = 0$, a pure PSNR model begins to exhibit minor flaws, in a way that we can detect undesirable artifacts being introduced to the SR output. Network interpolation offers a fair compromise between both models, where unwanted artifacts can be removed or reduced, with a liberty of navigating between smooth and sharp edges.

5.4 Pre-trained Model Results

We conclude this section with an evaluation of the loaded model trained on the DIV2K dataset. The results in section 4.3 show that the model under-performs in general (1/10 the PSNR compared to the custom-trained model). Figure 5.3 displays modified super-resolved edges on the curved side of the letter *B*. Furthermore, the obtained safety seal develops new inner edges. The state seal script (state name) is deleted and a number of its specific features are also removed. In summary, despite the loaded ESRGAN being trained on data samples possessing higher resolution and richer features, we notice the difference in performance.

The nature of the training dataset highly affects the ESRGAN performance. A model with the task of front-view face generation would produce human-like features if applied to a low-resolution animal or object image. This behavior is mainly leveraged in image translation [DC20] where input is required to gain learned features from the training dataset (e.g. translating a zebra to a horse, translating summer to winter, etc.).

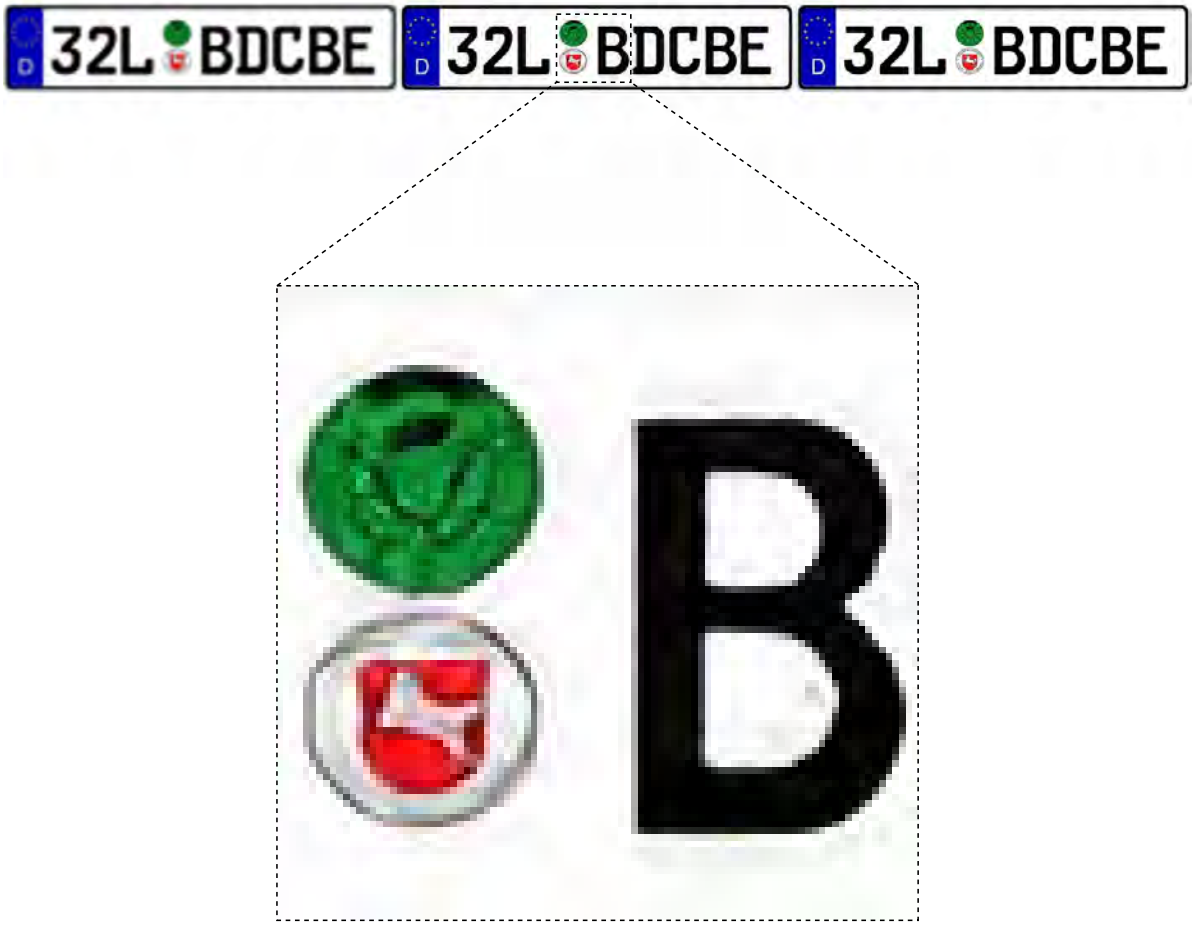


Figure 5.3: Magnified windows of the pre-trained ESRGAN model results

6 Conclusion

With Single Image Super-Resolution being one of the most attractive topics during the past two decades, numerous research papers have been written, algorithms have been developed, taking advantage of previous work while trying to correct or enhance discovered flaws or weak connection nodes in the built architectures. All with the sole purpose of delivering a state-of-the-art model.

In this thesis, we exploit the Enhanced Super-Resolution Generative Adversarial Network, a model developed by Wang *et al.* [Tan18] and based on the pioneer work done by Ledig *et al.* in the Super-Resolution Generative Adversarial Network (SRGAN) paper [Twi17]. In this work, a more sophisticated architecture is used for the generator, based on a chain of customizable blocks. The used block is a Residual-in-Residual Dense Block (RRDB), which is a complex neural network featuring two state-of-the-art designs, namely the dense block and residual block with Batch Normalization (BN) removal, and benefiting from the residual scaling technique. The discriminator is given a relativistic aspect implementing the Relativistic average Generative Adversarial Network (RaGAN) technique instead of an absolute comparison for real and fake data. Furthermore, the loss functions gain two additional terms. The perceptual loss helps give the output image better perceptual qualities at the expense of a high PSNR, white a weighted L_1 loss aims to keep a certain balance between sharp edges and undesired artifacts.

