TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANSSE SAARIMÄKI

SINGLE IMAGE SUPER-RESOLUTION USING CONVOLUTIONAL NEURAL NETWORKS

Master's thesis

Examiner: Professor Karen Eguiazarian (Egiazarian)

The examiner and topic of the thesis were approved on 31 October 2018

ABSTRACT

ANSSE SAARIMÄKI: Single Image Super-Resolution Using Convolutional Neural Networks
Tampere University of Technology
Master of Science Thesis, 57 pages, 3 Appendix pages
December 2018
Master's Degree Programme in Information Technology
Major: Audio-Visual Signal Processing
Examiner: Professor Karen Eguiazarian (Egiazarian)

Keywords: Single image super-resolution, convolutional neural networks, image enhancement

Enlargement of images is a common need in many applications. Although increasing the pixel count of an image is easy with simple interpolation methods, those fail to increase the amount of details in the image. Single image super-resolution (SISR) aims to solve this ill-posed problem of producing a high resolution (HR) image from a given low resolution (LR) image. A single LR image has always an infinite number of corresponding LR images, but some of those are more probable than others. This probability density can be estimated with machine learning techniques, and the most probable HR image can be constructed based on that estimate.

In recent years artificial neural networks have become the most popular machine learning methods. Convolutional neural networks (CNN) are a subtype of them, inspired by the human visual system. They are used extensively in all fields of image processing, including single image super-resolution. In this thesis different CNN based methods for SISR are compared, and their performance is analyzed using both quantitative and qualitative methods. In total four CNN methods were chosen, and they were compared to three other methods. One of the reference methods was based on more traditional machine learning, and the two others were based on self-similarity of the input images. In contrast to machine learning approach, self-similarity based methods utilize only information in the input image and do not require any training on external images.

The results show that CNN based methods outperform the alternative approaches in both quantitative metrics and qualitative analysis. The methods perform especially well with images that have clear structures and sharp edges, but highly textured images tend to be problematic. Six of the methods aim to minimize pixel-wise reconstruction error, which leads to overly smooth output on textured areas. One method was instead designed to maximize the perceptual quality of the images, at the cost of increased reconstruction error. It was able to generate very realistic textures in some cases, but had a tendency to hallucinate very implausible textures into flat areas. Also other CNN based methods tended to create erroneous but plausible details, which might be misleading in critical applications like medical imaging. CNN based SISR is more suitable for entertainment and other consumer applications, especially when the perceptually optimized methods are developed further.

TIIVISTELMÄ

ANSSE SAARIMÄKI: Yhden kuvan superresoluutio konvoluutioneuroverkkoja käyttäen
Tampereen teknillinen yliopisto
Diplomityö, 57 sivua, 3 liitesivua
Joulukuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Audio-Visual Signal Processing
Tarkastaja: professori Karen Eguiazarian (Egiazarian)

Avainsanat: Yhden kuvan superresoluutio, konvoluutioneuroverkot, kuvanparannus

Digitaalisten kuvien suurentaminen on tarpeellista monissa sovellutuksissa, ja se on helppoa suorittaa yksinkertaisilla interpolaatimenetelmillä. Ne eivät kuitenkaan kykene lisäämään kuvan yksityiskohtia, ja varsinainen resoluutio jää samaksi kasvaneesta pikselimäärästä huolimatta. Korkeamman resoluution kuvan tuottaminen yhdestä matalan resoluution kuvasta on inversio-ongelma, jonka yhden kuvan super-resoluutio pyrkii ratkaisemaan. Yhdellä matalan resoluution kuvalla on aina ääretön määrä korkean resoluution vastineita, mutta osa niistä on aina todennäköisempiä kuin toiset. Tätä todennäköisyysjakaumaa voi estimoida koneoppimismenetelmien avulla, ja todennäköisin korkean resoluution kuva voidaan muodostaa tämän estimaatin pohjalta.

Viime vuosina keinotekoiset neuroverkot ovat muodostuneet suosituimmaksi lähestymistavaksi koneoppimiseen. Konvoluutioneuroverkot kuuluvat tähän ryhmän, ja ne ovat saaneet inspiraationsa ihmisen näköhermostosta. Niiden käyttö on erittäin yleistä kaikilla kuvankäsittelyn aloilla, mukaan lukien yhden kuvan superresoluutiossa. Tässä diplomityössä vertaillaan erilaisia konvoluutioverkkoihin perustuvia superresoluutiomenetelmiä, ja niiden suorituskykyä analysoidaan sekä kvantitatiivisesti että kvalitatiivisesti. Neljää valittua konvoluutioverkkopohjaista menetelmää verrataan kolmeen muuhun, joista yksi perustuu perinteisempään koneoppimiseen. Kaksi muuta menetelmää hyödyntävät tyypillisissä kuvissa toistuvia samankaltaisia elementtejä, eivätkä ne tarvitse opettamista ulkoisella kuvadatalla kuten koneoppimismenetelmät.

Tulokset osoittavat, että konvoluutioverkkopohjaiset menetelmät suoriutuvat vaihtoehtoisia menetelmiä paremmin sekä kvantitaviisessa että kvalitatiivisessa analyysissa. Menetelmät suoriutuvat erityisen hyvin kuvista joissa on selkeitä rakenteita ja teräviä rajoja, mutta voimaakkaasti teksturoidut kuvat tuottavat ongelmia. Menetelmistä kaikki paitsi yksi pyrkivät minimoimaan pikselikohtaisesti lasketun rekonstruointivirheen, joka johtaa liian tasaisiin pintoihin tekstuuripitoisilla alueilla. Yksi menetelmistä pyrkii maksimoimaan kuvien havainnoidun laadun rekonstruointivirheen kustannuksella, ja se kykeneekin tuottamaan realistisen näköisiä tekstuureja sen ansiosta. Useimmissa tapauksissa sillä oli kuitenkin taipumuksena tuottaa tekstuureja alueille joissa niitä ei kuuluisi olla, ja lopputulos oli erittäin epäuskottava. Myös muilla konvoluutioverkkoja käyttävillä menetelmillä oli taipumusta generoida virheellisiä, mutta uskottavan näköisiä yksityiskohtia. Tämä rajoittaa niiden soveltuvuutta kriittisiin käyttökohteisiin, kuten lääketieteelliseen kuvantamiseen. Konvoluutioverkkopohjainen yhden kuvan superresoluutio soveltuukin paremmin viihdesovelluksiin ja muuhun kuluttajakäyttöön, erityisesti mikäli havainnoidulle laadulle optimoidut menetelmät kehittyvät pidemmälle.

PREFACE

This thesis was started while I was working as a research assistant in the Computational Imaging Group of the Signal Processing Laboratory at Tampere University of Technology. Most of the background research for this thesis was done during that employment, but the thesis is otherwise independent from the project I was working on.

First of all I want to thank my supervisor and examiner, professor Karen Eguiazarian, for his guidance during all phases of this project, and for the opportunity work in his research group. Also I want thank my former colleague Cristóvão Cruz, who helped me numerous times, and created the MATLAB testbench used for testing the methods in this thesis.

Last but not least, I want to thank my wife Nyyti, for her support and patience during this prolonged project.

In Tampere, Finland, on 20 November 2018

Ansse Saarimäki

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CPU	central processing unit
GPU	graphics processing unit
HR	high resolution
LR	low resolution
MISR	multi-image super-resolution
ML	machine learning
MSE	mean square error
NN	nearest neighbor
PSNR	peak-signal-to-noise ratio
SGD	stochastic gradient descent
SISR	single image super-resolution
SR	super-resolution
SSIM	structural similarity index measure

1. INTRODUCTION

Digital images have become ubiquitous in our everyday lives, ranging from consumer applications like computer generated graphics on a website and holiday photos taken with a cellphone camera, to critical professional applications like medical imaging and surveillance camera footage. As images have become more common, their quality has increased. There are multiple aspects of image quality, but one of the most important ones is the resolution of an image. Resolution has multiple definitions in the context of imaging, and even more outside it, but we will be discussing only spatial resolution of the images. It can be defined as the number discrete points in an image that can be distinguished from each other [48, p. 18].

The resolution of digital images is usually expressed as number of pixels in the image, although it might not correspond with the actual resolution as defined above. Number of pixels sets the upper limit for resolution, but it might be lower depending on the image formation process. This is a common issue with typical digital cameras, where the number of pixels is dictated by the sensor, but resolution is lower due to demosaicking, low quality optics, focusing issues, denoising etc.

Similar problem occurs when we want to increase the size of a digital image, for example to view it on a higher resolution display. Number of pixels in the image must correspond to the display resolution and the desired image size. It can be achieved by interpolating the image, but the actual resolution is not increased while the number pixels increases. This leads to blurry looking images, as the amount of details in the image does not increase. This limit is present in all digital imaging systems, and it cannot be surpassed by any interpolation method.

Super-resolution (SR) tries to overcome this limit and produce a higher resolution image, which actually contains details not visible in the source image. There are two distinct ways to achieve this. The first, and more traditional one is the multi-image super-resolution (MISR) approach, which produces a high resolution (HR) image from multiple low resolution (LR) images depicting the same scene. Each of the input images should contain unique information for MISR to work, meaning that the images should have sub-pixel level displacements between them. This way the pixels are sampled from unique locations, and the effective total resolution is higher than the resolution of any single input image.

MISR has a very limited applicability, as often only a single LR image is available. This has lead to development of single image super-resolution (SISR) methods. There is always an infinite number of unique HR images that could correspond to a single LR image, and SISR methods can only make an assumption of what the HR image could look like. MISR

methods on the contrary can extract actual HR details hidden in the LR images.

SISR can be performed by utilizing a large image database with both LR images and their corresponding HR images. Machine learning techniques can be then used to learn the most probable, input content dependable mapping from LR to HR image [54, 6, 30, 36]. Another way is to utilize the self-similarity of the input image [9, 13, 26, 4]. Especially natural images tend to have elements that repeat throughout the image, possibly with different scale, rotation, and other types of affine transformations. When processing the image patch-by-patch, the similar patches can be treated like individual images and processed with MISR methods.

Convolutional neural networks (CNN) are the most popular machine learning approach used in state-of-the-art SISR research [53, 56]. They are a specific subtype of artificial neural networks, that are machine learning methods originally inspired by the workings of biological brains [19, p. 13]. CNNs are a relatively old concept [34], but their popularity surged to its current state only after 2012, when Krizhevsky et al. [33] published their AlexNet image classification network. It outperformed all of the previous methods by a clear margin, and changed the field of machine learning research completely [19, p. 24].

CNNs were first applied to SISR in 2014 by Dong et al. [6], and since it has become the most prominent approach in state-of-the-art SISR research [3, 53, 56]. In this thesis we will give an overview of CNN based super-resolution in a form of a literature review, and discuss both the state-of-the-art methods and their historical background. Additionally, a comparison of selected methods is performed, with both quantitative and qualitative analysis of their performance. A total of seven methods were chosen to represent distinct SISR approaches, with four of them being based on CNNs, two on self-similarity, and one on classical machine learning. The results show that CNN based methods have unparalleled performance in terms of image quality, and SISR research has made huge leaps through the usage of CNNs.

For understanding the CNN based SISR methods, their theoretical foundations are discussed in Chapter 2. To explain the problem SR is trying to solve, general theory on digital sampling and interpolation is outlined first. After that we discuss the image quality metrics used for quantitative comparison of the SISR methods. Last part of that chapter will explain the theory behind CNNs, starting with basics of machine learning and ending with CNN specific details.

Chapter 3 discusses the different SR methods, including MISR, self-similarity SISR, and machine learning based SISR. The main focus will be on CNN based methods, and they are explained in more depth than alternative approaches. The seven chosen methods are also analyzed in that chapter. Chapter 4 describes the testing setup used in the comparison, and the results are shown and analyzed in Chapter 5. Findings are summarized in Chapter 6, which also discusses the potential applications and topics for further research.

2. THEORETICAL BACKGROUND

As a basis for the super-resolution problem in general, a brief analysis of sampling and interpolation theory is given in Section 2.1. Section 2.2 will give an overview of image quality metrics used for performance comparison of SR algorithms in this thesis. Section 2.3 explains the main concepts of general machine learning and serves as a primer for the next section. Section 2.4 explains the class of artificial neural networks known as feed-forward networks, of which convolutional networks are a part of. That section will also explain the basics of convolutional networks, but the next chapter demonstrates how to apply them to super-resolution.

2.1 Sampling and interpolation

To really understand the problem super-resolution is trying to solve, one has to understand the limitations imposed by the discrete sampling process and why pure interpolation cannot overcome them. Digital imaging is essentially about sampling a two-dimensional continuous signal, e.g. the scene "seen" by the optics of a camera. The same principles apply to two-dimensional sampling as to the one-dimensional cases like digital audio. Naturally the human visual system works very differently in comparison to the auditory system, and thus many of the methods applied for audio processing are not usable in imaging context even when they work theoretically. Basics are nevertheless the same, and thus we will be mostly discussing the one-dimensional case in this section.

Let us consider a continuous band-limited real-valued function $f(t), t \in \mathbb{R}$. A uniformly sampled version of it is

$$\tilde{f}(t) = f(t)s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T), n \in \mathbb{N}, \tag{2.1}$$

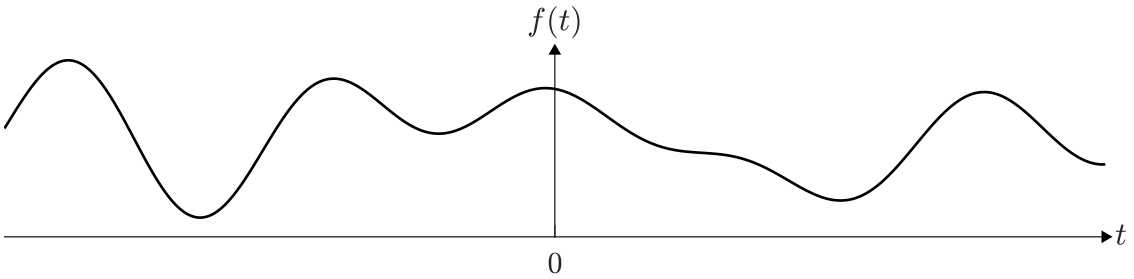
where ΔT is the sampling interval, $\delta(x)$ is the unit discrete impulse

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}, \tag{2.2}$$

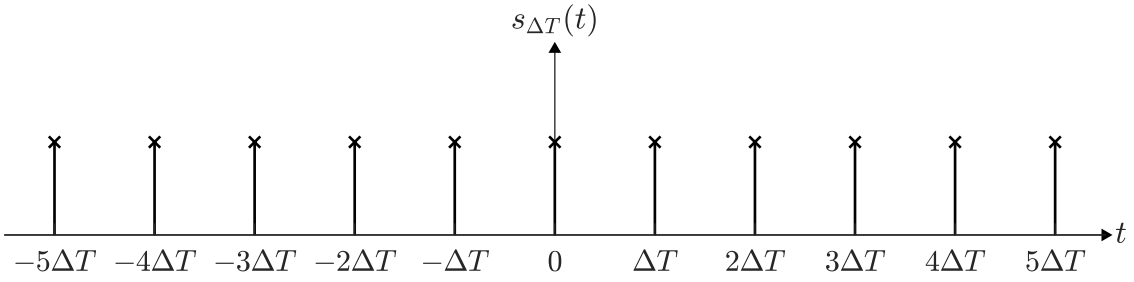
and

$$s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta T) \tag{2.3}$$

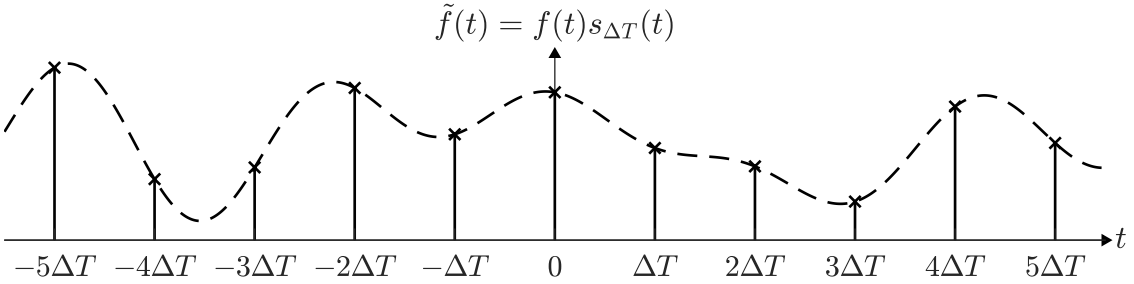
is the sampling function equal to a train of unit impulses with ΔT intervals. An example of function $f(t)$, its sampled version $\tilde{f}(t)$, and the sampling function $s_{\Delta T}(t)$ are illustrated



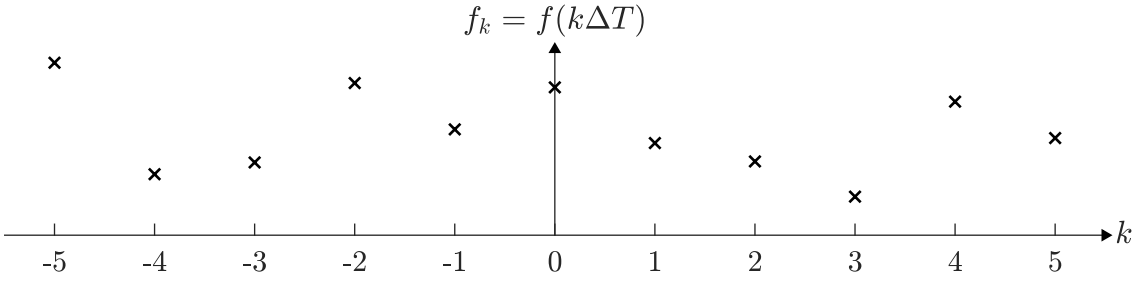
(a) A continuous band-limited signal $f(t)$.



(b) The sampling function $s_{\Delta T}(t)$.



(c) The sampled signal $\tilde{f}(t)$. The dashed line is included as a reference only and it is not part of the signal.



(d) The discrete sample sequence f_k.

Figure 2.1. *Illustration of a continuous signal and its sampling process.*

in Figure 2.1. We can also denote the sampled function as a sequence of discrete values $f_k = f(t_k), k \in \mathbb{N}$, in which $t_k = k\Delta T$ is the sample location. [18, p. 212]

The Fourier transform of function $f(t)$ is

$$\mathfrak{F}\{f(t)\} = F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi\omega t}\,\mathrm{d}t, \tag{2.4}$$

where ω is the frequency. Since the function $f(t)$ is band-limited, there exists a frequency $\omega_{\max}$, for which $F(\omega_{\max}) \neq 0$ and $F(\omega) = 0, \forall\omega > \omega_{\max}$. This the highest frequency

component of the function $f(t)$. The Fourier transform of the sampling function $s_{\Delta T}(t)$ is

$$\mathfrak{F}\{s_{\Delta T}(t)\} = S_{\Delta T}(\omega) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \delta\left(\omega - \frac{n}{\Delta T}\right), \tag{2.5}$$

which is also an impulse train, but with an interval of $\frac{1}{\Delta T}$. The Fourier transform of the sampled function $\tilde{f}(t)$ is thus $\tilde{F}(\omega) = \mathfrak{F}\{\tilde{f}(t)\} = \mathfrak{F}\{f(t)s_{\Delta T}(t)\}$. As a Fourier transform of a product of two functions is the convolution of Fourier transforms of those functions, the sampled function in Fourier-domain is

$$\begin{aligned} \tilde{F}(\omega) &= \mathfrak{F}\{f(t)s_{\Delta T}(t)\} \\ &= F(\omega) * S_{\Delta T}(\omega) \\ &= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(\omega - \frac{n}{\Delta T}\right), \end{aligned} \tag{2.6}$$

where the $*$ is the convolution operation. [18, p. 212–213]

The Nyquist-Shannon sampling theorem states that $f(t)$ can be perfectly reconstructed from $\tilde{f}(t)$ when the sampling rate $\frac{1}{\Delta T}$ is greater than twice the highest frequency component $\omega_{\max}$ of the function $f(t)$. The frequency limit of $\frac{1}{2\Delta T}$ is called the Nyquist rate (or the Nyquist frequency), and it is the highest frequency that can be recovered from the sampled signal. The case of $\omega_{\max} < \frac{1}{2\Delta T}$ is called oversampling, $\omega_{\max} = \frac{1}{2\Delta T}$ is critical sampling and $\omega_{\max} > \frac{1}{2\Delta T}$ is undersampling. In the case of oversampling, the original function $f(t)$ can be perfectly recovered from the sampled version $\tilde{f}(t)$ by filtering out all the frequencies greater than $\omega_{\max}$ with ideal low-pass filter. In the case of undersampling, the frequencies of $F(\omega)$ higher than $\frac{1}{2\Delta T}$ will be overlapping with the frequencies below. This effect is called aliasing, and it prevents a perfect reconstruction and causes noticeable artifacts in the sampled signal. [18, p. 214–215]

The perfect reconstruction using an ideal low-pass filter can be expressed mathematically in the Frequency domain as

$$F(\omega) = H(\omega)\tilde{F}(\omega), \tag{2.7}$$

where

$$H(\omega) = \begin{cases} \Delta T & -\omega_c \leq \omega \leq \omega_c \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

is the filter's frequency response and $\omega_c = \frac{1}{2\Delta T} > \omega_{\max}$ is the filter's cutoff frequency. The original signal $f(t)$ can then be acquired from $F(\omega)$ using inverse Fourier transform:

$$f(t) = \mathfrak{F}^{-1}\{F(\omega)\} = \int_{-\infty}^{\infty} F(\omega)e^{j2\pi\omega t}\, \mathrm{d}\omega \tag{2.9}$$

This filtering can also presented in spatial domain using convolution as

$$f(t) = h(t) * \tilde{f}(t), \tag{2.10}$$

where

$$h(t) = \mathfrak{F}^{-1}\{H(\omega)\} = 2\omega_c \Delta T \operatorname{sinc}(2\omega_c \Delta T t) = \operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \tag{2.11}$$

is the filter's impulse response. This ideal filter will attenuate all of the frequencies over ω_c, leaving only the original signal $f(t)$. [18, p. 215–216, 220]

In practice this perfect sampling can never be achieved. First problem arises with the notion of a continuous band-limited signal, as no signal of finite length can be truly band-limited [18, p. 218]. All practical images are finite in length, thus there will always be some aliasing introduced in the sampling process. The aliasing cannot be completely removed, although it can be attenuated greatly with proper pre-filtering. Second problem is the ideal low-pass filter required for the perfect reconstruction, as its impulse response, the sinc-function, has infinite length.

Although the perfect reconstruction of $f(t)$ is practically impossible, an adequate approximation $\hat{f}(t) \sim f(t)$ could be achieved through convolution $\hat{f}(t) = \hat{h}(t) * \tilde{f}(t)$ with some other interpolation filter $\hat{h}(t)$. For $\hat{h}(t)$ to be an interpolation filter, it has to satisfy the following requirements:

$$\hat{h}(t) = \begin{cases} 1 & t = 0 \\ 0 & t = n\Delta T, |n| = 1, 2, \ldots \end{cases} \tag{2.12}$$

This guarantees that the function values at sample locations will remain unchanged, thus $\hat{f}(n\Delta T) = \tilde{f}(n\Delta T) = f(n\Delta T), \forall n \in \mathbb{N}$. For simplicity we will assume from now on that the sampling interval $\Delta T = 1$, so that that the samples are always located at $t_k = k, k \in \mathbb{N}$. In the context of this thesis the actual sampling interval is irrelevant and it is also unknown for all the images used. Thus the interpolation can be represented in discrete terms as

$$\hat{f}(t) = \sum_{k=-\infty}^{\infty} f_k \hat{h}(t - k), \tag{2.13}$$

where $f_k = f(k)$ is the sequence of sampled values. [29]

The simplest possible way to interpolate is to use the so-called nearest neighbor (NN) method, which chooses the value of the nearest sample as the interpolated value [18, p. 65]. In terms of filter impulse response, the method is defined as

$$h_{\text{nn}}(t) = \begin{cases} 1 & -0.5 < t \leq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

Although it is mathematically very simple and computationally efficient, NN often produces visually unpleasing results. Images upsampled with NN interpolation typically contain blocky artifacts, which can be seen in Figure 2.4(a). Although those artifacts are intuitively explained with the filter's impulse response, their source is also easily visible in the filter's frequency response

$$H_{\text{nn}}(\omega) = \mathfrak{F}\{h_{\text{nn}}(t)\} = \operatorname{sinc}(\omega) = \frac{\sin(\pi\omega)}{\pi\omega}. \tag{2.15}$$

The amplitude response $|H_{\text{nn}}(\omega)|$ is illustrated in Figure 2.3, which clearly shows the poor performance when compared to the ideal reconstruction filter $|H_{\text{sinc}}(\omega)|$. Frequencies above the Nyquist-rate are attenuated inadequately and the resulting reconstruction is far from ideal.

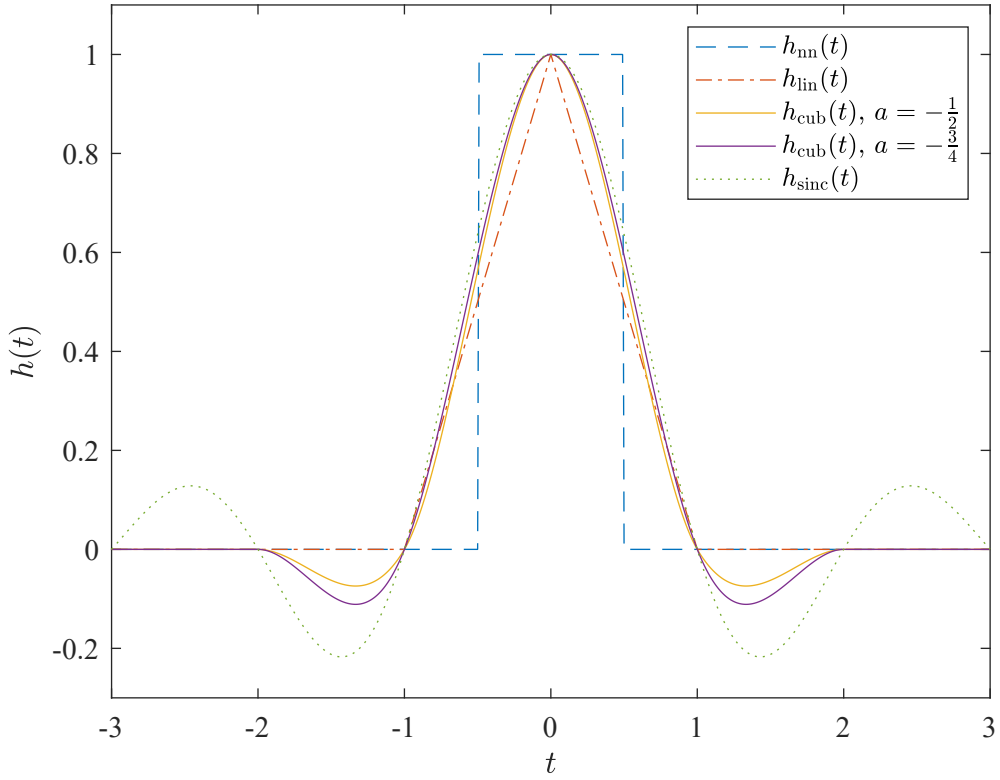


Figure 2.2. *Impulse responses of NN, linear, cubic and sinc interpolation filters.*

A simple improvement over NN interpolation is to use the two nearest neighbors, and linearly interpolate between them [18, p. 65]. In two-dimensional cases the method is commonly called bilinear interpolation, and four nearest neighbors are used. The impulse response of the one-dimensional filter is defined as

$$h_{\text{lin}}(t) = \begin{cases} 1 - |t| & |t| < 1 \\ 0 & \text{otherwise,} \end{cases} \tag{2.16}$$

and its frequency response is

$$H_{\text{lin}}(\omega) = \mathfrak{F}\{h_{\text{lin}}(t)\} = \text{sinc}^2(t) = \frac{\sin^2(\pi\omega)}{\pi^2\omega^2}. \tag{2.17}$$

With this filter the resulting image is notably smoother, but there is still visible jaggedness around the sharpest edges, as can be seen in Figure 2.4(b). Also the amplitude response (Fig. 2.3) has still room for improvement.

A commonly used compromise between computational complexity and output image quality is the cubic convolution interpolation introduced by Keys in 1981 [29]. We are

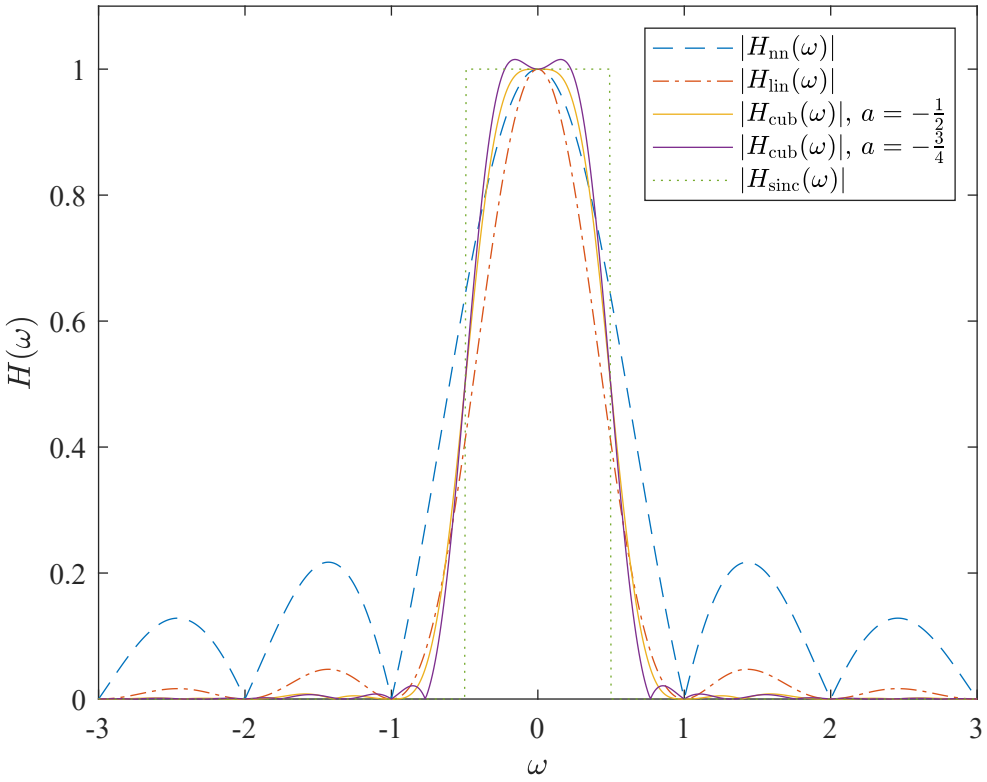


Figure 2.3. *Amplitude responses of NN, linear, cubic and sinc interpolation filters.*

referring its two-dimensional case as bicubic interpolation, although usage of that word is ambiguous in other literature. The one-dimensional cubic method interpolates based on the four nearest samples using a kernel defined as

$$h_{\mathrm{cub}}(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & 0 < |t| \leq 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & 1 < |t| \leq 2 \\ 0 & \text{otherwise,} \end{cases} \tag{2.18}$$

where a is an constant affecting the properties of the kernel. Keys proved that by setting the parameter as $a = -\frac{1}{2}$, the interpolated signal $\hat{f}(t)$ will converge fastest towards the original signal $f(t)$, when the sampling interval approaches zero [29]. Using this value the kernel reduces to

$$h_{\mathrm{cub}}(t) = \begin{cases} \frac{3}{2}|t|^3 - \frac{5}{2}|t|^2 + 1 & 0 < |t| \leq 1 \\ -\frac{1}{2}|t|^3 + \frac{5}{2}|t|^2 - 4|t| + 2 & 1 < |t| \leq 2 \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

and its frequency response becomes

$$H_{\mathrm{cub}}(\omega) = \mathfrak{F}\{h_{\mathrm{cub}}(t)\} = \frac{\sin^3(\pi\omega)(-2\pi\omega\cos(\pi\omega) + 3\sin(\pi\omega))}{\pi^4\omega^4}. \tag{2.20}$$

This form is used e.g. in MATLAB's `imresize` function [39], which is the reference resampling method in this thesis. Other values of a are also possible, and e.g. OpenCV uses $a = -\frac{3}{4}$ instead [42]. Impulse and amplitude responses for both values of a are demonstrated in Figures 2.2 and 2.3 respectively. For both options the impulse response is a quite close approximation of the sinc filter for $|t| < 1$, and they include a negative lobe in $1 < |t| < 2$ like the sinc filter. The amplitude shows a lot higher attenuation for the stop-band frequencies than the previous two methods and the attenuation is less pronounced for the higher pass-band frequencies.

Resulting images with bicubic upsampling are presented in Figures 2.4(c) and 2.4(d) with $a = -\frac{1}{2}$ and $a = -\frac{3}{4}$ respectively. The results for both are very similar, with sharper edges and less jaggedness in comparison to bilinear upsampling. Due to the negative lobes of the filter kernels, there is a notable ringing visible in high contrast edges such as in the back of the man. It is more pronounced in the image produced with $a = -\frac{3}{4}$. Although this ringing is mostly an unwanted effect it does increase the perceived sharpness and thus helps in some cases.

An approximation of the ideal reconstruction with a sinc filter has been included for comparison in Figure 2.4(e). As was said earlier, the truly ideal reconstruction cannot be achieved with finite length signal, which has been circumvented in this example by padding the input image with infinite zeros. Effectively this truncates the filter kernel to the size of the input image and makes the computation tractable. This zero-padding will lead to large amounts of ringing around the image borders and makes the image unusable, but it does highlight the fundamental problem of this ideal reconstruction method. Every sharp brightness transition in the input image will lead to similar ringing artifacts, which will be repeated around the edges until attenuated below the quantization noise level. Thus the ideal reconstruction would be a poor choice for image interpolation, even if it were attainable in practice.

The upsampled images in Figures 2.4(a)–2.4(e) have all been produced by first downsampling the original image (Fig. 2.4(f)) by a factor of 4 and then upsampled using the corresponding interpolation methods and scaling factor of 4. The downsampling method is using the bicubic filter and it is described in more detail below. This is the scheme that will be utilized in this thesis also for comparing the super-resolution methods. Included in the figures are PSNR (peak-signal-to-noise ratio) scores for the upsampled images, for which higher value means higher similarity with the ground truth image (the original high resolution image). The PSNR score and other image quality metrics will be discussed in the Section 2.2.

To resize the images in general, we can just utilize the equation 2.13 and set the new sample locations as $t = \frac{n\Delta T}{s}$, where s is the image scaling factor. This works well for upsampling($s > 1$), but with downsampling ($s < 1$) there is a huge risk of aliasing. This can be alleviated by using an anti-aliasing (AA) filter, which attenuates the frequencies above the new Nyquist-rate from the interpolated image. AA-filter is essentially a low-pass filter just like the interpolation filter, but with a lower cut-off frequency. Thus we can

(a) Nearest neighbour (PSNR: 21.65 dB)

(b) Bilinear (PSNR: 22.25 dB)

(c) Bicubic, $a = -\frac{1}{2}$ (PSNR: 22.68 dB)

(d) Bicubic, $a = -\frac{3}{4}$ (PSNR: 22.78 dB)

(e) Sinc (PSNR: 22.08 dB)

(f) Ground truth

Figure 2.4. *Example images produced by upsampling with different interpolation methods. The ground truth image (f) has been first downsampled with scaling factor of* $1/4$ *to produce a low resolution input. Images a–e have been upsampled from that LR image with factor of 4 using the corresponding interpolation method.*

achieve the goal with a single filter, by lowering the cut-off frequency of the interpolation filter. The cut-off frequency is inversely proportional to the width of the filter kernel [60, p. 374]. Thus with an image scaling factor of $s < 1$ and an interpolation filter $\hat{h}(t)$, the desired anti-aliasing filter $h_{\text{aa}}(t)$ will become

$$h_{\text{aa}}(t) = s\hat{h}(st). \tag{2.21}$$

The resulting interpolation kernel will have a support $\frac{1}{s}$ wider than the original un-dilated kernel. This way of producing a downsampling kernel is appropriate for all interpolation methods other than the nearest-neighbour, which should select the single nearest neighbour also for all the pixels of a downsampled image. This scheme is used also in MATLAB [38], and it will also be used when producing the low-resolution input images for testing the SR methods.

2.2 Image quality metrics

Image enhancement methods like super-resolution are typically used to produce images for humans to view, and thus maximizing the perceived quality of the output images is important. The only way to get a reliable estimate of the perceived quality, is to test with a large group of people, and a large dataset of different images. The images should have different types and levels of degradations, and they should be processed with multiple competing methods. The relative performance of the methods would then be compared using the mean opinion score (MOS) collected from those images [45]. This is obviously a very laborious task, and practically impossible to conceive during the development phase of a method. Thus there is a need for some quantitative image quality metric that can be easily calculated for any image. Numerous image quality metrics have been developed for this task, and they can be coarsely divided into two subgroups: signal fidelity metrics and perceptual visual quality metrics [37].

Signal fidelity metrics are the traditional methods like mean absolute error, mean square error (MSE), signal-to-noise ratio (SNR), peak-signal-to-noise ratio (PSNR), and their close relatives [37]. These are simple to calculate and well justified by the underlying physics, but they are not designed specifically for measuring image quality. They are widely known to correlate poorly with the perceived quality, but they are nevertheless universally used in different image processing tasks [12, 37, 45].

PSNR is the de-facto metric used in modern super-resolution research [6, 11, 26, 30, 35, 36, 41, 53, 55, 56], thus it chosen also for this thesis. PSNR is based on MSE, which is defined as

$$\text{MSE}(\mathbf{G}, \mathbf{D}) = \frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \left(G_{i,j} - D_{i,j}\right)^2, \tag{2.22}$$

where **G** is the reference (i.e. ground truth) image and **D** is the degraded image [24]. Both are grayscale images represented by matrices of size $n \times m$. PSNR itself is defined as

$$\mathrm{PSNR}(\mathbf{G},\mathbf{D}) = 10\log_{10}\left(\frac{\mathrm{Max}_G^2}{\mathrm{MSE}(\mathbf{G},\mathbf{D})}\right), \tag{2.23}$$

where Max_G is the highest possible brightness value for **G** (and **D**) [1]. For 8 bit grayscale images the value is $\mathrm{Max}_G = 255$.

PSNR is considered to be a poor estimator of perceived image quality, which motivates the use of perceptual visual quality metrics. Structural similarity index measure (SSIM) by Wang et al.[62] is one such metric and a very common choice to accompany PSNR in super-resolution research [11, 26, 30, 35, 36, 41, 53, 56]. SSIM is calculated locally for each pixel of the images **G**, and **D** with the equation

$$\mathrm{SSIM}(\mathbf{g},\mathbf{d}) = l(\mathbf{g},\mathbf{d})c(\mathbf{g},\mathbf{d})s(\mathbf{g},\mathbf{d}), \tag{2.24}$$

where vectors $\mathbf{g} = \{g_i | i = 0,1,\cdots,k-1\}$ and $\mathbf{d} = \{d_i | i = 0,1,\cdots,k-1\}$ represent the k-pixel local neighborhoods of corresponding pixels from images **G** and **D**. The term $l(\mathbf{g},\mathbf{d})$ corresponds to luminance, $c(\mathbf{g},\mathbf{d})$ to contrast and $s(\mathbf{g},\mathbf{d})$ to structural similarity of the pixel neighborhoods, and they are defined as

$$l(\mathbf{g},\mathbf{d}) = \frac{2\mu_g\mu_d + c_1}{\mu_g^2 + \mu_d^2 + c_1} \tag{2.25}$$

$$c(\mathbf{g},\mathbf{d}) = \frac{2\sigma_g\sigma_d + c_2}{\sigma_g^2 + \sigma_d^2 + c_2} \tag{2.26}$$

$$s(\mathbf{g},\mathbf{d}) = \frac{\sigma_{gd} + c_3}{\sigma_g\sigma_d + c_3}. \tag{2.27}$$

The local statistics μ_g, μ_d, σ_g, σ_d and σ_{gd} are estimated from 11×11 pixel neighborhood weighted with a circular-symmetric Gaussian window $\mathbf{w} = \{w_i | i = 0,1,\cdots,k-1\}$, which has standard deviation of 1.5 samples and has been normalized to unit sum. The estimated statistic are then defined as

$$\mu_g = \sum_{i=0}^{k-1} w_i g_i \tag{2.28}$$

$$\sigma_g = \left(\sum_{i=0}^{k-1} w_i (g_i - \mu_g)^2\right)^{\frac{1}{2}} \tag{2.29}$$

$$\sigma_{gd} = \sum_{i=0}^{k-1} w_i (g_i - \mu_g)(d_i - \mu_d). \tag{2.30}$$

Constants c_1, c_2 and c_3 are included to avoid null denominators and to stabilize the metric. The authors chose to use values $c_1 = (0.01\mathrm{Max}_G)^2$, $c_2 = (0.03\mathrm{Max}_G)^2$ and $c_3 = \frac{c_2}{2}$, which are also used in this thesis.

The equation 2.24 defines the SSIM score locally for a specific part of an image, but we are interested in the quality of the whole image. For evaluation of full images we use the mean SSIM

$$\mathrm{MSSIM}(\mathbf{G},\mathbf{D}) = \frac{1}{nm}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}\mathrm{SSIM}(\mathbf{g}_{i,j},\mathbf{d}_{i,j}), \tag{2.31}$$

where vectors $\mathbf{g}_{i,j}$ and $\mathbf{d}_{i,j}$ are the neighborhoods of pixels $G_{i,j}$ and $D_{i,j}$ from $n \times m$ images $\mathbf{G}$ and $\mathbf{D}$. In the rest of this thesis SSIM will always refer to the MSSIM from equation 2.31, as the local SSIM by itself is useless in this context.

Both PSNR and SSIM were described above only for grayscale images, as the scores are typically calculated only on the luminance (Y) channel of YCbCr images. If quality score on all color channels is needed, both metrics can be easily extended to work for RGB images, by changing the matrices $\mathbf{G}$ and $\mathbf{D}$ to 3-dimensional tensors $\boldsymbol{\mathsf{G}}$ and $\boldsymbol{\mathsf{D}}$ of size $n \times m \times 3$ and extending the summations in equations 2.22 and 2.31 to work along the third dimension also.

Both PSNR and SSIM belong to a group of image quality metrics called full-reference metrics, as the score is calculated based on the image's similarity with a reference image. There are also metrics that estimate the quality purely from the degraded image without any reference information, and they are respectively called no-reference metrics. Also metrics utilizing only a part of the reference image exist, and they are called reduced-reference metrics. [37]

Only PSNR and SSIM are used in this thesis, as they are the metrics conventionally used in the field of super-resolution research. Nevertheless, it would be justified to include other, more advanced metrics, as even SSIM has been proven to be a poor estimate for perceived quality. Ponomarenko et al. [45] have shown that PSNR and SSIM correlate as poorly with the mean opinion score, and Horé et al. [24] have shown that PSNR and SSIM are very similar and mostly differ by their sensitivities to specific image degradations.