The model also offers a compromise between a PSNR-based model and a GAN-based model via the network interpolation technique without the need for re-training.

We train the network on two custom-created datasets. The first dataset is a straightforward dataset defined as samples of RGB images with a white background and a centered randomly-generated number sequence of length 10. The second dataset is more complex, constructed to match a standard German license plate. For generating the data samples, we applied a customary car-license-plate frame, a copy of the green safety seal, and

copies of all available state seals. For the identification, we used two randomly generated alphanumeric sequences with a total length of 8.

With a scaling factor of 4 and a count of 5000 samples for training, the model overperforms on the first defined dataset. The purpose behind this phase was to test the model's performance on a set of data created exclusively for the specified task.

The second dataset is meant to introduce more complicated outlines, with different shapes and colors. As expected, the model still outperforms at scaling the alphanumeric sequence, meaning the part of the license plate granting its unicity. The model also delivers a distinguishable set of seals, another changing aspect of the plate, yet far from being given the label *high quality*. This is mainly due to the limited resources used to create the dataset. Such a network is usually trained on $2k$ resolution images. Hence, we need to provide sample elements with higher resolution.

Nonetheless, the model proved its capability, and with appropriate training data, we could aim for a scaling factor of 8.

Future Work

As previously mentioned, the next step would be to train the model using a High-Resolution image dataset. The scaling factor could be increased to 8 instead of 4 with the corresponding training parameters adapted accordingly.

Going back to the CSI use case mentioned in Chapter 1, a clear shot at the license plate isn't always guaranteed, making the frames used for training the model rarely the case at hand. A solution would be to pair the SR model with a trainable *Adaptive Threshold* algorithm as a precursor. The aim of this algorithm is to separate the desired license frame from its background.