2.3 Machine learning basics

Main focus of this thesis will be on convolutional neural networks and their application on super-resolution, which covers only a small part of the whole field of machine learning research and its applications. The same principles still apply to SR and CNNs as do to other fields, and definition of those principles is in order. This section will describe only the basic concepts in a general level, and more detailed explanations and concrete examples will be given in the next section in the context of neural networks.

Machine learning in general refers to algorithms, or computer programs, that can learn to execute some task based on the data that is used to train those algorithms. But what actually is the definition of learning in this case? Mitchell defined it as follows: "A computer

program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [40, p. 2]

In our case the task T would be super-resolution, the experience E would be the dataset used for training the algorithm and the performance measure P would be the quality metrics described in Section 2.2. Learning itself is not the task, it is just the means for producing a computer program, often called model, that solves the task.

2.3.1 Types of ML tasks

The way an model solves the task T can be seen as a function that maps the input data to the desired form of output data, and the different groups of tasks can be described in terms of their input and output data types. We represent an example (a single instance) of input data with vector $\mathbf{x} \in \mathbb{R}^n$, where each entry x_i of the vector corresponds to a feature of the input data. In the case of SR the example would be a single image we want to super-resolve and its pixels would be the features. A vector is chosen as the example only to keep the definition simpler, but a matrix could be also used as it would make more sense for images.

Type of the task T is the most important factor when selecting a suitable machine learning method, and also the experience E and performance measure P are highly dependent on T. Thus it makes sense to classify machine learning algorithms based on the task they are trying to solve. The largest and most widely known groups of tasks are classification and regression [19, p. 98–99], which will be discussed in more detail below.

Classification is the largest group of ML tasks and it has been the driving force behind the modern machine learning advancements [19, p. 98]. In this task learning is used to produce an classifier that assigns input example to one of k categories (or classes). The classifier is usually a function $f : \mathbb{R}^n \rightarrow \{1, \cdots, k\}$. When $y = f(\mathbf{x})$, the model classifies an input example $\mathbf{x}$ to category identified by the integer y. The model could also output for each of the categories the probability of $\mathbf{x}$ belonging to that class. Typical example is image classification, which is also the most common usage for convolutional networks. E.g. the algorithm might be trained to classify pictures of cats, dogs and humans, and given an input image it will categorize it to one of these classes. [19, p. 98]

Regression task is defined by Goodfellow et al. [19, p. 99] as predicting a single numerical value given some numerical input, and it can represented by function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. With input $\mathbf{x}$ the output $y = f(\mathbf{x})$ will be a scalar value. A common application for regression is the prediction of price developments in stock market.

The above definitions by Goodfellow et al. [19] are relatively strict, in the sense that they allow only a single output for the model. In the image classification example mentioned above, the model would have to choose only a single category, even if the model were given a picture with both a human and a dog in it.

For tasks that require multi-value outputs with important relationships between those values (e.g. vectors), Goodfellow et al. [19, p. 99] introduce group called "structured output". Basically this includes all types of tasks that do not fit into above definitions of classification and regression. A group this broad is almost useless for categorizing the tasks and we will instead broaden the definition of the above mentioned groups to include multiple outputs. Thus we consider super-resolution to be a special case of regression, although it has multiple output values (one or three for each pixel of the output image), and their relative locations are of utmost importance.

There are of course multiple other types of ML tasks, but many of them can be considered as subgroups of the above mentioned regression and classification, at least when we extend the definitions to include multiple outputs. Some tasks still fit poorly into either of those, like machine translation, speech recognition and other tasks with natural language output.

2.3.2 Supervised and unsupervised learning

Usually the experience E is organized as a dataset, which is a collection of individual examples, and the algorithm is allowed to experience this whole dataset during training. The dataset contents provide another way to coarsely divide machine learning algorithms into two subgroups, based on the type of experience the algorithm is allowed to have during training. These groups are called supervised and unsupervised learning algorithms.

In supervised learning the dataset consists of both inputs examples $\mathbf{x}$ and associated output labels or targets $\mathbf{y}$. In the earlier classification example, the label $\mathbf{y}$ would tell in which category (cat, dog or human) the training example belongs. For regression tasks such as SR, we prefer calling the label $\mathbf{y}$ target instead, as it better describes the nature of that data. For SR the target $\mathbf{y}$ would be the so-called ground truth image, a high resolution image of which the input $\mathbf{x}$ is a downsampled version. The supervised case can be viewed as estimating the probability distribution $p(\mathbf{y}|\mathbf{x})$, and predicting the most probable output $\mathbf{y}$ given the input $\mathbf{x}$).

In unsupervised learning the dataset consists of only input examples $\mathbf{x}$, with no labels or targets, and the goal is to learn some useful properties about the structure of dataset. A good example of this is clustering, where the task is to divide the input data into clusters of similar examples. Clustering algorithm has to learn the division rules by itself, with no other guidance than the number of clusters needed. Texture synthesis is another prime example of this type of learning, and somewhat closely related to SR. Idea in synthesis is to implicitly learn the probability distribution $p(\mathbf{x})$ that produced the examples in the dataset, so that new examples can be synthesized from it. The goal could also be learning the probability distribution explicitly like in the case of density estimation. [19, p. 103]

There are of course cases that fall somewhere between the supervised and unsupervised, as the dataset could include labels or targets only for some the examples. This semi-supervised

case, and the unsupervised case, will be outside the scope this thesis, as we are interested only in supervised learning in the form of super-resolution.

2.3.3 Datasets and model performance

The performance measure P is not as useful for categorizing the different algorithms, but it is still an essential concept in machine learning basics. It was stated earlier that in our case P would be the image quality metrics of Section 2.2, but specifying only the metric is not enough. It is important to define also the data that the performance will be measured on. Thus far we have only mentioned the training dataset, and although the we want the model to perform well on the training data, that performance is not what we are ultimately after. Instead we want to maximize the model's performance on data it has never experienced before, i.e. we want the model to generalize well to any data that we want to use it for.

Maximizing the model's performance only on the training data is easy; we can just select a model with enough capacity to store every input example and the corresponding output of the dataset, and train the model to associate them to each other. This model would give the correct output every time it is given an example from the training dataset, but most likely it would not work at all for any other data. This is an extreme example of overfitting, a condition that impairs the models generalization capability.

The opposite of overfitting is underfitting, which happens when the model's performance is poor even on the training dataset. If the model's capacity is inadequate for the task, underfitting will definitely happen. Nevertheless, even a model with sufficient capacity might underfit if the training procedure fails due to some reason.

Overfitting and underfitting can also be defined in terms of two different performance measures: training error and generalization error. Training error is calculated on the training dataset and generalization error on a separate dataset consisting of examples the model has not seen before. This dataset is called the test set, and thus the error calculated on it is also known as test error. Underfitting occurs when the test error is too large, and overfitting when the training is error notably smaller than test error. Both conditions are to be avoided and a well performing model minimizes both the training error and the gap between training and test errors.

It is also possible for the test error to be lower than the training error, but usually this indicates that the examples in the test set are significantly easier to predict than the training examples. Testing on a set like this might not give a realistic estimate of the real generalization error, and the test set should be changed. Ideally the test set and training should be identically distributed, but still independent from each other.

Adjusting the model's capacity is the main way of controlling the model's tendency to overfit or underfit. The capacity is the model's ability to fit to a wide variety of functions, an abstract concept that cannot be exactly quantified or defined. The actual learning happens

by adjusting the model's trainable parameters according to the experience given, and the number of these parameters correlates strongly with the model's capacity. The parameter count itself is not the only element affecting the capacity though, as the operations these parameters control and their interactions define what functions the model can fit to.

Most machine learning algorithms have settings for adjusting their behavior, and these settings are called hyperparameters to distinguish them from the trainable parameters of the model. These parameters are chosen to be set by the user and not learned from the data, usually because they are either difficult to optimize or unsuitable for learning from the data. For example the model's capacity cannot be properly learned from the training data, as maximum capacity would always minimize the training error and thus be chosen by the algorithm.

For most of the hyperparameters there is no way to choose optimal values without trying different combinations then and choosing the best performing ones. But we cannot choose the values based on the test set performance, as the information from the test set would indirectly affect the learning process. Thus we need a third set of data, the validation dataset, that will be used for choosing the hyperparameters and testing performance during the training process. Usually it is a small part split from the training set. The validation error will typically underestimate the real generalization error, because the hyperparameters have been "learned" from it, although the validation data will not be used for the actual parameter updates. It will nevertheless be a better estimate than the training error alone.

2.4 Feedforward neural networks

Previous section discussed the basic concepts of machine learning, but did not give any details on how these concepts can be applied in practice, as it is highly dependent on the actual algorithm family used. As convolutional neural networks are a special case of feedforward neural networks, we will introduce them using fully connected networks, or multilayer perceptrons, as an example. They are the simplest form of neural networks and based on the perceptron concept originally introduced by Rosenblatt in 1958 [47], which is a linear binary classifier loosely inspired by the neurons of a human brain. As we are not interested in classification, in the following examples the perceptron will perform linear regression instead.

An example of a simple multilayer perceptron can found in Figure 2.5. A multilayer perceptron contains always an input layer, an output layer and one or more hidden layers, i.e. two or more perceptrons chained together. The number of layers is referred as the depth of the network, which is the origin of the term "deep learning". Input layer is just a collection of input values and thus not a perceptron like the other two layers. Our example has a vector valued input $\mathbf{x} = [x_1, x_2]^\top$, hidden layer $\mathbf{h} = [h_1, h_2, h_3]^\top$ and an output value y. In the graph the variables are denoted with nodes and their relations are indicated by the edges.

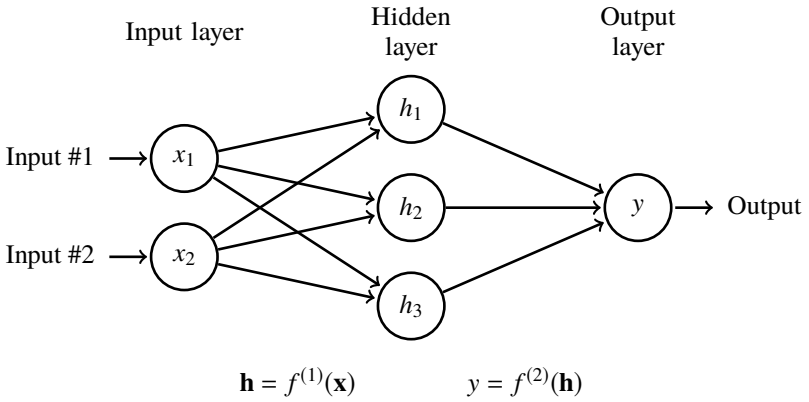


Figure 2.5. *A simple multilayer perceptron.*

This network can represented as a chain of functions $f(\mathbf{x}) = f^{(2)}(f^{(1)}(\mathbf{x}))$, where the hidden layer is represented by the equation $\mathbf{h} = f^{(1)}(\mathbf{x})$ and the output layer by $y = f^{(2)}(\mathbf{h})$. The functions $f^{(1)}$ and $f^{(2)}$ are defined as

$$f^{(1)}(\mathbf{x}) = \mathbf{W}^\top \mathbf{x} + \mathbf{b} \tag{2.32}$$

$$f^{(2)}(\mathbf{h}) = \mathbf{w}^\top \mathbf{h} + c, \tag{2.33}$$

where $\mathbf{W}$ is a 2×3 matrix containing the weights of the hidden layer, vector $\mathbf{b} = [b_1, b_2, b_3]^\top$ contains the biases for the hidden layer, vector $\mathbf{w} = [w_1, w_2, w_3]^\top$ contains the output layer weights and c is the output bias. Together these parameters $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{w}, \mathbf{b}, c\}$ are the trainable parameters of this network, and we will use the notation $f(\mathbf{x}; \boldsymbol{\theta})$ to indicate the models dependency on the parameters, when referring to the network.

2.4.1 Activation layers

Neural networks are known for their ability to approximate nonlinear functions, but the above example consists of only linear functions. No matter how many linear functions we chain to together, the model will stay linear. Thus we need to introduce some nonlinearity to the model, which is achieved using activation functions. The most common activation function in modern neural networks is the rectified linear unit (ReLU) [19, p. 171], which is defined as

$$g(z) = \max\{0, z\}. \tag{2.34}$$

The activation function is used in the hidden layers to modify the output of the nodes. In the above example we would utilize it after the function $f^{(1)}$ so that the model becomes $f(\mathbf{x}) = f^{(2)}(g(f^{(1)}(\mathbf{x})))$.

Earlier neural networks used the sigmoid, or logistic, activation function, which is defined as

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{2.35}$$

Sigmoid function's output range is $(0,1)$, and it saturates strongly negative values to zero and strongly positive values to one. This saturation leads to problem known as vanishing gradients with deeper networks. Vanishing gradients can be mitigated by using ReLU activation instead in the hidden layers, but sigmoid function is still useful as the output unit in classifier networks. For regression tasks like SR the best choice is using linear output unit.

2.4.2 Loss functions

Although the goal of training a neural network is maximizing some performance metric (in our case PSNR and SSIM), the same metric used in performance comparison is not typically used for steering the training process. In some cases it could be possible, at least with some minor modifications to the metric, but it still might not feasible. The metric used in the training process is called the loss function (or cost function), and the goal of the process is to minimize it on the training data. In this thesis the loss function refers to a loss calculated on a single example, and the cost function refers to loss calculated on the whole dataset or specific subset of it, although in some literature they are used interchangeably.

Given a loss function $\mathcal{L}(\hat{\mathbf{y}},\mathbf{y})$, where $\hat{\mathbf{y}} = f(\mathbf{x})$ is the network output and $\mathbf{y}$ is the training target corresponding to the input $\mathbf{x}$, the loss function output is a scalar with lower values corresponding to greater similarity between vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$. The only other requirement for the loss funtion is that it is differentiable over $\hat{\mathbf{y}}$, as the parameter updates done during the training are based on the gradient of the loss function.

The most common loss function used in image processing tasks like SR, is the ℓ_2 loss, which is defined as the square of ℓ_2 norm of the difference $\mathbf{y} - \hat{\mathbf{y}}$

$$\mathcal{L}_{\ell_2}(\hat{\mathbf{y}},\mathbf{y}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 . \tag{2.36}$$

This loss function is essentially the sum of squared errors, thus minimizing it minimizes also MSE and maximizes PSNR. This makes it an intuitive choice for maximizing PSNR, but it has been recently shown to be a suboptimal choice.

Zhao et al. [68] compared the performance of models trained using different loss functions in three different image restoration tasks: joint demosaicking and denoising, JPEG deblocking, and super-resolution. They discovered that models optimized with the closely related ℓ_1 loss

$$\mathcal{L}_{\ell_1}(\hat{\mathbf{y}},\mathbf{y}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_1 \tag{2.37}$$

produced better results MSE-wise than otherwise identical models trained with ℓ_2 loss. They attributed this difference to ℓ_2 loss' higher tendency to get stuck in a local minimum during training. Using combination of both losses Zhao et al. were able to decrease the MSE even further, with best results given by first using ℓ_1 and then finalizing with ℓ_2.

The motivation of Zhao et al. was to increase the perceived quality of the images, as MSE and its derivatives have been known to be poor estimates of perceived quality. Thus they tested also loss functions based on the SSIM metric and proposed a hybrid approach using a mixture of ℓ_1 and SSIM loss functions. There has been also perceptual loss functions that utilize high level features extracted from neural networks trained for image classification. The distance between the features extracted from super-resolved image and the features from the target image is used instead of per-pixel differences. This approach was first introduced for texture synthesis [14, 15], but it has been also employed for super-resolution [28, 35, 49].

As we are not interested in the networks performance on a single training example, we introduce the total cost function

$$\mathcal{J}(\boldsymbol{\theta};\mathbf{X},\mathbf{Y}) = \frac{1}{m}\sum_{i=0}^{m-1}\mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i), \tag{2.38}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are matrices containing all m training example pairs $\mathbf{x}_i$ and $\mathbf{y}_i$ [19, p. 149]. Ultimately we would want to minimize

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}}\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}),\mathbf{y}), \tag{2.39}$$

which is the expected cost of any $\mathbf{x}$ and $\mathbf{y}$ belonging to the data-generating distribution p_{data} [19, p. 272]. As the actual distribution is typically unknown, we can only estimate the cost $\mathcal{J}(\boldsymbol{\theta})$ with the equation 2.39.

The difference between the network output and the target is not always the only thing we want to minimize, and many times it beneficial to keep the trainable parameters $\boldsymbol{\theta}$ at relatively low values. This limits the model's capacity and thus lowers its tendency to overfit. This is achieved by introducing a parameter norm penalty $\Omega(\boldsymbol{\theta})$ to the total cost function

$$\tilde{\mathcal{J}}(\boldsymbol{\theta};\mathbf{X},\mathbf{Y}) = \mathcal{J}(\boldsymbol{\theta};\mathbf{X},\mathbf{Y}) + \alpha\Omega(\boldsymbol{\theta}), \tag{2.40}$$

where α is a hyperparameter affecting the strength of the norm penalty. This regularized total cost $\tilde{\mathcal{J}}$ will be the target of optimization and minimizing it requires balancing the loss $\mathcal{L}$ and the norm penalty Ω. The most widely used norm penalties are the ℓ_2 penalty $\Omega_{\ell_2}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2$ and the ℓ_1 penalty $\Omega_{\ell_1}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$ [19, p. 227–232]. The penalty is typically calculated separately for every layer of the network and the α parameter can be set individually for every layer. Parameter norm penalty is one form of regularization, which refers to any modifications done to the model to prevent overfitting [19, p. 224]. The above mentioned norm penalties are the most widespread regularization methods, and are often referred as just ℓ_2 and ℓ_1 regularization.

2.4.3 Optimization

Although the cost $\mathcal{J}(\boldsymbol{\theta};\mathbf{X},\mathbf{Y})$ is easy to calculate, finding the optimal parameter values

$$\boldsymbol{\theta}_0 = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta};\mathbf{X},\mathbf{Y}) \tag{2.41}$$

is practically impossible due to the high nonlinearity of a typical neural network. The cost function will be non-convex with numerous local minima, and the goal of the optimization is to find a minimum that is low enough for the task at hand. This is done using the gradient descent algorithm, which iteratively updates the parameters $\boldsymbol{\theta}$ by choosing

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) \tag{2.42}$$

as the next set of parameter values after each update. The scalar ϵ is the learning rate parameter affecting the speed of descent and $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y})$ is the gradient of the cost function. The parameter values are updated towards the negative gradient, with the norm of the gradient and the learning rate dictating the magnitude of the updates.

The calculation of the gradient

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Y}) = \frac{1}{m} \sum_{i=0}^{m-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i) \tag{2.43}$$

has to be done for each update step, and its computational complexity is $O(m)$. With datasets large enough to be useful, the cost of this operation becomes prohibitively high. As this cost function is only an estimate of the ideal cost $\mathcal{J}(\boldsymbol{\theta})$, we can estimate the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ with a smaller subset of m' training samples. This subset is called minibatch and the number of samples m' is referred as the batch size. The estimated gradient then becomes

$$\mathbf{g} = \frac{1}{m'} \sum_{i=0}^{m'-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i), \tag{2.44}$$

and the samples in the minibatch are redrawn from the training set for every update step. Every sample will be used only once, until the whole training set has been processed and the cycle begins again. This cycle is called epoch, and the number of epochs tells us how many times the model has seen each individual training sample.

This minibatch gradient descent algorithm is often called stochastic gradient descent (SGD), although originally that term referred only to the extreme case of $m' = 1$ [19, p. 275–276]. SGD, or some variation of it, is used for optimization in practically every modern neural network. Algorithm 2.1 illustrates how a simple version of SGD could be implemented. The learning rate ϵ_k is set separately for each iteration k, as in practical scenarios it has to be decreased as the training progresses. This is due to the noisy gradient estimate used in SGD, which does not converge to zero even if a minimum is reached. [19, p. 290–291]

A common extension to SGD is momentum, which is meant to accelerate the learning process. It is inspired by its physical namesake and introduces a variable $\mathbf{v}$ which represents velocity. The velocity defines the parameter update direction and magnitude, and the velocity is computed at each iteration from the current gradient and previous velocity. The algorithm 2.1 can be extended with momentum by changing the update step to $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon_k \mathbf{g}$ and $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$. The hyperparameter $\alpha \in (0, 1)$ defines how fast the velocity decays.

Require: Learning rate schedule $\epsilon_0, \epsilon_1, \ldots$
  Initialize parameters $\boldsymbol{\theta}$
  $k \leftarrow 0$
  while stopping criterion not met **do**
    Sample a minibatch of m' examples $\mathbf{x}_i$ and $\mathbf{y}_i$ from the training set.
    Compute the gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m'} \sum_{i=0}^{m'-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i)$
    Update the parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

Algorithm 2.1. *Stochastic gradient descent*

Learning rate is one of the most difficult hyperparameters to set, and its effects on the model performance are significant [19, p. 302]. A single learning rate for every parameter is rarely an optimal choice, as the cost function is typically sensitive to some directions in the parameter space and less sensitive to others. This has lead development of adaptive optimizers, which automatically alter the learning rate during the training process separately for every parameter. Common adaptive algorithms are for example AdaGrad by Duchi et al. [8], and Adam by Kingma and Ba [32].