A Training Graphs

Figure A.1: PSNR loss: a model comparison

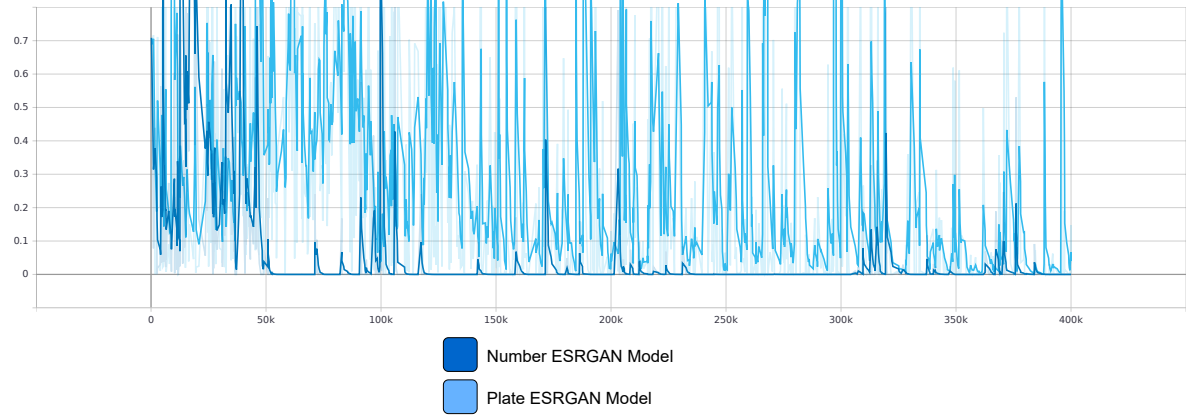
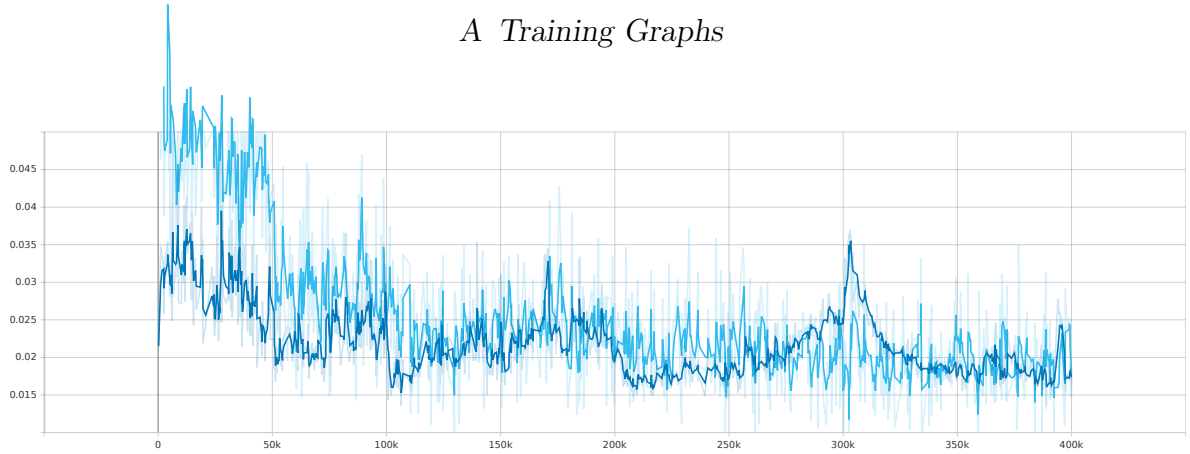
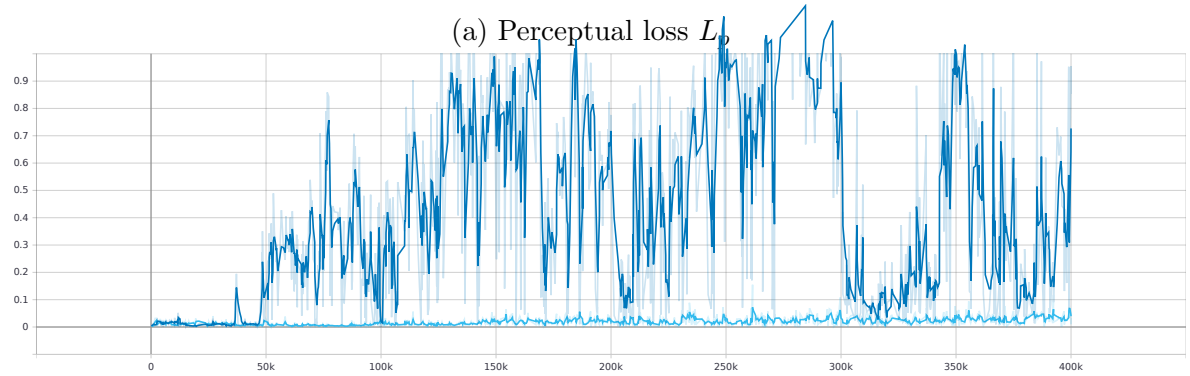