With all gradient based optimizers, the calculation of the actual gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ is done with an algorithm known as back-propagation. It based on the chain rule of calculus, which states that given functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, and scalars x, $z = g(x)$ and $y = f(z) = f(g(x))$, the derivative of y over x can be calculated as

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}z} \frac{\mathrm{d}z}{\mathrm{d}x}. \tag{2.45}$$

This can be easily extended to multidimensional cases. Given functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^m \to \mathbb{R}^n$, vectors $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{z} = g(\mathbf{x})$, and scalar $y = f(\mathbf{z}) = f(g(\mathbf{x}))$, the partial derivative of y becomes

$$\frac{\partial y}{\partial x_i} = \sum_{j=0}^{n-1} \frac{\partial y}{\partial z_j} \frac{\partial z_j}{\partial x_i}. \tag{2.46}$$

This can be written in vector notation as

$$\nabla_{\mathbf{x}} y = \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right)^{\top} \nabla_{\mathbf{z}} y, \tag{2.47}$$

where $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g. [19, p. 201–203]

Equation 2.47 forms the basis of the back-propagation algorithm. It shows that to calculate the gradient of variable $\mathbf{x}$, we can multiply a Jacobian matrix $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ with a gradient $\nabla_{\mathbf{z}} y$. In the back-propagation algorithm this product is calculated for each layer of a neural network recursively, starting from the output and progressing backwards to the input layer. The gradients of the later layers are used to calculate the gradients of earlier layers, i.e. the gradient is propagated backwards in the network. The derivation of the complete algorithm is outside the scope of this thesis, and we refer to the book Deep Learning by Goodfellow et al. [19] for further information on the topic.

2.4.4 Convolutional layers

Fully connected neural networks have a few limitations that make them poorly suitable for image processing tasks, but those limitations can be overcome using convolutional layers. Fully connected networks are limited fo fixed size inputs and outputs, and they become computationally heavy with large input and output sizes, as each node of a layer is connected to every node of the next layer. With n inputs and m outputs, the number of trainable parameters of a fully connected layer becomes nm (or $(n+1)m$ if bias parameters are counted). [19, p. 330]

Convolutional layers work similarly to traditional discrete convolution, and thus the number parameters stays constant regardless of the input size, as the same set of filter coefficients is used for every input segment. In addition to enabling arbitrary-sized inputs and lowering the number of parameters, convolutional layers make the network spatially invariant. This is desirable in super-resolution and similar image processing tasks as the input patch should be processed identically regardless of its location in the input image. [19, p. 331–335]

A simple one-dimensional convolutional layer is shown in Figure 2.6(a). Its input is vector $\mathbf{x} = [x_0, x_1, x_2, x_3, x_4]^\top$, and its output is vector $\mathbf{y} = [y_0, y_1, y_2]^\top$ whose values are defined by the equation

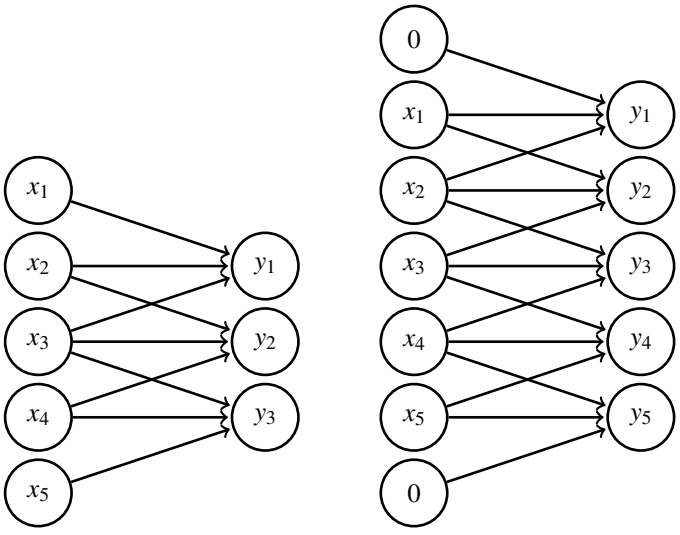
$$y_i = \sum_{j=0}^{2} w_j x_{i+j} + c, \tag{2.48}$$

where w_j are the coefficients of the filter vector $\mathbf{w} = [w_0, w_1, w_2]^\top$ and c is the bias parameter. With a filter of length n, the output will always be $n-1$ shorter than the input, unless some form of input padding is used. Figure 2.6(b) shows a convolutional layer with the same input and filter, but with zero-padding used in the input. [19, p. 342–343]

The output size can be kept identical to the input size by using zero-padding, but sometimes it is desirable to change the data size inside the network. Typical image classification networks decrease the size deliberately with pooling layers to introduce invariance to small input translations, and to lower the computational costs [19, p. 335–339]. Pooling is less useful in super-resolution and thus outside scope of this thesis.

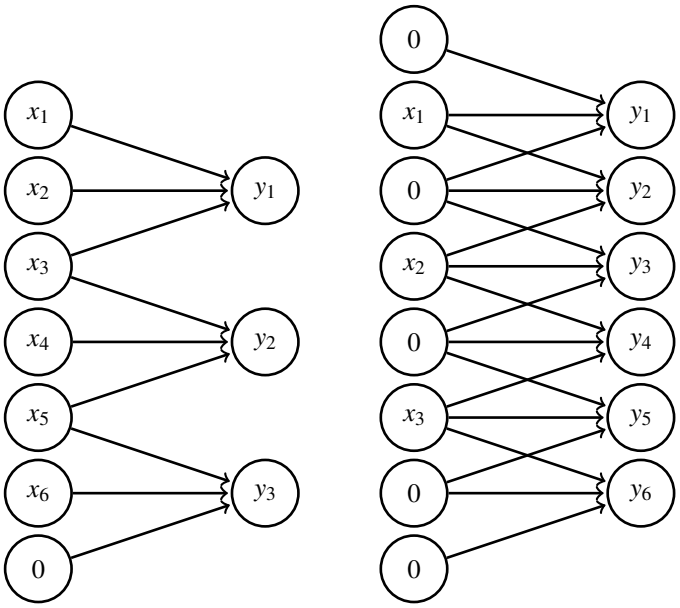
Although super-resolution aims to increase the resolution of images, some SR networks utilize also donwsampling layers in their architecture [21]. Downsampling can be done with strided convolution, which is illustrated in Figure 2.7. Convolution with a stride k progresses in steps of k elements over the input, and produces output of length $\frac{j}{k}$, when input length is j and input padding used. Strided convolution can also thought as normal convolution, where all but every kth output value is discarded.

If we want to increase the size with a factor of k instead of decreasing it, we can use convolution with a stride of $\frac{1}{k}$. This scheme is also known as deconvolution, transposed convolution, sub-pixel convolution, and numerous other names [51], but we will call it



(a) No input padding **(b)** Zero-padding

Figure 2.6. *Two versions a simple one-dimensional convolutional layer with input length of 5 and filter length of 3. The first one has no input padding which leads to smaller sized output and the second one uses zero-padding to preserve the input size in the output.*



(a) Stride 2 **(b)** Deconvolution (stride $\frac{1}{2}$)

Figure 2.7. *Two examples of one-dimensional convolution where data size changes deliberately. First one shows convolution with stride 2, which halves the size. Second one shows deconvolution with factor of 2 (stride $\frac{1}{2}$), which doubles the size of input.*

deconvolution from now on. The term sub-pixel convolution is used for a closely related scheme, which will be explained later in more detail. Deconvolution can be exemplified as normal convolution, where the input vector has zeros added between every input element. This is visualized in Figure 2.7(b) with filter length of 3, stride $\frac{1}{2}$, and zero-padded input.

Until now we have discussed only one-dimensional inputs, outputs, and filters, but images are usually two-dimensional and represented as matrices. In the case of color images we have normally three channels for every pixel, and we need to use three-dimensional arrays to represent them. To extend the concept of vectors and matrices to arbitrary number of dimensions, tensors are typically used [19, p.31]. Further discussion of the tensor theory is beyond the scope of this thesis, but we will use tensors to describe multi-dimensional arrays in the rest of this chapter.

All of the one-dimensional examples we have discussed thus far can be easily extended to multiple dimensions. Images are typically arranged into $n \times m \times c$ tensors, where n is the number of number rows, m is the number of columns and c is the number color channels. With grayscale images $c = 1$, but they are still processed like other three-dimensional tensors. A convolutional layer at the input of a image processing network would have filter kernel of $k_1 \times k_2 \times c$, where k_1 and k_2 hyperparameters define the filter size. Although the filter has a third dimension of length c, this is still considered two-dimensional convolution. The filter length in third dimension is set as the same as the input's third dimension, and the output will be two-dimensional.

Each hidden convolutional layer will typically have f unique filter kernels, and the layer output is arranged as a $n \times m \times f$ tensor, assuming that input size is $n \times m \times c$ and padding is used. Thus the next layer will have filter size of $k_1 \times k_2 \times f$. The output layer will have the number of filters set according to the desired number color channels.

Figure 2.8 illustrates a two-dimensional case of deconvolution, with a $4 \times 4 \times 1$ input tensor, $4 \times 4 \times 1$ filter kernel, stride of $\frac{1}{2}$, and zero-padding. The output is a $8 \times 8 \times 1$ tensor. White squares are used to depict the pixels of the input image, gray squares are the zeros used for padding, and colored squares illustrate the connection between output pixels and the filter coefficients used to calculate those pixels.

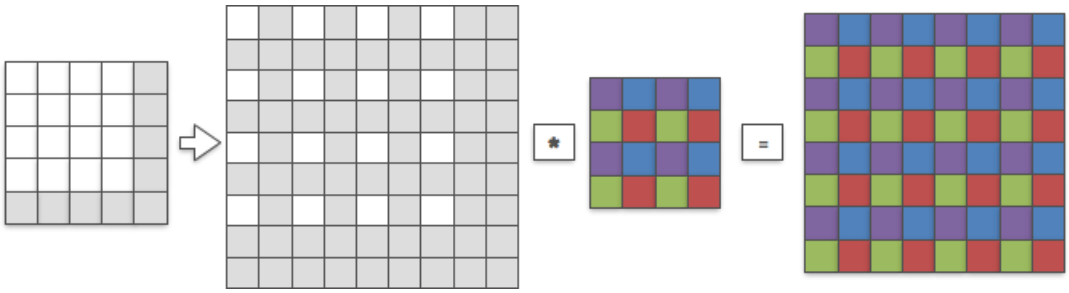


Figure 2.8. *Two-dimensional deconvolution with stride $\frac{1}{2}$ [51].*

Figure 2.9 shows an alternative way to implement similar upscaling operation inside a convolutional network. This method was introduced by Shi et al. [50, 51] and they named it "efficient sub-pixel convolution", but we will refer to it as just sub-pixel convolution. The input and outputs are the same size as in the previous example, but the filter is different. Instead of a single filter, four individual kernels of size $2 \times 2 \times 1$ are used with stride 1, producing an output tensor of size $4 \times 4 \times 4$. The output is then rearranged into a $8 \times 8 \times 1$

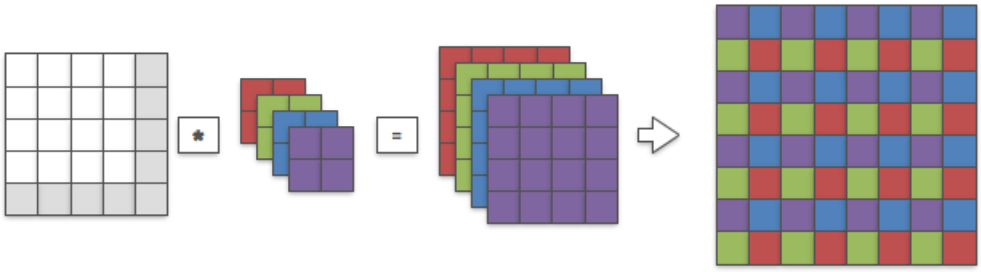


Figure 2.9. *Two-dimensional sub-pixel convolution with scaling factor of 2 [51].*

in the way illustrated by the color coding. In the general case this transforms a $n \times m \times cr^2$ sized tensor to size $rn \times rm \times c$, where r is the scaling factor and c is the number of color channels.

In this simple example case the difference between these two methods is purely implementational, as both methods use identical number of filter coefficients to produce and identically sized output. However, Shi et al. [51] showed that their sub-pixel convolution allows for greater flexibility with the filter sizes used, and is easier to implement efficiently.

There are also other modifications and extensions to convolutional layers, but most of the super-resolution networks utilize only normal convolutional layers, and the above mentioned upsampling convolutions. Concrete examples of those networks are given in Section 3.5.

3. SUPER-RESOLUTION

Super-resolution (SR) aims to produce an high resolution (HR) image, given one or more low resolution (LR) images as input. In contrast to simple interpolation techniques like the ones described in Section 2.1, SR methods aim at recovering or estimating the information missing from the low-resolution image. There are multiple ways to approach the problem and SR algorithms can be classified in numerous ways. The simplest way is to classify them either single image SR (SISR) or multi-image SR (MISR), based on the number of low resolution input images used. MISR tackles the problem with multiple different images depicting the same scene, with each image having different sub-pixel alignments (translation, rotation etc.), different scales, different blurring, or other similar variations between them. As a single low resolution image could have been produced by scaling down infinite number of different high resolution images, using multiple different images introduces additional constraints for the possible high resolution source and thus enables the reconstruction of the high resolution details. [41]

In many practical applications only a single low resolution image is available, which is why SISR algorithms have attracted more research interest in recent years. SISR algorithms can be coarsely split into two different groups. Methods of the first group use only the information available in the source image by exploiting the self-similarity of the scene, whereas the second group utilizes external databases of different LR–HR image pairs. Self-similarity based methods split the image into smaller patches and super-resolve those patches individually using similar patches found elsewhere from the image. Ideally those patches would represent the identical objects, but with different scales, translations, rotations etc. like in the case of MISR. For example, many typical man made scenes i.e. urban environments contain a lot of recurring elements which can be utilized in super-resolution [26].

Algorithms using external image databases employ machine learning techniques to find a typical mapping from high to low resolution, and use that knowledge to estimate a probable high resolution image from the low resolution input. These techniques work especially well in applications where the image content is limited to specific cases i.e. human faces or hand written characters. Multiple different machine learning techniques can be used but especially convolutional neural networks and other deep learning methods have proven popular in recent years [6, 30, 67].

This chapter will give an overview of different approaches to super-resolution in a form of literature review. The next two sections will give overviews of MISR (Section 3.1) and SISR (Section 3.2) in general. Section 3.3 will briefly describe SISR algorithms based on self-similarity. The main focus of this thesis is on methods based on convolutional neural

networks, and Section 3.5 will describe those. Methods based on more classical machine learning techniques will be discussed briefly in Section 3.4.

3.1 Multi-image super-resolution

All digital imaging systems, whether a digital still camera or i.e. a flatbed scanner, produce a discrete, sampled version of the continuous objects present in the scene. As was described in the Section 2.1, frequencies in the original signal higher than half of the sampling frequency will be folded to lower frequencies causing aliasing artifacts. To prevent these artifacts, cameras employ an optical low-pass filter in front of the sensors that blurs the image before sampling. Ideal anti-aliasing filter is impossible to produce, and thus some aliasing will always occur. These aliased frequencies are indistinguishable in a single image, but when there are multiple unique images depicting the same scene, those frequencies can be recovered. This phenomenon is exploited in classical MISR [17].

MISR algorithms were the first ones applied on digital images, but the concept of super-resolution imaging was introduced even before the prevalence of digital imaging. First papers concentrated on the so called optical SR, which tries to overcome the optical resolution limit caused by diffraction, instead of digital sampling [41, 65]. The first optical SR algorithm was introduced by Gerchberg [16] in 1974, although the theoretical foundation was established ten years earlier by Harris [22].

The first application of super-resolution on digital images was by Tsai and Huang [59] in 1984, which used multiple low resolution images to produce a single high resolution image. The method by Tsai and Huang, like the optical SR methods before it, worked in frequency domain. However, frequency domain SR has its limitations [65], and due to that spatial domain methods are more prevalent in modern research [41, 65].

The first spatial domain method was introduced by Peleg et al. [43], and it was based on sub-pixel displacements between the images. Irani and Peleg improved the spatial domain MISR further in 1991 [27] by introducing the iterative back-projection algorithm. This algorithm works by first producing an initial estimate of the HR image, and then producing simulated low-resolution images from the initial estimate. The simulated low-resolution images are compared to the original input images, and if the HR estimate is correct, the images should be identical. If there are differences between the images, the error is back-projected to the high resolution estimate to produce a new estimate and this process is iteratively repeated until the error is minimized. The iterative back-projection has been popular in many subsequent works, including SISR algorithms [11, 26].

There are also other approaches to multi-image super-resolution, but they are out of the scope of this thesis. All MISR algorithms are typically reconstruction based algorithms, which recover only information already existing but indistinguishable in the low resolution images. For these algorithms to work, it is required that every input image contains unique information. As the low resolution images have limited information, the performance of

these methods is also limited especially in high scaling factors. To alleviate this problem, the so called hallucination based methods have been introduced. They can utilize information gathered from external database of HR–LR correspondences and create details that do not exist in the low resolution image. Although this approach could be integrated to multi-image reconstruction methods, it is mostly employed in single image SR and thus will discussed in more detail in the next section [41].

3.2 Single image super-resolution

In many applications it is infeasible or even impossible to acquire multiple images of the same scene, which has lead to development of SISR algorithms [41]. This approach has been the main interest in recent SR research, mostly because of its broad applicability but also due to recent advances in machine learning techniques which have improved the performance significantly [6, 30, 67].

To recreate the information missing in the low resolution, extra information has to be acquired somehow. Since only a single image is available, the reconstruction approach of MISR is impossible. The details have to be created by either using machine learning methods with external database of LR–HR exemplar pairs, or utilizing the self-similarity of the low resolution image. The latter method is enabled by the fractal nature of typical scenes, as image elements tend to recur throughout the image in different scales, translations, rotations etc. [26]. Self-similarity methods can utilize some of the reconstruction techniques typical of MISR algorithms [17], as the image is processed in patches and similar patches can be considered as individual images depicting the same object.

Self-similarity based methods have the advantage of not requiring an extensive database of training images, but they have other drawbacks which have shifted the research focus towards learning-based methods. Self-similarity based methods have typically high computational cost for the processing of an image, since an extensive search of similar patches is required for each image patch processed [26]. Learning-based methods require a computationally intensive training process. However it is done only once and the super-resolving of a single image is relatively fast in comparison to self-similarity based methods. Self-similarity based methods produce good results only for images that have large amounts of similar patches, which is not true for most natural images [26]. In those cases machine learning methods typically outperform those based on self-similarity. Popularity of learning-based methods has also increased due to recent advances in convolutional neural networks and other deep learning techniques utilized in other image processing applications and they have large potential also in super-resolution [6, 30, 67].

3.3 Self-similarity based SISR

First single image SR algorithm utilizing only the self-similarity within a low resolution image was introduced by Ebrahimi and Vrscay in 2007 [9]. It was based on local scale

invariance, which means that image patches are similar to themselves within small scaling factors. For example, a sharp and straight transition between two flat surfaces will appear identical across different scales. Glasner et al. took a different approach in 2009 [17], with an algorithm based on the observation that typically similar image patches occur repeatedly within an image, both with the same scale and different scales. The low resolution image is processed patch by patch and for each patch the image is searched for one or more similar patches. If multiple patches within same scale are found, those can be used with traditional multi-image SR methods for scaling up the input patch. If at least one similar patch with larger scale is found, it can be used directly for estimating the missing high-frequency contents.

Freedman and Fattal combined the two approaches mentioned above and improved on the computational costs, by searching for similar patches within the local neighborhood of the target patch [13]. They also applied the method for video in addition to still images. Until 2015 all of the self-similarity based methods had considered only translated, scaled and rotated patches when searching similar patches, but Huang et al. [26] introduced SelfExSR, that utilizes also perspective and affine transformations. Their method was aimed specifically at images of buildings, urban environments and other man-built structures. Those images typically contain planar structures with recurring elements, whose perspective distortion can be easily estimated. Their method performs comparably to other methods of the time in terms of PSNR, but the computational complexity of SelfExSR is significantly higher than of the competing methods.

Self-similarity based methods have been used extensively also in other image processing tasks, especially in denoising, and some of those methods have inspired also SISR algorithms. A notable example of this is the BM3D algorithm by Dabov et. al. [5] originally published in 2007. This BM3D paradigm has been applied for super-resolution by Egiazarian et. al. first in 2007 [10] and later on in 2015 [11]. It utilizes both sparsity and non-local self-similarity by searching for a group of similar patches, which are arranged to a three-dimensional block. The block is then collaboratively filtered along all three dimensions. This idea was further refined by Cruz et al. in 2017 [4] with their WSD-SR method. It replaces the three-dimensional filtering with one-dimensional Wiener filter working in the similarity domain, which is the third dimension of the patch block with Haar-transform applied along it. Both WSD-SR and its predecessor work iteratively and use back-projection to ensure that the SR estimate corresponds to the LR input.

3.4 Traditional Machine Learning Methods for SISR

Multiple different machine learning methods have been applied to SISR, although most of them have become less popular while the research focus has shifted towards deep learning methods. This is apparent from the results of NTIRE2017 challenge [53], where 18 teams out of 20 used deep neural networks. Out of the remaining two, only RAISR by Romano et al. [46] was based on machine learning. Similar trend continued in the NTIRE2018

challenge [56], where 30 teams out of 31 used deep learning methods, and the remaining one was a method based on RAISR.

Out of the classical ML methods, neighborhood embedding has proven to be a popular and well performing approach [54, 55, 57, 64]. A+ introduced by Timofte et al. [55] in 2015 can be consider the baseline in this family of methods. It combines the approaches of earlier methods ANR [54] and SF [64], by utilizing neighborhood regression and sparse coding. The implementation runs on the central processing unit (CPU) and its computational efficiency is still better than most modern state-of-the-art methods, although its output quality is low on today's standards [1]. A+ was further improved with IA in 2016 by the Timofte et al. [57]. Many of the improvements they introduced can be applied also to other ML based SR methods.

RAISR introduced by Romano et al. [46] is one the fastest algorithms published, and it achieves its performance by learning a set of convolutional filters that adapt to the image contents. The training input images are split into patches and the patches are grouped with a hashing algorithm. For each group a separate set of filters is learned. When super-resolving, the LR image is split into patches and for each patch the correct filter set is chosen based on the same hashing algorithm. The input images are upscaled with bilinear interpolation before processing. The output quality is on the level of A+, but its computational complexity is significantly lower.

3.5 Convolutional Neural Networks based SISR

CNN based SISR methods can be coarsely divided in to two subclasses, based on the way the actual upsampling is done. To distinguish these methods we adapt the terminology used by Shi et al. [51] and refer to these methods as either HR or LR networks, based on the resolution of the input image. HR networks are older of the two, introduced with the first CNN approach to SISR in 2014 by Dong et al. [6]. In that approach the low resolution input image has to be upsampled to the target resolution before processing it in the network. Dong et al. used bicubic interpolation, although any other interpolation method could be used. As the network input is the same size as the output, the same network can be trained to process multiple scaling factors with a single model as demonstrated by Kim et al. [30].

LR networks do the actual upsampling inside the network, and earliest example of this is the sub-pixel convolution method introduced by Shi et al. in 2016 [50]. With this approach the network input is the original low resolution image, and the upsampling is done at the last stage of the network using a learnable filter. This approach has two advantages: first, the computational costs are lower when most the processing is done in the LR space, and second, the learned upscaling filter can perform more optimally than bicubic upscaling. Also, when using the bicubic interpolation as the first processing step, some of the original image information will be lost. Main disadvantage of this approach is, that the same network structure or model cannot be directly used for multiple scaling factors.

As was mentioned in Section 2.4, the networks can also be classified by the loss function they use. Most of the SR methods aim to maximize the PSNR score of the output image and choose either ℓ_1 or ℓ_2 losses as the optimization target. This approach works well when the aim is to reconstruct the HR image as accurately as possible, but the results are not always visually pleasing. It has been shown that the PSNR correlates poorly with the perceived quality, which has prompted researchers to try different perceptual losses [49] for the network training. Another way to produce perceptually better images is adversarial training, in which two different networks are trained in parallel: a discriminative network that tries to distinguish generated images from real ones and a generative network that tries to generate images that fool the discriminative network [20]. These generative adversarial networks (GAN) have been successfully used for SISR by Ledig et al. [35] and Sajjadi et al. [49] and they can produce visually pleasing images with realistic textures and details.

The next two subsections describe in more detail the two above mentioned approaches to upsampling, HR and LR networks. The third subsection overviews the SR methods aiming for maximal perceptual quality instead of high PSNR scores. This section will not cover the theoretical details of convolutional neural networks, as they can be found from Section 2.4 instead.

3.5.1 HR Networks for SISR

The first CNN based method for super-resolution was the SRCNN by Dong et al. [6] published in 2014. The network structure is simple compared to the state-of-the-art CNNs [53], but it managed to outperform the competing methods of its time. It is still a good example of how an end-to-end mapping from a low resolution input to a super-resolved output can be achieved with neural networks, and it serves as a performance baseline method in many newer publications. The network input is an LR image upscaled to the target high resolution using bicubic interpolation, which leads to the name HR network used by Shi et al. [51]. This type of network can also be thought as a network for correcting the errors between the interpolated image and the ground truth image.

The SRCNN network structure is visualized in Figure 3.1. It is a fully convolutional feed-forward network consisting of three convolutional layers with ReLU (rectified linear unit) activations between them. The authors named the three stages as "Patch extraction and representation", "Non-linear mapping" and "Reconstruction" to draw an analogy to sparse coding based SR methods and to show that those can be viewed also as convolutional networks.

The filter sizes f_1 and f_3 shown in the Figure 3.1 were chosen by the authors to be 9 and 5 respectively, in which the case size of the input patch used to construct a single output pixel (the receptive field of the network) is 13×13 pixels. SRCNN does not use any padding in the convolutional layers, which makes the output image 12 pixels smaller than input on both dimensions.

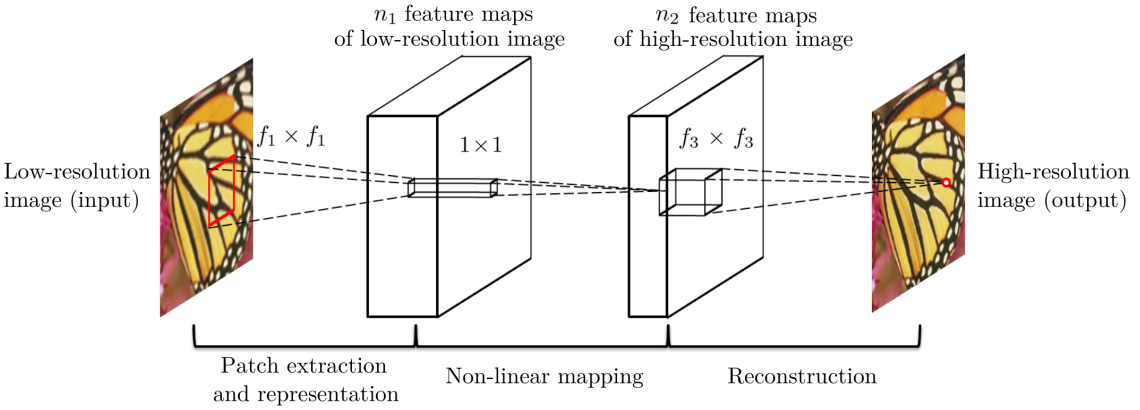


Figure 3.1. *The network structure of SRCNN [6].*

In 2015 Kim et al. [30] introduced VDSR, which improves over SRCNN in a few ways. First, it increases the number of convolutional layers from 3 to 20. It uses the same filter size of 3×3 on each layer, which increases the receptive field to 41×41. Additionally zero padding is used on every convolutional layer, which keeps the image size unchanged throughout the network. The second notable improvement is the residual connection. Instead of passing the whole LR input image through the network like SRCNN, VDSR network is trained only to estimate the difference of bicubic upscaled input and the desired HR output. The VDSR structure is visualized in Figure 3.2.

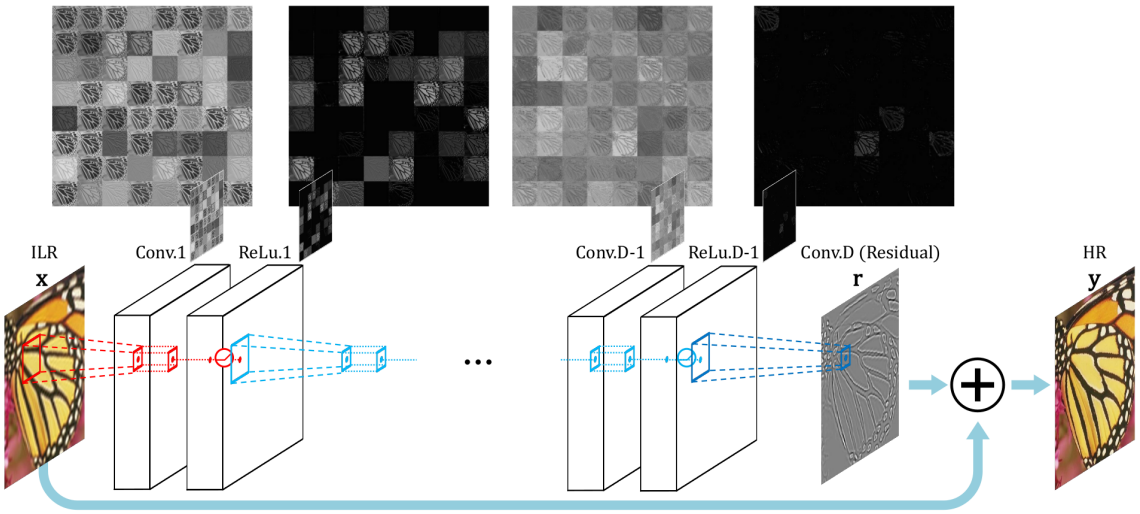


Figure 3.2. *The network structure of VDSR [30].*

A year later Kim et al. [31] introduced DRCN, a method very similar to VDSR, with the biggest difference being the recursive structure of the network. One of the convolutional layers and its ReLU activation is used multiple times in a single forward pass. This layer's output is feeded back to its input up to 16 times, forming a loop inside the network. All 16 intermediate outputs are used when constructing the final output image. Every convolutional layer uses 3×3 filters, and through the longest chain the image passes 20 convolutional layers. This produces 41×41 receptive field identical to VDSR with less

trainable parameters. The performance is almost identical to VDSR in terms of PSNR, but the reduced parameter count comes with the cost of increased complexity and more difficult training process.

In 2017 Zhang et al. [67] introduced DnCNN, which uses the same model for three different tasks: multi scale SR, Gaussian denoising, and JPEG deblocking. The network structure is almost identical to VDSR, with the exception of added batch normalization layers in the hidden layers. In SR task it performs almost identically with VDSR, while still performing well in the denoising and deblocking tasks.

3.5.2 LR Networks for SISR

Although using a bicubic upscaling as preprocessing simplifies the network design and enables multiple scaling factors with a single model, in 2016 Shi et al. demonstrated that this approach is sub-optimal and introduces unnecessary computational overhead [50]. They proposed a scheme where the image is upsampled at the last stage of the network with a method they called sub-pixel convolution. They named the network ESPCN, and its structure is visualized in Figure 3.3. The main advantages of doing the feature extraction in LR space are the lower computational costs and memory requirements, and more optimal performance when the upsampling filters can be learned separately for each feature.

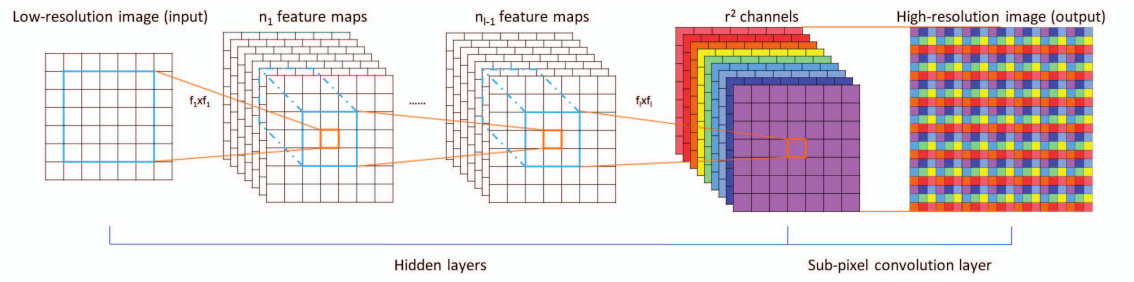


Figure 3.3. *The network structure of ESPCN for scale factor of r [50].*

During the same year Dong et al. [7] independently proposed a similar scheme with their improved version of SRCNN utilizing deconvolution as the upsampling layer. Both deconvolution and sub-pixel convolution can achieve identical results when the filters are learned, and the differences are mostly in the implementation. A more detailed explanation of both upsampling approaches can be found from the Section 2.4.

Ledig et al. [35] improved upon the ESPCN concept with their SRResNet architecture by incorporating the residual block structure originally introduced by He et al. [23] for image classification. SRResNet takes the residual connection of VDSR even further, by arranging the layers to blocks of two consequent convolutions and connecting the input of each block to its output. The residual connections are used to alleviate the problem of vanishing gradients and thus ease the training of deeper networks. The SRResNet structure is shown in Figure 3.4. Although their paper mainly concentrated on perceptual quality

with their SRGAN architecture, their SRResNet model optimized for MSE outperformed DRCN by a clear margin [35].

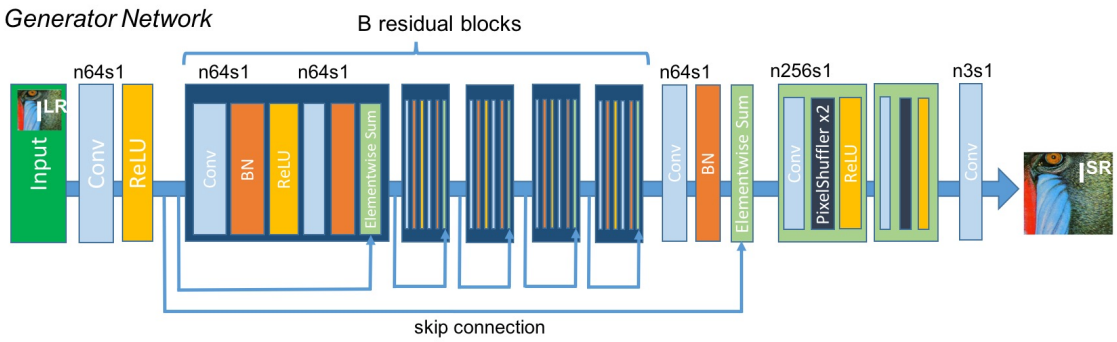
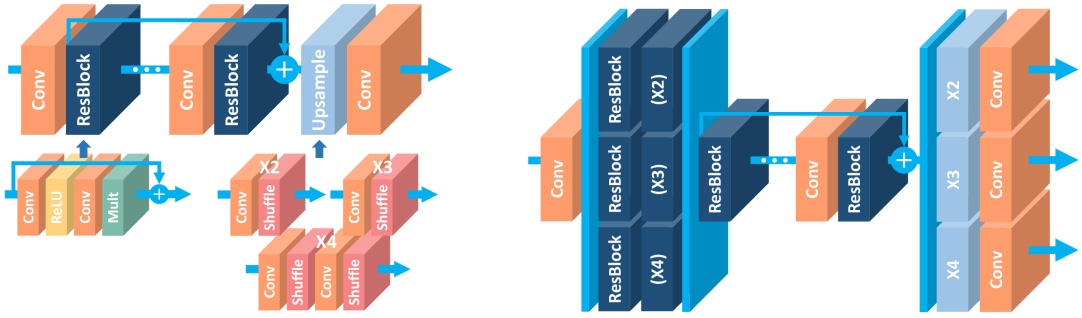


Figure 3.4. *The network structure of SRResNet [35].*

SRResNet structure proved out be a popular starting point for contestants in the NTIRE2017 Challenge on Single Image Super-Resolution [53], and the winner of the competition, EDSR by Lim et al.[36], is very similar to SRResNet with few notable differences. EDSR leaves out the batch normalization completely, which lowers the memory usage and increases the performance. In addition, they use a constant scaling of the residuals to keep the training procedure more stable. These two techniques allow the increasing of the network depth and filter count, which leads state-of-the-art results in terms of PSNR. They also introduce a multi-scale architecture called MDSR, where the input and output sections of the network are separate for scale factors of 2, 3, and 4, but the central layers are shared. Both networks are shown in Figure 3.5.



(a) EDSR **(b)** MDSR

Figure 3.5. *The network structures of EDSR and MDSR [30].*

The residual block structure was taken even further by the DenseNet architecture introduced by Huang et al. in 2017 [25], in which output of every layer is directly connected to the inputs of every subsequent layer. It was first utilized for SISR by Tong et al. in 2017 [58] with their SRDenseNet method, and later extended by Wen et al. in 2018 [63] with the DRNet architecture.

Dense connections are also used in the D-DBPN network by Haris et al. [21], but otherwise it has a very different approach in comparison to other SISR networks. Instead of doing all

feature extraction in LR space and upsampling only at the very last stage, it incorporates multiple upsampling units in conjunction with immediate downsampling. This approach is inspired by the iterative back-projection algorithm originally introduced for MISR by Irani and Peleg [27] and the authors name this approach as deep back-projection network. The network architecture is illustrated in Figure 3.6.

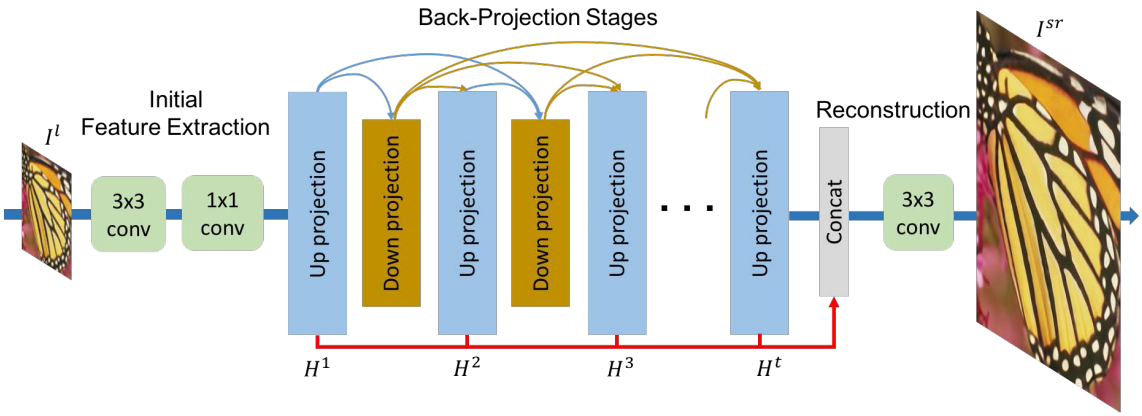


Figure 3.6. *The network structure of D-DBPN [21].*

D-DBPN was one the winners of the NTIRE 2018 Challenge on Single Image Super-Resolution [56], and three other methods shared the first place with it on the competition track with bicubic downsampling and scaling factor of 8. One those was utilizing the EDSR network structure and two others utilized a pyramid structure where the upsampling is done gradually, by increasing the resolution in multiple steps. ProSR by Wang et al. [61] was other one of those pyramid methods, and its architecture is shown in Figure 3.7.

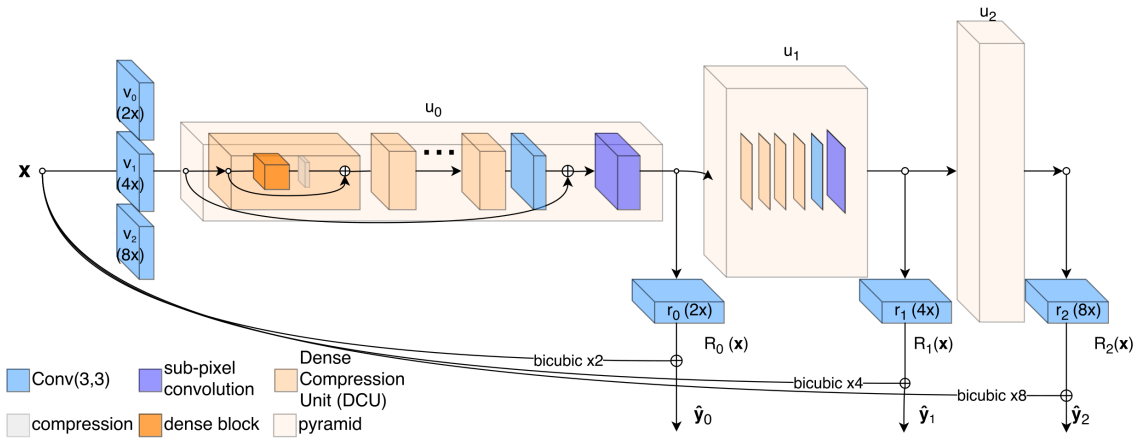


Figure 3.7. *The network structure of ProSR [61].*

ProSR network can produce multiple scales with the same model, with a separate input layer used for each factor like in the MDSR network. It also utilizes a similar residual connection between input and output as VDSR, with bicubic upsampling used for scaling the input for residual estimation. Dense connections are used within the pyramid levels and sub-pixel convolution is used for upsampling at the last stage of each pyramid level.

There are of course other CNN architectures for SISR, but they are outside the scope of this thesis. The above mentioned ones are the most prevalent and represent the state-of-the-art super-resolution, as can be seen in the results of recent SISR competitions [3, 53, 56].

3.5.3 Networks Optimized for Perceptual Quality

Although optimizing for ℓ_1 or ℓ_2 losses will produce the highest PSNR scores, the results are typically overly smooth and lack high frequency details and texture. One approach to gain more visually pleasing results, is to change the optimization target from MSE to a perceptual loss function. In the context of texture synthesis, Gatys et al. [14, 15] introduced the idea of using the features from a pre-trained image classification network for determining the perceptual similarity of two images. This approach was applied to super-resolution first by Johnson et al. [28] in 2016, who used the features from VGG image classification network [52].

Ledig et al. [35] and Sajjadi et al. [49] independently developed this approach further by using a generative adversarial network (GAN) in addition to the perceptual loss from VGG features. GANs consist of two different networks: one generating the images, and another network (the adversarial network) trying to discriminate between generated images and ground-truth target images. They are trained in parallel to compete with each other: generative network tries to fool the adversarial network and adversarial network is trained to discriminate better.

Ledig et al. use the SRResNet architecture with their SRGAN method [35], and Sajjadi et al. use similar structure with their ENet-PAT method. ENet-PAT does the upscaling inside network like SRResNet, but uses nearest neighbor interpolation instead of sub-pixel convolution. Both methods produce perceptually pleasing SR images, but their PSNR and SSIM scores are significantly lower than of any MSE optimized method. In some cases even bicubic interpolation scores higher. Ledig et al. did extensive mean-opinion-score testing for different SR methods, and concluded that the SRGAN produces perceptually better images by a clear margin in comparison to methods optimized for low MSE.