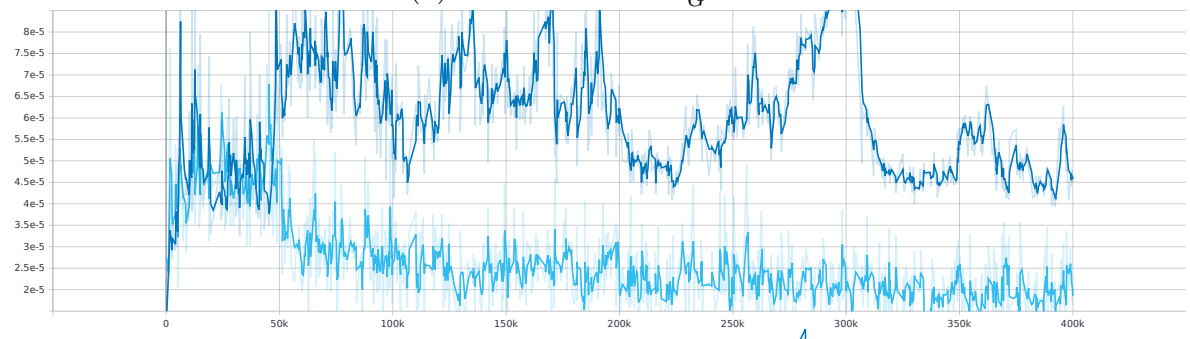
Figure A.2: ESRGAN discriminator loss: a model comparison