Wang et al. extended their ProSR architecture with adversarial training, and named this method as ProGanSR [61]. In contrast to SRGAN and ENet-PAT, only adversarial loss is utilized in the training process of ProGanSR, and no ℓ_1, ℓ_2 or perceptual losses are used. They adopt a curriculum learning scheme, where the network is trained with samples of gradually increasing scaling factors and difficulty. The authors argue that this scheme, along with the residual output structure of ProSR, stabilizes the training process and enables them to use only adversarial loss as the optimization target.

4. TESTING METHODOLOGY

Four CNN-based algorithms with publicly available implementations were chosen for this comparison: D-DBPN [21], EDSR [36], ENet-PAT [49], and VDSR [30]. In addition to CNN-based methods, A+ [55] was chosen to represent classical ML methods, and SelfExSR [26] and WSD-SR [4] were selected to represent the self-similarity based approach. Main features of these methods are shown in Table 4.1.

Table 4.1. *Summary of the compared SR methods.*

Method	Type	Platform (CNN library)	Network type	Loss function	No. of trainable parameters
D-DBPN	CNN	Python (PyTorch)	LR	ℓ_2	5.95 M (x2) 10.4 M (x4)
EDSR	CNN	Lua (Torch)	LR	ℓ_1	43 M
ENet-PAT	CNN	Python (Tensorflow)	LR	Perceptual, adversarial and texture losses	853 k
VDSR	CNN	MATLAB (MatConvNet)	HR	ℓ_2	670 k
A+	Neighbourhood embedding	MATLAB	-	-	-
SelfExSR	Self-similarity	MATLAB	-	-	-
WSD-SR	Self-similarity	MATLAB	-	-	-

Comparisons were made on four widely used datasets: Set5 by Bevilacqua et al. [2], Set14 by Zeyde et al. [66], Urban100 by Huang et al. [26], and DIV2K validation set from NTIRE 2017 competition [1, 53]. As these sets are common, they have also been used during the development of most of the selected methods. The results on these datasets have likely affected the choice of hyperparameters for those methods, and thus the results on these datasets cannot be considered as unbiased estimates of the true generalization capability of those methods. For this reason TAMPERE17 image database by Ponomarenko et al. [44] has been included in the comparison. None of the chosen methods report results for TAMPERE17, and most of them even predate the publication of this database, which makes the database an optimal choice for this comparison.

Three scaling factors (2, 3 and 4) were used for all methods except for ENet-PAT and D-DBPN. ENet-PAT supports only factor of 4 and D-DBPN supports factors 2 and 4. All HR input images are first downscaled with bicubic downsampling (MATLAB's imresize function), and then super-resolved with each of the chosen methods. The resulting images are then compared to the ground truth images with PSNR and SSIM metric. When calculating the metrics, the borders of the image are discarded, with the width of border set to the scaling factor. It is common to calculate these metrics only on the luma (Y) channel of YCbCr color space, as human visual system is more sensitive to changes in

brightness than changes in color. In addition to luma channel metrics, we also calculate both of the metrics on the full RGB image for comparison. All methods are integrated into a MATLAB testbench which automates the downscaling, super-resolving and quality metric calculation.

All except one of the methods were tested on a Linux server with a 2.4 GHz Intel Xeon E5-2640 v4 CPU, 16 GB of usable RAM, and a nVidia Tesla P100 GPU with 16 GB of memory. The only exception is SelfExSR, which depends on Windows-only binaries and it was tested on Windows 10 desktop with 3,4 GHz Intel i5-4670 CPU and 16 GB of RAM. The SelfExSR utilized two CPU cores concurrently and all the other methods were running on a single thread. The CNN based methods, D-DBPN, ENet-PAT, EDSR and VDSR, utilize also the GPU for the computations.

The computational complexity of the algorithms was compared by calculating the average processing time of a single image in each dataset. This comparison is not completely fair for non-MATLAB methods, due to the limitations of the MATLAB testbench and the way the integration of external non-MATLAB methods has been done as Python and Lua methods cannot be called directly from MATLAB. When testing external methods, the testbench stores the LR input image to disk and calls the external method through a system call. External method then initializes itself (loads the network model etc.), reads the input image from disk, super-resolves the image and stores the output to disk. This output image is then read from the disk by the testbench, which then resumes the normal process. All this overhead processing is counted when measuring the processing time of a single image. Native MATLAB methods do not need similar extraneous file storing or reading and the initialization is done only once and it is not counted in the measured processing time.

Due to the limited amount of GPU memory available, some of the CNN based methods are not able to process the largest images in a single pass. Thus the input images have to be split into smaller pieces before processing and then stitched back together to produce the final image. This is done recursively so that the input image patch is split into four equally sized pieces until the pixel count of the patch is under a specified value. When splitting the patch, extra padding is left so that the patches overlap a little. For EDSR and D-DBPN this splitting was already implemented by their original authors, but for ENet-PAT and VDSR it had to be implemented separately. The maximum pixel count was set through trial-and-error to highest possible value that enabled processing of all the images with all scaling factors. The values were 276677 for D-DBPN, 70000 for EDSR, 260100 for ENet-PAT, and 1050000 for VDSR. The padding parameters were left to the default values for EDSR (10 pixels) and D-DBPN (16 pixels) and set to 32 pixels for ENet-PAT and VDSR. For D-DBPN, the self ensemble mode was enabled, in which the output is produced by averaging over 8 different SR images, each produced with a different combination of flips and rotation.

5. RESULTS

Quantitative analysis was performed with PSNR and SSIM metrics, by calculating them both on the luma channel and full RGB image. Results were averaged over all the images of a dataset and the results are shown in following tables. Tables 5.1, 5.2 and 5.3 show the PSNR and SSIM results for scaling factors 2, 3 and 4 respectively. Higher values indicate better performance, and in each row the best performing method is highlighted with blue color, the second best with green and the third best with red.

Table 5.1. *Average PSNR and SSIM scores for scaling factor of 2.*

Dataset	Metric	Bicubic	A+	SelfExSR	WSD-SR	VDSR	EDSR	D-DBPN
Set5	$PSNR_Y$	33.68	36.58	36.58	37.21	37.56	38.21	**38.23**
	$PSNR_{RGB}$	31.80	34.34	34.50	35.07	35.17	**36.03**	36.02
	$SSIM_Y$	0.9306	0.9547	0.9547	0.9577	0.9591	0.9614	**0.9614**
	$SSIM_{RGB}$	0.9093	0.9357	0.9372	0.9405	0.9404	**0.9450**	0.9449
Set14	$PSNR_Y$	30.24	32.29	32.37	32.83	32.99	33.86	**33.89**
	$PSNR_{RGB}$	28.68	30.49	30.73	31.12	31.11	32.16	**32.19**
	$SSIM_Y$	0.8694	0.9060	0.9061	0.9103	0.9123	0.9198	**0.9207**
	$SSIM_{RGB}$	0.8468	0.8807	0.8853	0.8887	0.8864	0.8995	**0.9004**
Urban100	$PSNR_Y$	26.88	29.24	29.55	30.29	30.75	**32.98**	32.71
	$PSNR_{RGB}$	25.44	27.66	28.06	28.71	29.05	**31.32**	31.06
	$SSIM_Y$	0.8412	0.8946	0.8978	0.9067	0.9144	**0.9361**	0.9339
	$SSIM_{RGB}$	0.8280	0.8821	0.8875	0.8962	0.9017	**0.9271**	0.9248
DIV2K	$PSNR_Y$	32.44	34.53	34.54	34.91	35.39	**36.57**	36.49
	$PSNR_{RGB}$	31.04	32.98	33.09	33.41	33.75	**35.07**	34.99
	$SSIM_Y$	0.9042	0.9328	0.9326	0.9364	0.9402	**0.9488**	0.9481
	$SSIM_{RGB}$	0.8933	0.9225	0.9237	0.9272	0.9298	**0.9411**	0.9403
TAMPERE17	$PSNR_Y$	30.50	32.23	32.28	32.68	33.07	34.25	**34.29**
	$PSNR_{RGB}$	28.75	30.21	30.45	30.70	30.86	32.23	**32.27**
	$SSIM_Y$	0.8517	0.8899	0.8908	0.8948	0.8984	0.9137	**0.9138**
	$SSIM_{RGB}$	0.8310	0.8687	0.8738	0.8767	0.8767	0.8978	**0.8979**

The results are consistent across different datasets and scaling factors, with EDSR and D-DBPN being the best performing methods by a clear margin. On scaling factors 2 and 4, and on all four metrics, EDSR outperforms D-DBPN on Urban100 and DIV2K datasets by a small margin, but D-DBPN outperforms EDSR consistently on Set14, albeit with a very small margin. The results on Set5 are more even, with D-DBPN scoring higher on Y-channel metrics and EDSR on RGB-metrics. Differences on Set5 scores are neglible, but consistent across both scaling factors. EDSR attains higher SSIM scores on TAMPERE17 with scaling factor 4, but DPBN performs better on the PSNR metrics and on all metrics of scaling factor 2. Scaling factor 3 is not supported by D-DBPN, thus EDSR is the clear winner in all test cases. Otherwise their results are so close that they can be considered equally well performing.

The third place after EDSR and D-DBPN is contested by WSD-SR and VDSR, with VDSR scoring higher on most of the metrics. The results even out on scaling factor 4, with WSD-SR exceeding VDSR marginally on PSNR scores of Set5 and Set14. On all scaling factors VDSR outperforms WSD-SR consistently on DIV2K and TAMPERE17 datasets.

Table 5.2. *Average PSNR and SSIM scores for scaling factor of 3.*

Dataset	Metric	Bicubic	A+	SelfExSR	WSD-SR	VDSR	EDSR
Set5	$PSNR_Y$	30.40	32.60	32.65	33.50	33.68	**34.68**
	$PSNR_{RGB}$	28.63	30.56	30.76	31.51	31.51	**32.67**
	$SSIM_Y$	0.8685	0.9082	0.9094	0.9187	0.9212	**0.9293**
	$SSIM_{RGB}$	0.8381	0.8798	0.8834	0.8929	0.8934	**0.9055**
Set14	$PSNR_Y$	27.54	29.13	29.24	29.72	29.75	**30.44**
	$PSNR_{RGB}$	26.07	27.48	27.72	28.12	28.03	**28.89**
	$SSIM_Y$	0.7751	0.8197	0.8225	0.8298	0.8319	**0.8459**
	$SSIM_{RGB}$	0.7480	0.7891	0.7972	0.8020	0.8002	**0.8215**
Urban100	$PSNR_Y$	24.46	26.05	26.46	26.95	27.13	**28.82**
	$PSNR_{RGB}$	23.03	24.52	24.99	25.40	25.52	**27.25**
	$SSIM_Y$	0.7363	0.7987	0.8103	0.8209	0.8286	**0.8663**
	$SSIM_{RGB}$	0.7154	0.7784	0.7926	0.8023	0.8080	**0.8505**
DIV2K	$PSNR_Y$	29.65	31.08	31.16	31.47	31.74	**32.75**
	$PSNR_{RGB}$	28.25	29.56	29.73	29.98	30.15	**31.29**
	$SSIM_Y$	0.8308	0.8645	0.8667	0.8726	0.8772	**0.8937**
	$SSIM_{RGB}$	0.8135	0.8470	0.8515	0.8566	0.8596	**0.8801**
TAMPERE17	$PSNR_Y$	28.03	29.27	29.35	29.68	29.99	**31.04**
	$PSNR_{RGB}$	26.33	27.39	27.60	27.81	27.97	**29.17**
	$SSIM_Y$	0.7536	0.7975	0.8006	0.8067	0.8125	**0.8363**
	$SSIM_{RGB}$	0.7235	0.7667	0.7743	0.7785	0.7811	**0.8113**

Table 5.3. *Average PSNR and SSIM scores for scaling factor of 4.*

Dataset	Metric	Bicubic	ENet-PAT	A+	SelfExSR	WSD-SR	VDSR	EDSR	D-DBPN
Set5	$PSNR_Y$	28.43	28.85	30.30	30.32	31.39	31.36	32.50	**32.53**
	$PSNR_{RGB}$	26.70	26.99	28.33	28.51	29.46	29.26	**30.57**	30.56
	$SSIM_Y$	0.8108	0.8156	0.8595	0.8614	0.8815	0.8830	0.8982	**0.8983**
	$SSIM_{RGB}$	0.7730	0.7738	0.8228	0.8282	0.8485	0.8471	**0.8685**	0.8679
Set14	$PSNR_Y$	26.00	25.93	27.32	27.45	27.98	27.97	28.72	**28.77**
	$PSNR_{RGB}$	24.57	24.51	25.75	25.99	26.43	26.33	27.24	**27.28**
	$SSIM_Y$	0.7037	0.6852	0.7506	0.7545	0.7657	0.7674	0.7868	**0.7869**
	$SSIM_{RGB}$	0.6742	0.6538	0.7179	0.7275	0.7351	0.7333	0.7609	**0.7615**
Urban100	$PSNR_Y$	23.14	23.63	24.34	24.80	25.16	25.17	**26.65**	26.55
	$PSNR_{RGB}$	21.71	22.10	22.83	23.34	23.63	23.60	**25.12**	25.03
	$SSIM_Y$	0.6596	0.6944	0.7204	0.7392	0.7502	0.7535	**0.8043**	0.7987
	$SSIM_{RGB}$	0.6333	0.6641	0.6946	0.7166	0.7259	0.7273	**0.7838**	0.7781
DIV2K	$PSNR_Y$	28.11	27.61	29.27	29.35	29.65	29.80	**30.74**	30.73
	$PSNR_{RGB}$	26.68	26.09	27.75	27.92	28.16	28.22	**29.28**	29.27
	$SSIM_Y$	0.7750	0.7544	0.8088	0.8121	0.8196	0.8236	**0.8453**	0.8440
	$SSIM_{RGB}$	0.7526	0.7249	0.7860	0.7920	0.7981	0.8004	**0.8271**	0.8258
TAMPERE17	$PSNR_Y$	26.61	25.83	27.61	27.69	28.00	28.17	29.14	**29.16**
	$PSNR_{RGB}$	24.93	24.13	25.79	25.99	26.19	26.26	27.35	**27.38**
	$SSIM_Y$	0.6808	0.6482	0.7228	0.7277	0.7349	0.7404	**0.7684**	0.7673
	$SSIM_{RGB}$	0.6445	0.6088	0.6859	0.6951	0.6998	0.7029	**0.7372**	0.7363

A+ and SelfExSR perform very similarly, with a clear margin between them and WSD-SR and VDSR. SelfExSR manages to outperform A+ clearly on Urban100 dataset, which is a set introduced by the authors of SelfExSR, but on all the other datasets the differences are minor.

ENet-PAT is the only method in this comparison, that has not been optimized for PSNR or any other reconstruction error metric. Instead, the authors have aimed for a purely perceptual quality, which is evident in the PSNR and SSIM scores of ENet-PAT in Table 5.3. ENet-PAT manages to outperform bicubic upsampling only on Set5 and Urban100, and on other datasets it looses on all metrics.

The computational complexity of the methods was analyzed by measuring the average time it took to process a single image, and the results are visible in Table 5.4. VDSR is overall the fastest method, with only A+ surpassing it on Set5 with scaling factors 3 and 4. In this comparison, the self-similarity based methods SelfExSR and WSD-SR are orders of magnitude slower than other methods, with WSD-SR being slowest on all datasets and scales. As was mentioned in Chapter 4, the overhead caused by the integration of non-MATLAB methods into the testbench inflates the measured time for D-DBPN, EDSR and ENet-PAT. As this overhead is mostly caused by the method initialization, it is not strongly dependent on the size of the input image. With the larger images of the DIV2K dataset, the overhead becomes relatively smaller. Thus EDSR and ENet-PAT manage to outperform the A+ on DIV2K, which is otherwise the second fastest method. It is also worth noting, that as VDSR upsamples the input image to target resolution before passing it through the network, its processing time is not dependent on the scaling factor in this setup. All other methods are consistently slower on smaller scale factors due to larger input images, except for WSD-SR, which is fastest on factor 4 like others, but slowest on scale factor 3.

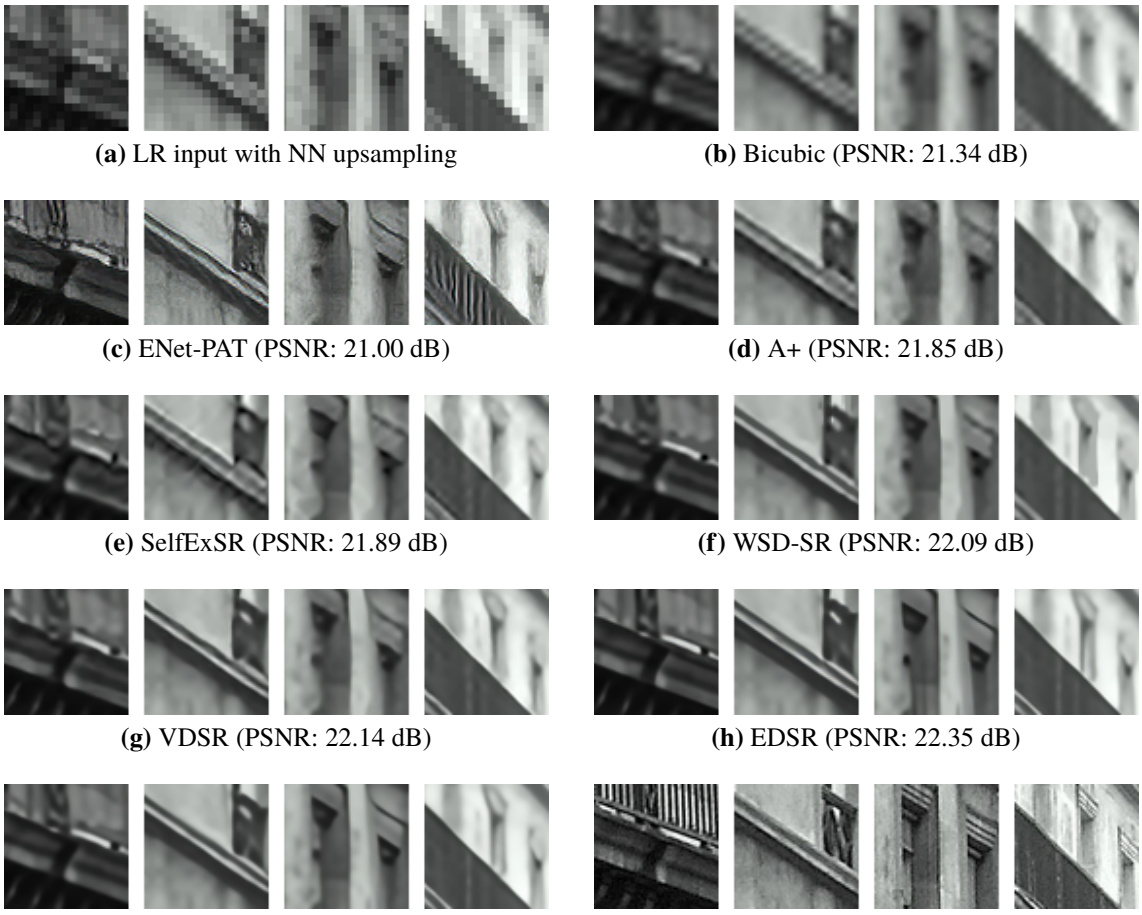
Table 5.4. *Average processing time of a single image for each method, dataset and scaling factor.*

Dataset	Scale	ENet-PAT	A+	SelfExSR	WSD-SR	VDSR	EDSR	D-DBPN
Set5	2		0.84	39.65	60.31	**0.79**	5.22	7.37
	3		**0.44**	28.49	78.18	0.61	5.16	
	4	5.47	**0.33**	24.95	79.16	0.60	5.13	6.86
Set14	2		1.57	95.80	207.49	**0.66**	5.61	9.02
	3		0.88	67.92	238.39	**0.65**	5.35	
	4	5.31	0.67	57.66	213.54	**0.65**	5.22	7.68
Urban100	2		6.16	388.11	951.21	**0.85**	7.94	16.79
	3		3.84	269.04	1136.63	**0.85**	6.52	
	4	5.75	3.02	226.72	986.45	**0.88**	5.97	11.13
DIV2K	2		23.57	1474.76	1955.21	**1.82**	16.65	48.39
	3		15.01	1345.34	2165.90	**1.82**	11.45	
	4	7.26	12.03	882.99	1841.03	**1.82**	8.78	23.73
TAMPERE17	2		1.77	134.23	197.84	**0.66**	5.77	9.22
	3		1.02	88.87	219.47	**0.67**	5.47	
	4	5.40	0.77	79.74	195.66	**0.66**	5.27	7.75

For qualitative analysis 5 images were chosen to highlight the differences between the methods. First two images are numbers 66 and 92 from Urban100, and 64×64 pixel patches extracted from them can be seen in Figures 5.1 and 5.2 respectively. Due to the large size of the original images, full resolution versions are included only in Appendix A on Figures A.1 and A.2. Third example is the image number 858 from DIV2K validation dataset, and patches extracted from it are shown in Figure 5.3 and full size images can be found from Appendix A, Figure A.3. Last two images are numbers 238 and 256 from TAMPERE17 dataset and they are shown in full size in Figures 5.4 and 5.5 respectively.

All figures show the output images from the seven selected SR methods and bicubic interpolation, with their corresponding PSNR_Y scores included. The low resolution input

images and the ground truth HR images have been added for reference. To visualize better the amount of input information, the LR images have been upsampled with NN interpolation to the target resolution. Only scaling factor of 4 has been used for the figures, as the differences between the methods are more visible on higher scaling factors and it is the only factor supported by all of the methods.