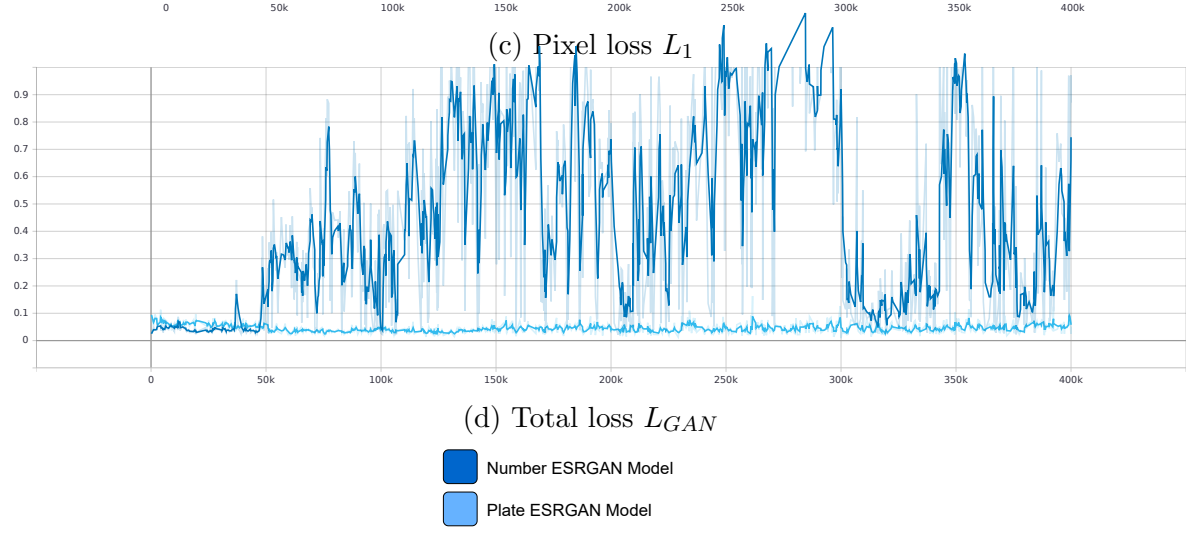
(a) Perceptual loss L_p



(b) Adversarial loss L_G^{RaGAN}



(c) Pixel loss L_1



(d) Total loss L_{GAN}

Figure A.3: ESRGAN generator loss: a model comparison

Acronyms

AI	Artificial Intelligence.
BN	Batch Normalization.
CNN	Convolutional Neural Network.
CSI	Crime Scene Investigation.
DenseNet	Dense Convolutional Network.
DL	Deep Learning.
DRCN	Deeply-Recursive Convolutional Network.
EDSR	Enhanced Deep Residual Networks for Single Image Super-Resolution.
ESRGAN	Enhanced Super-Resolution Generative Adversarial Network.
GAN	Generative Adversarial Network.
HR	High-Resolution.
ILSVRC	ImageNet Large Scale Visual Recognition Challenge.
LLE	Locally Linear Embedding.
LR	Low-Resolution.
LReLU	Leaky Rectified Linear Activation.
ML	Machine Learning.

MRI	Magnetic Resonance Imaging.
MSE	Mean Squared Error.
NLP	Natural Language Processing.
NN	Neural Network.
PSNR	Peak Signal-to-Noise Ratio.
RaGAN	Relativistic average Generative Adversarial Network.
ReLU	Rectified Linear Activation.
ResNet	Residual Network.
RGAN	Relativistic Generative Adversarial Network.
RRDB	Residual-in-Residual Dense Block.
SGAN	Standard Generative Adversarial Network.
SGD	Stochastic Gradient Descent.
SISR	Single Image Super-Resolution.
SR	Super-Resolution.
SRCNN	Super-Resolution Convolutional Neural Network.
SRGAN	Super-Resolution Generative Adversarial Network.
SRResNet	Super-Resolution Residual Network.
SSIM	Structural Similarity Index.
VGG	Visual Geometry Group.

Bibliography

[Asi08] Hassan Aftab; Atif Bin Mansoor; Muhammad Asim. "A New Single Image Interpolation Technique For Super Resolution". In: (Dec. 2008) (cit. on pp. 2, 17).

[AN02] A. J.Tatem; H.G. Lewis; P.M. Atkinson and M.S. Nixon. "Super-Resolution Mapping of Urban Scenes from IKONOS Imagery Using a Hopfield Neural Network". In: *IEEE 2001 International Geoscience and Remote Sensing Symposium* (Aug. 2002), pp. 3203–3205 (cit. on p. 1).

[Ba17] Diederik P. Kingma; Jimmy Lei Ba. "Adam: A Method For Stochastic Optimization". In: *arXiv:1412.6980v9* (Jan. 2017) (cit. on p. 34).

[Bet15] Leon A. Gatys; Alexander S. Ecker; Matthias Bethge. "Texture Synthesis Using Convolutional Neural Networks". In: *arXiv:1505.07376v3* (Nov. 2015) (cit. on p. 19).

[Cho18] François Chollet. *Deep Learning with Python*. Manning Publications, 20 Baldwin Road, PO Box 761, Shelter Island, NY 11964, 2018 (cit. on pp. 6, 7, 12).

[Cmg21] Cmglee. *Bicubic Interpolation*. 2021. URL: `https://en.wikipedia.org/wiki/Bicubic_interpolation` (cit. on p. 16).

[Coo21] Gordon Cooper. *Enhance Image! Real-time Super Resolution with ARC EV Processor IP*. 2021. URL: `https://www.synopsys.com/designware-ip/technical-bulletin/super-resolution-with-arc-ev.html` (cit. on p. 15).