(a) LR input with NN upsampling **(b)** Bicubic (PSNR: 21.34 dB)

(c) ENet-PAT (PSNR: 21.00 dB) **(d)** A+ (PSNR: 21.85 dB)

(e) SelfExSR (PSNR: 21.89 dB) **(f)** WSD-SR (PSNR: 22.09 dB)

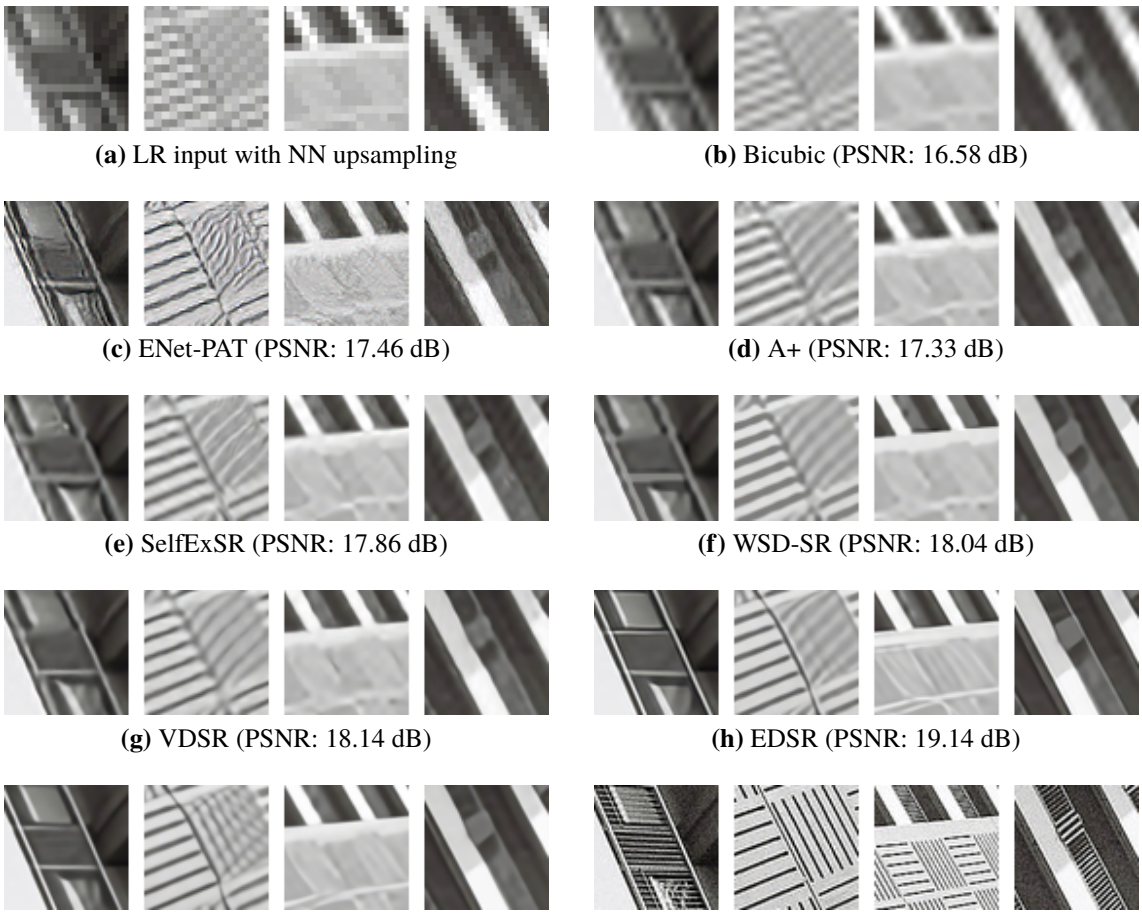
(g) VDSR (PSNR: 22.14 dB) **(h)** EDSR (PSNR: 22.35 dB)

(i) D-DBPN (PSNR: 22.27 dB) **(j)** Ground truth

Figure 5.1. *Patches extracted from image 66 from Urban100 dataset super-resolved with scaling factor of 4.*

Urban100 was the most difficult dataset to super-resolve based on the average PSNR scores, and Figures 5.1 and 5.2 visualize some of the reasons for this. As the images mostly represent buildings and similar man-made structures, there are a lot of simple geometric shapes with sharp edges visible. These features are relatively easy to reconstruct for all of the methods, but the problems arise with the areas between the edges. They often contain high contrast, high frequency details and textures, and some of the images are considerably noisy. Figure 5.1(j) highlights the noise and texture visible in the ground truth image, which is hard to reconstruct for all the methods. PSNR values are low, and all methods except ENet-PAT produce overly smooth and unrealistic surfaces. ENet-PAT on the other hand hallucinates highly distorted and unplausible details. The rightmost patch of Figure 5.1(c) is a good example of this, as ENet-PAT produces repetitive pattern of curly vertical

lines to an area which is relatively smooth on the ground truth image.



(a) LR input with NN upsampling **(b)** Bicubic (PSNR: 16.58 dB)

(c) ENet-PAT (PSNR: 17.46 dB) **(d)** A+ (PSNR: 17.33 dB)

(e) SelfExSR (PSNR: 17.86 dB) **(f)** WSD-SR (PSNR: 18.04 dB)

(g) VDSR (PSNR: 18.14 dB) **(h)** EDSR (PSNR: 19.14 dB)

(i) D-DBPN (PSNR: 19.06 dB) **(j)** Ground truth

Figure 5.2. *Patches extracted from image 92 from Urban100 dataset super-resolved with scaling factor of 4.*

Figure 5.2 highlights another great source of difficulty in the Urban100 dataset. The ground truth image contains a lot of repetitive high contrast line patterns, with frequency contents way beyond the Nyquist frequency limit of the LR image. These features are low-pass filtered into to almost completely flat surfaces in the input image (Figure 5.2(a)), and understandably none of the methods are able to reconstruct these patterns. This leads to very low PSNR values and unplausible output images. Also worth noting is the aliasing visible in the second patch from left, which leads to all methods producing line patterns with wrong orientation. Only D-DBPN manages to produce lines with somewhat correct orientation.

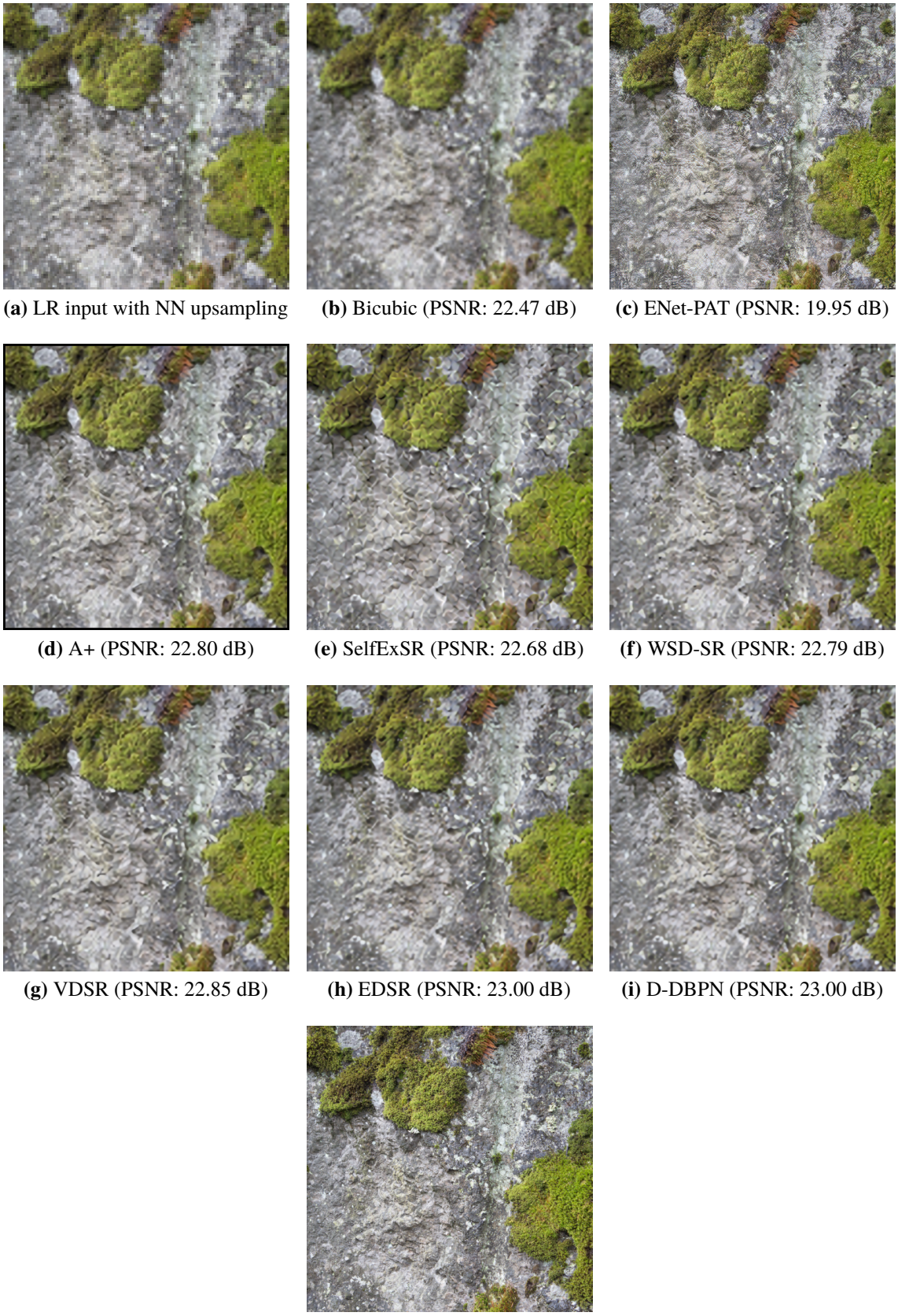
Natural images tend to have a lot of irregular textures, that are very difficult to super-resolve accurately. However, methods optimized for perceptual quality can produce very believable results in these cases. One example of this is image 858 from DIV2K shown in Figure 5.3. ENet-PAT manages to create very natural looking textures, which are almost indistinguishable from the ground truth. All other methods produce very smooth textures, that make the images look very artificial.

(a) LR input with NN upsampling
(b) Bicubic (PSNR: 28.23 dB)
(c) ENet-PAT (PSNR: 27.28 dB)
(d) A+ (PSNR: 29.09 dB)
(e) SelfExSR (PSNR: 29.05 dB)
(f) WSD-SR (PSNR: 29.23 dB)
(g) VDSR (PSNR: 29.36 dB)
(h) EDSR (PSNR: 29.94 dB)
(i) D-DBPN (PSNR: 29.94 dB)
(j) Ground truth

Figure 5.3. *Patches extracted from image 858 from DIV2K validation dataset super-resolved with scaling factor of 4.*

Another example is image 238 from TAMPERE17, a highly textured image of a rock surface covered in moss, which is shown in Figure 5.4. Once again ENet-PAT manages to produce the most realistic looking output, but with a closer look one can distinguish the somewhat noisy and oscillative looking artifacts. With natural images like this, humans have very strong expectations about the textures, and it is very difficult to fool them with artificial details. For all of the PSNR optimized methods this image is almost equally difficult, and the difference in PSNR_Y is only 0.32 dB between SelfExSR and EDSR/D-DBPN. This image also highlights quite well how poorly SelfExSR performs on natural images and it produces similar, highly distinctive artifacts on all images with irregular textures.

Figure 5.5 shows a photo of a bulletin board with various posters, the image number 256 from TAMPERE17. It has a lot of smooth surfaces and sharp edges, that should be easy to super-resolve accurately. Nevertheless the A+ and SelfExSR outputs are relatively blurry, and ENet-PAT produces ringing like artifacts around the edges. Other methods, especially EDSR and D-DBPN produce very sharp and accurate edges. They also manage to improve the readability of the text quite notably, which is evident especially in the upper left corner. Using SR for text processing does have its risks though, as we can see in the upper right



(a) LR input with NN upsampling **(b)** Bicubic (PSNR: 22.47 dB) **(c)** ENet-PAT (PSNR: 19.95 dB)

(d) A+ (PSNR: 22.80 dB) **(e)** SelfExSR (PSNR: 22.68 dB) **(f)** WSD-SR (PSNR: 22.79 dB)

(g) VDSR (PSNR: 22.85 dB) **(h)** EDSR (PSNR: 23.00 dB) **(i)** D-DBPN (PSNR: 23.00 dB)

(j) Ground truth

Figure 5.4. *Image 238 from TAMPERE17 dataset super-resolved with scaling factor of 4.*

(a) LR input with NN upsampling

(b) Bicubic (PSNR: 27.66 dB)

(c) ENet-PAT (PSNR: 27.68 dB)

(d) A+ (PSNR: 28.60 dB)

(e) SelfExSR (PSNR: 28.50 dB)

(f) WSD-SR (PSNR: 28.95 dB)

(g) VDSR (PSNR: 29.30 dB)

(h) EDSR (PSNR: 30.79 dB)

(i) D-DBPN (PSNR: 31.02 dB)

(j) Ground truth

Figure 5.5. *Image 256 from TAMPERE17 dataset super-resolved with scaling factor of 4.*

corner. The "eestec" logo has become "oostoc" in the super-resolved images, and errors like this might be critical in many use cases.

In this subjective analysis, it was noted that higher PSNR might not always signify higher perceived quality, as is the case with some of the ENet-PAT output images. Although, when comparing only PSNR optimized methods, the perceived quality correlates extremely well with PSNR and SSIM scores and the top-performing methods EDSR and D-DBPN consistently produce the most pleasing result.

6. CONCLUSIONS

In this thesis, a literature review and comparative analysis on single image super-resolution was performed. The main focus was on methods utilizing convolutional neural networks. These were primed in Chapter 2, where theoretical background was given on the topics of sampling and interpolation, image quality metrics, and convolutional neural networks. A survey on both the history of super-resolution, and modern developments of it were given in Chapter 3. SISR methods based on traditional machine learning, and methods based on self-similarity were discussed briefly to provide a reference point to CNN based methods. For our comparison, A+ was chosen to represent the traditional machine learning approach, and SelfExSR and WSD-SR to represent self-similarity based methods.

CNN based SISR methods can roughly be divided into two groups, which were the main focus of Chapter 3. The older group contains methods using HR networks, which upsample the image before feeding it to the network. The more modern and efficient approach are the LR networks, that upsample the image at the last stages of the network. Four CNN methods were chosen for comparison: VDSR, EDSR, D-DBPN, and ENet-PAT. VDSR is the only representative of the HR network group, with rest of the methods falling to the latter group. In this comparison EDSR is the purest example of LR networks, as it has a very deep architecture and the upsampling is done at the very last layers, with learnable filters. Also ENet-PAT does the upsampling at the last stages, but uses nearest neighbor interpolation instead. D-DBPN uses learnable filters, but adopts a structure were the data is first upsampled and immediately downsampled. The upsampling-downsampling-scheme is repeated multiple times throughout the network.

In total seven different algorithms were chosen for performance comparison, and they were tested on five datasets. Four of those were widely used datasets: Set5, Set14, Urban100, and DIV2K. As those sets have been used also during the development phase of most of the compared methods, the image quality results on those sets might be biased. Thus a fifth dataset, TAMPERE17, was included. It has not been used by the authors of any chosen method, so the results on it are less biased.

Quantitative analysis was done using two image quality metrics, PSNR and SSIM. Both are de facto metrics used in SISR research, and they are typically calculated only on the luma channel of images transformed to YCbCr color space. Additionally, we calculated the metrics on the full RGB color space to observe if the relative performance of the methods is dependent on the color space. Thus four different metrics are calculated for each dataset, which makes a total of 20 test cases for each scaling factor.

Results show that CNN based SISR methods offer unparalleled performance in comparison

to other methods, on both PSNR and SSIM metrics. On scaling factor of 4, EDSR and D-DBPN share the overall first place, with only neglible difference between them. EDSR scores highest on 12 out of 20 cases, with D-DBPN taking the first place on rest. On the scaling factor of 2, the number of first places is split evenly between the two methods.

There is a clear margin to the next best methods, VDSR and WSD-SR. Their results are quite close to each other, but clearly below EDSR and D-DBPN. The difference in PSNR_Y between them and the top-performers is typically around 1 dB, which is quite notable and the difference is clearly visible in the resulting images. VDSR outperforms WSD-SR by a small margin in most of the test cases. WSD-SR scores higher in 6 out of 20 cases on scaling factor of 4, and 3 out of 20 on other scaling factors.

There is once again a clear drop in all metrics, when moving to the next methods, A+ and SelfExSR. They are both from 2015, and have been included to highlight how rapidly the field has progressed in recent years. They are both around 1.5 dB below EDSR and D-DBPN, with SelfExSR scoring a little higher than A+ on all test cases.

ENet-PAT was the only method optimized for perceptual image quality, instead of minimizing MSE based reconstruction error. This is clearly visible in the quantitative results, and ENet-PAT is below even bicubic interpolation on 12 out of 20 cases. ENet-PAT tries to generate perceptually pleasing and plausible details, which comes with the cost of very low performance on quantitative metrics. ENet-PAT does succeed at its goal on few cases, by producing images almost indistinguishable from the ground truth images. Nevertheless, in most cases it fails and produces very distinctive and unplausible artifacts.

ENet-PAT highlights a very notable issue in the other methods. Optimizing for low reconstruction error leads to very good results on images with sharp edges and relatively flat regions, but on highly textured images they produce overly smooth results that look very unnatural. This issue has been noted also by other researchers in the field, and the first SISR competition concentrating on perceptual quality was organized recently [3]. The competion organizers noted that none of the commonly used full-reference image quality metrics are able to properly estimate the mean-opinion-scores, and many of them (including PSNR and SSIM) are actually anti-correlated with perceptual quality. No-reference metrics had a better correlation with the mean-opinion-score, but even the best of them were inadequate for proper perceptual quality assessment.

Since most of the methods in this comparison were optimized for MSE, the PSNR and SSIM values seemed to correlate well with the perceived quality. PSNR and SSIM correlated quite strongly with each other, with both RGB and Y color spaces. There were a few cases where the relative ordering of the methods changed with different metrics, but in those cases the differences were minor. There seems to be no extra information to be gained from calculating PSNR and SSIM in the RGB color space. Also usage of both PSNR and SSIM seems questionable in this context, as they both favor similar methods.

The numerical results are in line with other literature [53, 21], and they highlight another

trend in SISR research. The performance of MSE-optimized SISR seems to have reached a saturation point recently, as D-DBPN, the winner of NTIRE2018 [56] competition, performs almost identically to the previous year's winner EDSR [53]. Furthermore, the NTIRE2018 competition actually had the first place shared by 4 separate teams, as their PSNR scores were within 0.05 dB from each other.

Most likely in the near future the SISR research focus will shift towards perceptually optimized methods and even higher scaling factors, as the practical limits are approached in MSE-optimized SISR and scaling factors of 4 or less. Nevertheless, the current state-of-the-art still leaves room for improvement, especially in the computational complexity.

Tests were done using a very high-end GPU, and still D-DBPN took almost 50 seconds process an average image from the DIV2K dataset on scaling factor of 2. The input images were around 1000 pixels wide in that case, which is relatively low resolution by modern standards. A typical cellphone camera produces much larger images, and has a lot less processing power available. Thus algorithms like D-DBPN would be out of the question in mobile applications, or in any similarly powered embedded systems.

CNN-based methods present the absolute state-of-the-art in SISR, both in image quality and computational complexity, although there are still applications in which the self-similarity based approach might be more suitable. The two tested self-similarity methods were magnitudes slower, but they do have the advantage of not requiring any external data or prior training. In addition, CNNs have a larger tendency to hallucinate completely erroneous details, that might be critical in some applications. The more traditional machine learning methods share the same disadvantages as CNNs, but have lower image quality. Earlier methods like A+ had an advantage on computational complexity, but that gap has been closed by modern GPUs.

It is worth noting that all comparisons were done in an artificial setting, as the LR input images were downsampled with bicubic from the ground truth images without any noise or other degradations. In real world scenarios the mapping from HR to LR is usually unknown, and the LR images have noise, blur, and compression artifacts. None of the tested methods take these issues into consideration, and they would most likely perform sub-optimally with real world data [53, 56]. An interesting aspect for further research would be the incorporation of perceptual quality optimization with input data degraded by unknown noise, blur, and compression artifacts. This would correspond well to typical consumer level applications for SISR, like video upscaling, and photo enhancement. In those use-cases perceptual quality is prioritized over reconstruction accuracy, and they would be highly suitable applications for CNN based SISR.

REFERENCES

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

[2] M. Bevilacqua, A. Roumy, C. Guillemot, M.l.A. Morel, Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding, in: Proceedings of the British Machine Vision Conference 2012, 2012, British Machine Vision Association, pp. 1–135.

[3] Y. Blau, R. Mechrez, R. Timofte, T. Michaeli, L. Zelnik-Manor, 2018 PIRM Challenge on Perceptual Image Super-resolution, in: ECCV, 2018.

[4] C. Cruz, R. Mehta, V. Katkovnik, K.O. Egiazarian, Single Image Super-Resolution Based on Wiener Filter in Similarity Domain, IEEE Transactions on Image Processing, Vol. 27, Iss. 3, Mar. 2018, pp. 1376–1389.

[5] K. Dabov, A. Foi, V. Katkovnik, K. Egiazarian, Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering, IEEE Transactions on Image Processing, Vol. 16, Iss. 8, Aug. 2007, pp. 2080–2095.