[CB14] Ian J. Goodfellow; Jean Pouget-Abadie; Mehdi Mirza; Bing Xu; David Warde-Farley; Sherjil Ozair; Aaron Courville and Yoshua Bengio. "Generative Adversarial Nets". In: *arXiv:1406.2661v1* (June 2014) (cit. on p. 12).

[Das17] Siddharth Das. *CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more...* 2017. URL: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 (cit. on p. 8).

[Dee18] Deeplizard. *Batch Normalization ("Batch Norm") Explained*. 2018. URL: https://deeplizard.com/learn/video/dXB-KQYkzNU (cit. on p. 14).

[DC20] Hajar Emami; Majid Moradi Aliabadi; Ming Dong and Ratna Babu Chinnam. "SPA-GAN: Spatial Attention GAN for Image-to-Image Translation". In: *arXiv:1908.06616v3* (Dec. 2020) (cit. on p. 52).

[Fei16] Justin Johnson; Alexandre Alahi; Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *arXiv:1603.08155v1* (Mar. 2016) (cit. on pp. 19, 27).

[Gop15] Ghuatam; Das; M Gopi. "Curvature minimizing depth interpolation for intuitive and interactive space curve sketching". In: *Computer Graphics International (CGI), New York, USA* (June 2015) (cit. on p. 17).

[GM12] Marco Bevilacqua; Aline Roumy; Christine Guillemot and Marie-Line Alberi Morel. "Low-complexity single-image super-resolution based on nonnegative neighbor embedding". In: *British Machine Vision Conference* (2012) (cit. on p. 2).

[HT15] Chao Dong; Chen Change Loy; Kaiming He and Xiaoou Tang. "Image Super-Resolution Using Deep Convolutional Networks". In: (July 2015), pp. 1–14 (cit. on pp. 2, 17, 19).

[HA78] H.S Hou and H.C Andrews. "Cubic splines for image interpolation and digital filtering". In: *IEEE Transactions Acoustics, Speech, Signal Processing* (1978), pp. 508–517 (cit. on p. 17).

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167v3* (Mar. 2015) (cit. on p. 14).

[Jol18] Alexia Jolicoeur-Martineau. "The relativistic discriminator: a key element missing from standard GAN". In: *arXiv:1807.00734v3* (Sept. 2018) (cit. on pp. 12, 25, 26).

[Kak11] Emil Bilgazyev; Boris Efraty; Shishir K. Shah; Ioannis A. Kakadiaris. "Improved Face Recognition Using Super-Resolution". In: *IEEE RFID Virtual Journal* (Dec. 2011) (cit. on p. 1).

[Kre19] Prof. Dr. Ralf Krestel. *CNN-for-Images*. University of Passau, 2019 (cit. on p. 7).

[LL16] Jiwon Kim; Jung Kwon Lee and Kyoung Mu Lee. "Deeply-Recursive Convolutional Network for Image Super-Resolution". In: *arXiv:1511.04491* (Nov. 2016) (cit. on p. 18).

[LB15] S. Schulter; C. Leistner and H. Bischof. "Fast and Accurate Image Upscaling With Super-Resolution Forests". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 3791–3799 (cit. on p. 17).

[LO04] Xin Li and Michael T. Orchard. "New Edge-Directed Interpolation". In: *IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 10, NO. 10* (Oct. 2004), pp. 1521–1527 (cit. on p. 17).

[Maa18] Gao Huang; Zhuang Liu; Laurens van der Maaten; Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *arXiv:1608.06993v5* (Jan. 2018) (cit. on pp. 10, 11).

[NL17] Bee Lim; Sanghyun Son; Heewon Kim; Seungjun Nah and Kyoung Mu Lee. "Enhanced Deep Residual Networks for Single Image Super-Resolution". In: *arXiv:1707.02921* (July 2017) (cit. on pp. 18, 23).

[NS17] R. Dahl; M. Norouzi; and J. Shlens. "Pixel recursive super resolution". In: *arXiv preprint arXiv:1702.00783* (Mar. 2017) (cit. on pp. 3, 19).

[ON15] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *arXiv:1511.08458v2* (Dec. 2015) (cit. on p. 7).

[Rue14] Kanwal K. Bhatia; Anthony N. Price; Wenzhe Shi; Jo V. Hajnal; Daniel Rueckert. "Super-Resolution Reconstruction of Cardiac MRI Using Coupled Dictionary Learning". In: *IEEE International Symposium on Biomedical Imaging* (July 2014), pp. 947–950 (cit. on p. 1).

[SP15] J. Salvador and E. Pérez-Pellitero. "Naive Bayes Super-Resolution Forest". In: *IEEE International Conference on Computer Vision (ICCV)* (Dec. 2015), pp. 325–333 (cit. on p. 17).

[San13] Anil B Gavade; Prasana Sane. "Super Resolution Image Reconstruction By Using Bicubic Interpolation". In: *Advanced Technologies in Electrical and Electronic Systems* (Nov. 2013), pp. 204–209 (cit. on pp. 15, 16).

[Sim18] Osvaldo Simeone. "A Very Brief Introduction to Machine Learning With Applications to Communication Systems". In: *arXiv:1808.02342v4* (Nov. 2018) (cit. on p. 5).

[Sim03] Zhou Wang; Alan Conrad Bovik; Hamid Rahim Sheikh; Eero P. Simoncelli. *The SSIM Index for Image Quality Assessment*. 2003. URL: `https://www.cns.nyu.edu/~lcv/ssim/#MAD` (cit. on p. 20).

[Sim04] Zhou Wang; Alan Conrad Bovik; Hamid Rahim Sheikh; Eero P. Simoncelli. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* (Apr. 2004), pp. 600–612 (cit. on pp. 2, 19).

[Sun15] Kaiming He; Xiangyu Zhang; Shaoqing Ren; Jian Sun. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385v1* (Dec. 2015) (cit. on p. 10).

[Tan18] Xintao Wang 1; Ke Yu 1; Shixiang Wu 2; Jinjin Gu 3; Yihao Liu 4; Chao Dong 2; Chen Change Loy 5; Yu Qiao 2; Xiaoou Tang. "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks". In: (Sept. 2018), pp. 1–23 (cit. on pp. iv, 2, 3, 18, 19, 21–23, 25, 27, 28, 34, 54).

[Twi17] Christian Ledig; Lucas Theis; Ferenc Huszár; Jose Caballero; Andrew Cunningham; Alejandro Acosta; Andrew Aitken; Alykhan Tejani; Johannes Totz; Zehan Wang; Wenzhe Shi Twitter. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: (May 2017), pp. 1–19 (cit. on pp. iv, 1–3, 17–19, 21–23, 27, 54).

[VB16] Annina Simon; Mahima Singh Deo; S. Venkatesan and D.R. Ramesh Babu. "An overview of machine learning and its applications". In: *EJESE, Volume 1, Issue 1* (Jan. 2016), pp. 22–24 (cit. on p. 5).

[X14] Cui; Z.; Chang; H.; Shan; S.; Zhong; B.; Chen; X. "Deep network cascade for image super-resolution". In: *European Conference on Computer Vision* (2014), pp. 49–64 (cit. on p. 2).

[Xie] Dr. Yao Xie. "Lecture 4: Data-processing, Fano". In: *ECE587, Information Theory, Duke University* (), pp. 1–24 (cit. on p. 1).

[Xio04] Hong Chang; Dit-Yan Yeung; Yimin Xiong. "Super-Resolution Through Neighbor Embedding". In: *IEEE Xplore* (July 2004) (cit. on p. 17).

[Yu18] Weifeng Ge; Bingchen Gong; Yizhou Yu. "Image Super-Resolution via Deterministic-Stochastic Synthesis and Local Statistical Rectification". In: *arXiv:1809.06557v1* (Sept. 2018) (cit. on p. 1).

[ZS15] Andrew Zisserman and Karen Simonyan. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (Apr. 2015) (cit. on pp. 2, 8, 18).

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, September 2, 2021

IHEB CHHIBI