[6] C. Dong, C.C. Loy, K. He, X. Tang, Learning a Deep Convolutional Network for Image Super-Resolution, in: ECCV, 2014, pp. 184–199. Available: http://link.springer.com/10.1007/978-3-319-10593-2_13

[7] C. Dong, C.C. Loy, X. Tang, Accelerating the Super-Resolution Convolutional Neural Network, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.), ECCV, Cham, Aug. 2016, Springer International Publishing, Lecture Notes in Computer Science, p. 17.

[8] J. Duchi, E. Hazan, Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, Journal of Machine Learning Research, Vol. 12, 2011, pp. 2121–2159.

[9] M. Ebrahimi, E.R. Vrscay, Solving the Inverse Problem of Image Zooming Using "Self-Examples", in: Image Analysis and Recognition, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 117–130. Available: http://dx.doi.org/10.1007/978-3-540-74260-9_11

[10] K. Egiazarian, A. Foi, V. Katkovnik, Compressed Sensing Image Reconstruction Via Recursive Spatially Adaptive Filtering, in: 2007 IEEE International Conference on Image Processing, Sept. 2007, IEEE, pp. I – 549–I – 552.

[11] K. Egiazarian, V. Katkovnik, Single image super-resolution via BM3D sparse coding, in: 2015 23rd European Signal Processing Conference (EUSIPCO), Aug. 2015, IEEE, pp. 2849–2853.

[12] A. Eskicioglu, P. Fisher, Image quality measures and their performance, IEEE Transactions on Communications, Vol. 43, Iss. 12, 1995, pp. 2959–2965.

[13] G. Freedman, R. Fattal, Image and video upscaling from local self-examples, ACM Transactions on Graphics, Vol. 30, Iss. 2, Apr. 2011, pp. 1–11.

[14] L. Gatys, A.S. Ecker, M. Bethge, Texture Synthesis Using Convolutional Neural Networks, in: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 262–270. Available: http://papers.nips.cc/paper/5633-texture-synthesis-using-convolutional-neural-networks

[15] L.A. Gatys, A.S. Ecker, M. Bethge, A Neural Algorithm of Artistic Style, CoRR, Vol. abs/1508.0, Aug. 2015.

[16] R. Gerchberg, Super-resolution through Error Energy Reduction, Optica Acta: International Journal of Optics, Vol. 21, Iss. 9, Sept. 1974, pp. 709–720.

[17] D. Glasner, S. Bagon, M. Irani, Super-resolution from a single image, in: 2009 IEEE 12th International Conference on Computer Vision, Sept. 2009, IEEE, pp. 349–356.

[18] R.C. Gonzalez, R.E. Woods, Digital image processing, 3rd ed., Prentice Hall, Upper Saddle River, NJ, USA, 2008. Available: https://dl.acm.org/citation.cfm?id=1076432

[19] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[20] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, Cambridge, MA, USA, 2014, MIT Press, NIPS'14, pp. 2672–2680.

[21] M. Haris, G. Shakhnarovich, N. Ukita, Deep Back-Projection Networks for Super-Resolution, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[22] J.L. Harris, Diffraction and Resolving Power*, Journal of the Optical Society of America, Vol. 54, Iss. 7, July 1964, p. 931.

[23] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, IEEE, pp. 770–778.

[24] A. Horé, D. Ziou, Image Quality Metrics: PSNR vs. SSIM, in: 2010 20th International Conference on Pattern Recognition, Aug. 2010, IEEE, pp. 2366–2369.

[25] G. Huang, Z. Liu, L.v.d. Maaten, K.Q. Weinberger, Densely Connected Convolutional Networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017, IEEE, pp. 2261–2269.

[26] J.B. Huang, A. Singh, N. Ahuja, Single image super-resolution from transformed self-exemplars, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, IEEE, pp. 5197–5206.

[27] M. Irani, S. Peleg, Improving resolution by image registration, CVGIP: Graphical Models and Image Processing, Vol. 53, Iss. 3, May 1991, pp. 231–239.

[28] J. Johnson, A. Alahi, L. Fei-Fei, Perceptual Losses for Real-Time Style Transfer and Super-Resolution, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.), Computer Vision – ECCV 2016, Springer International Publishing, Cham, Mar. 2016, pp. 694–711. Available: http://link.springer.com/10.1007/978-3-319-46475-6_43

[29] R.G. Keys, Cubic Convolution Interpolation for Digital Image Processing, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 29, Iss. 6, Dec. 1981, pp. 1153–1160.

[30] J. Kim, J.K. Lee, K.M. Lee, Accurate Image Super-Resolution Using Very Deep Convolutional Networks, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 38, Iss. 2, Nov. 2015, pp. 295–307.

[31] J. Kim, J.K. Lee, K.M. Lee, Deeply-Recursive Convolutional Network for Image Super-Resolution, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, IEEE, pp. 1637–1645.

[32] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, arXiv, Iss. cs.LG/1412.6980, 2014.

[33] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, USA, 2012, Curran Associates Inc., NIPS'12, pp. 1097–1105.

[34] Y. LeCun, Generalization and network design strategies, in: Pfeifer, R., Schreter, Z., Fogelman, F., Steels, L. (eds.), Connectionism in perspective, Elsevier, 1989.

[35] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, W. Shi, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, arXiv, Iss. 1609.04802, Sept. 2016.

[36] B. Lim, S. Son, H. Kim, S. Nah, K.M. Lee, Enhanced Deep Residual Networks for Single Image Super-Resolution, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), July 2017, IEEE, pp. 1132–1140.

[37] W. Lin, C.C. Jay Kuo, Perceptual visual quality metrics: A survey, Journal of Visual Communication and Image Representation, Vol. 22, Iss. 4, May 2011, pp. 297–312.

[38] MATLAB images.internal.resize.contributions source code, 2017.

[39] MATLAB images.internal.resize.cubic source code, 2017.

[40] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[41] K. Nasrollahi, T.B. Moeslund, Super-resolution: a comprehensive survey, Machine Vision and Applications, Vol. 25, Iss. 6, Aug. 2014, pp. 1423–1468.

[42] OpenCV resize source code, 2018. Available: https://raw.githubusercontent.com/opencv/opencv/3.4.2/modules/imgproc/src/resize.cpp

[43] S. Peleg, D. Keren, L. Schweitzer, Improving image resolution using subpixel motion, Pattern Recognition Letters, Vol. 5, Iss. 3, Mar. 1987, pp. 223–226.

[44] M. Ponomarenko, N. Gapon, V. Voronin, K. Egiazarian, Blind estimation of white Gaussian noise variance in highly textured images, Electronic Imaging, Vol. 2018, Iss. 13, Jan. 2018, pp. 382–1.

[45] N. Ponomarenko, L. Jin, O. Ieremeiev, V. Lukin, K. Egiazarian, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti, C.C.C. Jay Kuo, Image database TID2013: Peculiarities, results and perspectives, Signal Processing: Image Communication, Vol. 30, Jan. 2015, pp. 57–77.

[46] Y. Romano, J. Isidoro, P. Milanfar, RAISR: Rapid and Accurate Image Super Resolution, IEEE Transactions on Computational Imaging, Vol. 3, Iss. 1, Mar. 2017, pp. 110–125.

[47] F. Rosenblatt, The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain, PSYCHOLOGICAL REVIEW, 1958, pp. 65–386.

[48] J.C. Russ, F.B. Neal, The Image Processing Handbook, Seventh Edition, 7th ed., CRC Press, Inc., Boca Raton, FL, USA, 2016.

[49] M.S.M. Sajjadi, B. Schölkopf, M. Hirsch, EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis, in: 2017 IEEE International Conference on Computer Vision (ICCV), Dec. 2016, pp. 4501–4510.

[50] W. Shi, J. Caballero, F. Huszar, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, Z. Wang, Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, IEEE, pp. 1874–1883.

[51] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, Z. Wang, Is the deconvolution layer the same as a convolutional layer?, arXiv, Iss. 1609.07009, Sept. 2016.

[52] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv, Iss. 1409.1556, Sept. 2014.

[53] R. Timofte, E. Agustsson, L.V. Gool, M.H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K.M. Lee, X. Wang, Y. Tian, K. Yu, Y. Zhang, S. Wu, C. Dong, L. Lin, Y. Qiao, C.C. Loy, W. Bae, J. Yoo, , NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), July 2017, IEEE, pp. 1110–1121.

[54] R. Timofte, V. De, L.V. Gool, Anchored Neighborhood Regression for Fast Example-Based Super-Resolution, in: 2013 IEEE International Conference on Computer Vision, Dec. 2013, IEEE, pp. 1920–1927.

[55] R. Timofte, V. De Smet, L. Van Gool, A+: Adjusted anchored neighborhood regression for fast super-resolution, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9006, 2015, pp. 111–126.

[56] R. Timofte, S. Gu, J. Wu, L. Van Gool, NTIRE 2018 Challenge on Single Image Super-Resolution: Methods and Results, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018, pp. 852–863.

[57] R. Timofte, R. Rothe, L.V. Gool, Seven Ways to Improve Example-Based Single Image Super Resolution, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, IEEE, pp. 1865–1873.

[58] T. Tong, G. Li, X. Liu, Q. Gao, Image Super-Resolution Using Dense Skip Connections, in: 2017 IEEE International Conference on Computer Vision (ICCV), Oct. 2017, IEEE, pp. 4809–4817.

[59] R. Tsai, T. Huang, Multiframe image restoration and registration, in: Advances in Computer Vision and Image Processing, 1984.

[60] M. Vetterli, J. Kovacevic, V.K. Goyal, Foundations of Signal Processing, Cambridge University Press, 2014.

[61] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, C. Schroers, A Fully Progressive Approach to Single-Image Super-Resolution, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018.

[62] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image Quality Assessment: From Error Visibility to Structural Similarity, IEEE Transactions on Image Processing, Vol. 13, Iss. 4, Apr. 2004, pp. 600–612.

[63] R. Wen, K. Fu, H. Sun, X. Sun, L. Wang, Image Superresolution Using Densely Connected Residual Networks, IEEE Signal Processing Letters, Vol. 25, Iss. 10, Oct. 2018, pp. 1565–1569.

[64] C.Y. Yang, M.H. Yang, Fast Direct Super-Resolution by Simple Functions, in: 2013 IEEE International Conference on Computer Vision, Dec. 2013, IEEE, pp. 561–568.

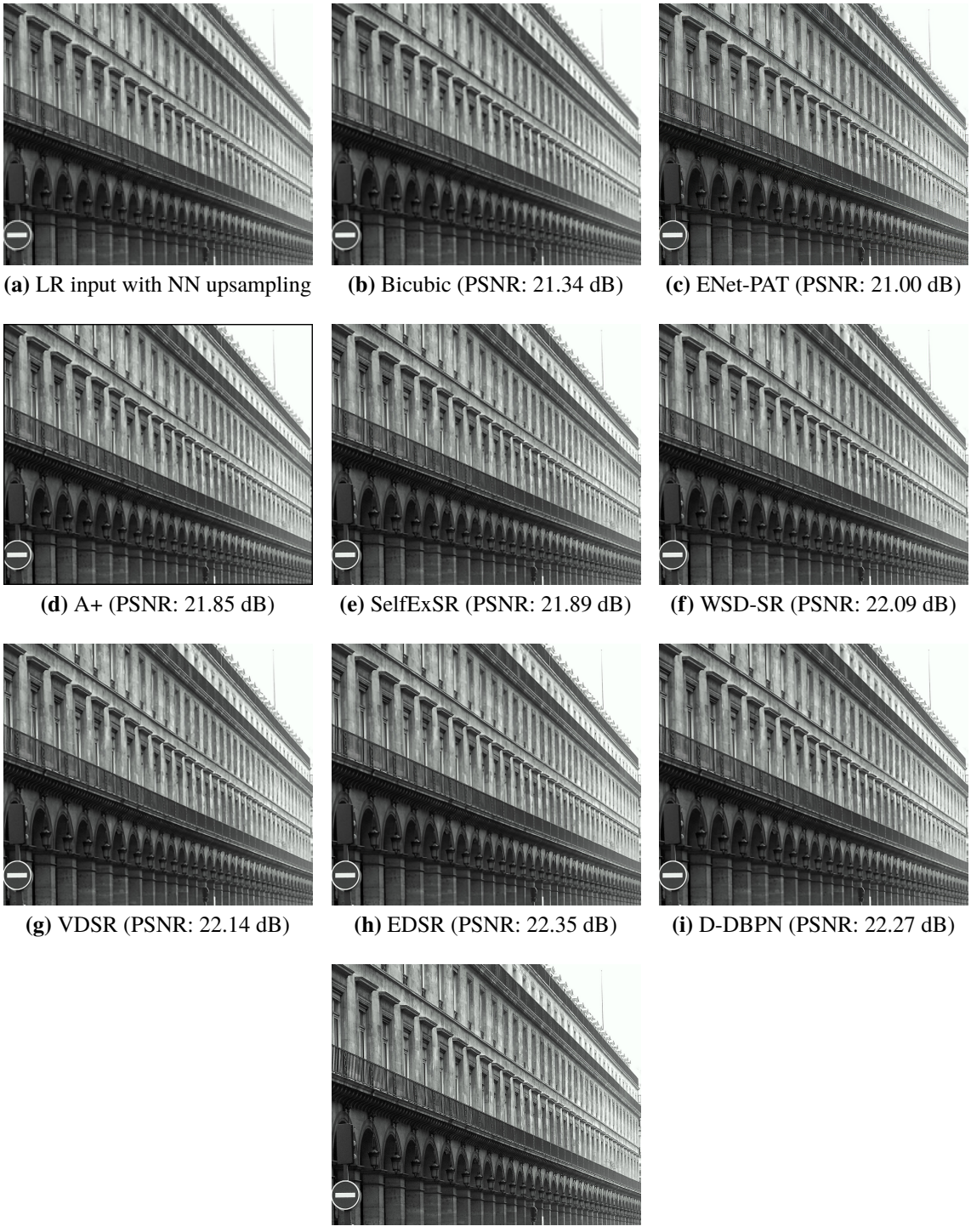
[65] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang, L. Zhang, Image super-resolution: The techniques, applications, and future, Signal Processing, Vol. 128, Nov. 2016, pp. 389–408.

[66] R. Zeyde, M. Elad, M. Protter, On Single Image Scale-Up Using Sparse-Representations, in: Boissonnat, J.D., Chenin, P., Cohen, A., Gout, C., Lyche, T., Mazure, M.L., Schumaker, L. (eds.), Curves and Surfaces, Berlin, Heidelberg, 2012, Springer Berlin Heidelberg, pp. 711–730.

[67] K. Zhang, W. Zuo, Y. Chen, D. Meng, L. Zhang, Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising, IEEE Transactions on Image Processing, Aug. 2017, pp. 1–1.

[68] Y. Zhao, R. Wang, W. Dong, W. Jia, J. Yang, X. Liu, W. Gao, GUN: Gradual Upsampling Network for single image super-resolution, arXiv, Iss. 1703.04244, Mar. 2017.

APPENDIX A: FULL SIZE OUTPUT IMAGES



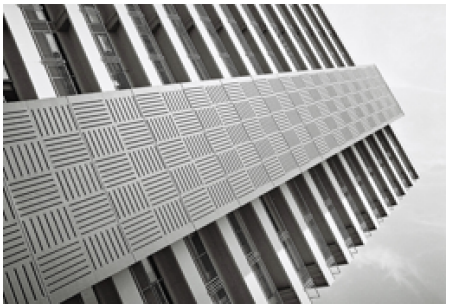
(a) LR input with NN upsampling **(b)** Bicubic (PSNR: 21.34 dB) **(c)** ENet-PAT (PSNR: 21.00 dB)

(d) A+ (PSNR: 21.85 dB) **(e)** SelfExSR (PSNR: 21.89 dB) **(f)** WSD-SR (PSNR: 22.09 dB)

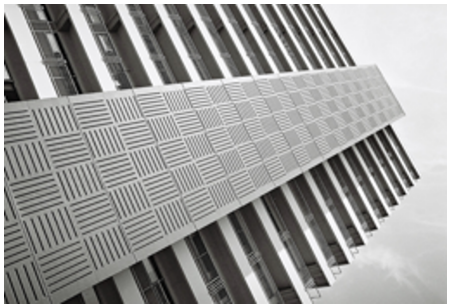
(g) VDSR (PSNR: 22.14 dB) **(h)** EDSR (PSNR: 22.35 dB) **(i)** D-DBPN (PSNR: 22.27 dB)

(j) Ground truth

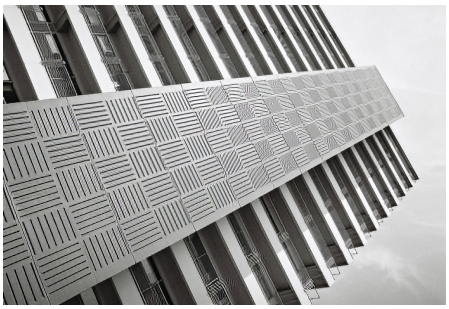
Figure A.1. *Image 66 from Urban100 dataset super-resolved with scaling factor of 4.*



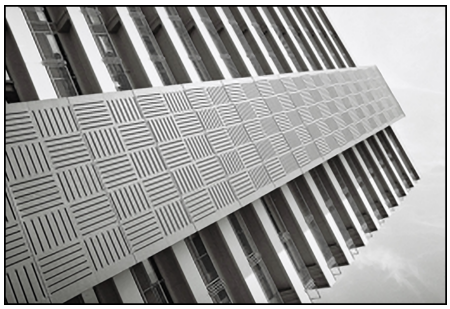
(a) LR input with NN upsampling



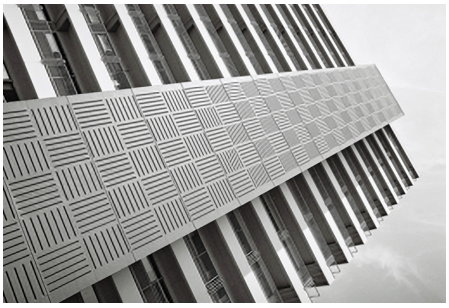
(b) Bicubic (PSNR: 16.58 dB)



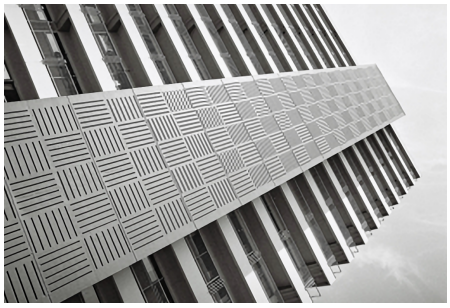
(c) ENet-PAT (PSNR: 17.46 dB)



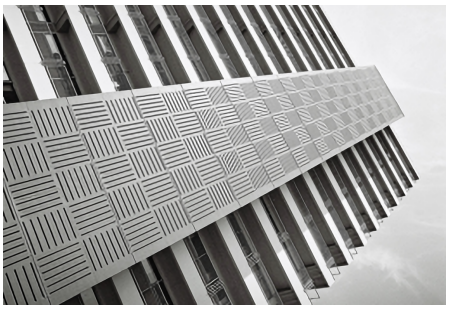
(d) A+ (PSNR: 17.33 dB)



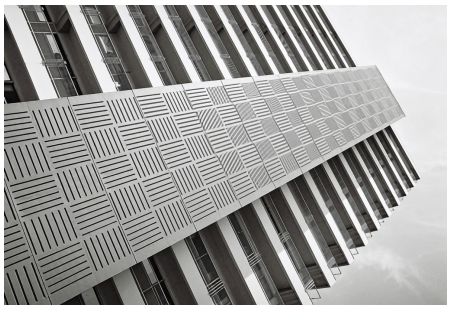
(e) SelfExSR (PSNR: 17.86 dB)



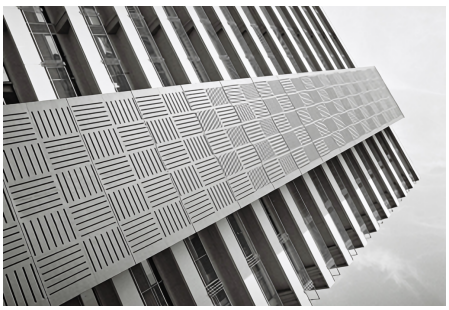
(f) WSD-SR (PSNR: 18.04 dB)



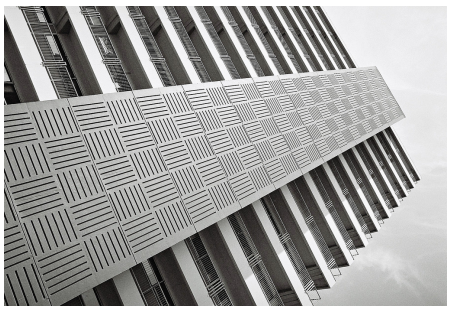
(g) VDSR (PSNR: 18.14 dB)



(h) EDSR (PSNR: 19.14 dB)



(i) D-DBPN (PSNR: 19.06 dB)



(j) Ground truth

Figure A.2. *Image 92 from Urban100 dataset super-resolved with scaling factor of 4.*

(a) LR input with NN upsampling

(b) Bicubic (PSNR: 28.23 dB)

(c) ENet-PAT (PSNR: 27.28 dB)

(d) A+ (PSNR: 29.09 dB)

(e) SelfExSR (PSNR: 29.05 dB)

(f) WSD-SR (PSNR: 29.23 dB)

(g) VDSR (PSNR: 29.36 dB)

(h) EDSR (PSNR: 29.94 dB)

(i) D-DBPN (PSNR: 29.94 dB)

(j) Ground truth

Figure A.3. *Image 858 from DIV2K dataset super-resolved with scaling factor of 4.*