Tampere University

German Felipe Torres Vanegas

A DEEP LEARNING FRAMEWORK FOR VIDEO TEMPORAL SUPER-RESOLUTION

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Joni Kämäräinen
Esin Guldogan
October 2020

ABSTRACT

German Felipe Torres Vanegas: A Deep Learning Framework for Video Temporal Super-Resolution
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
Major: Data Engineering and Machine Learning
October 2020

This thesis introduces a deep learning approach for the problem of video temporal super-resolution. Specifically, a network architecture and training schemes are proposed to produce an output video as it was captured using half the exposure time of the camera. By the recursive application of this model, the temporal resolution is further expanded by a factor of $4, 8, \ldots, 2^N$. The only assumption is made is that the input video has been recorded with a camera with the shutter fully open. In extensive experiments with real data, it is demonstrated that this methodology intrinsically handles the problem of joint deblurring and frame interpolation. Moreover, visual results show that the recursive mechanism makes frames sharper and sharper in every step. Nevertheless, it fails at generating temporally smooth videos.

Keywords: temporal super-resolution, exposure time, deblurring, deep learning, convolutional neural networks

PREFACE

This thesis was done under the supervision of Professor Joni Kämäräinen, head of Computer Vison group at Tampere University, and this work was also linked to a research project of Huawei Company in Tampere. I express my profound gratitude to Professor Joni Kämäräinen for letting me work as research assistant since a very early stage of my master degree and providing me great support and guidance. His enthusiasm and advice was a source of motivation to go forward in the project.

I would like to thank Esin Guldogan, Marti Ilmoniemi and Samu Koskinen from Huawei, who were always willing to provide comments about my work. Especially, I thank them for giving access to data that was utterly valuable for experimentation. I extend my gratitude to Professor Jiri Matas who generously shared his knowledge and brilliant ideas that undoubtedly nurtured this research topic. Besides, I would like to thank Dr. Said Pertuz for being my first mentor in academia and showing me his altruistic passion towards science and research, in general.

Last but not least, I would like to thank my family. To my parents, Anibal and Aminta, for being my support in life and spurring me to dream up the highest. To my brothers, Julian and Carlos, who are my confidants and primarily models of excellence.

Tampere, 12th October 2020

German Felipe Torres Vanegas

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

CNN	Convolutional Neural Network
DNN	Deep Neural Network
ISP	Image Signal Processing
LSI	Linear Spatially-Invariant
PSF	Point Spread Function
PSNR	Peak-Signal-to-Noise Ratio
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SSIM	Structural Similarity
STSR	Space-Time Super-Resolution
VFI	Video Frame Interpolation
VTSR	Video Temporal Super-Resolution

1 INTRODUCTION

Remarkable advances in video camera devices have been made during the last decades. For instance, Iphone X camera is capable of capturing HD-resolution videos at a speed up to 240 frames-per-second [1], which is 8 times faster, at much better resolution, than the best phone camera devices available 10 years ago [2]. However, high-quality devices are often more expensive in both computational resources and commercial cost. Therefore, there is still need for algorithms that produce very sharp and slow-motion frame sequences from low-frame-rate videos in cheaper devices. This might help as a video enhancement solution for end-user purposes or traffic surveillance, as some application examples. Even, this serves as a preprocessing step that improves the performance in higher computer vision tasks such as object detection [3] and object tracking [4]. The main challenge in this restoration task is the image blur. In practice, cameras require a finite exposure time to accumulate light from the scene, which turns into an averaging process and makes visual content less clear (blurry) in the presence of any movement. Figure 1.1 exemplifies the inherent blur under the photography process.

Image or video super-resolution primary refers as the process of recovering high-resolution spatial details from low resolution input image or video sequence [5]. Similarly, *Video Temporal Super-Resolution* (VTSR) is defined as the operation that estimates fast frame rate (short exposure) details from low frame rate (long exposure) frames. Accordingly, the end-result of VTSR is a video that is captured at higher frame rate along with less visible blur effect. The concept of temporal super-resolution was firstly coined by Shimano *et al.* [6], although there were already works tackling the problem of "space-time super resolu-

Figure 1.1. *Blurry picture from Sony dataset. Capturing a moving car generates blur.*

tion" [7, 8]. Notwithstanding, these approaches correspond to traditional reconstruction methods which are computationally expensive and require huge inference time.

In the recent years, deep learning frameworks have demonstrated astonishing performance in different image restoration tasks such as image deblurring [9, 10] and video interpolation [11, 12]. Results are incredible not only for the output image quality but also for its fast computation. Particularly, Kupyn *et al.* [10] proposed a neural network architecture that takes only 0.04 seconds in average to produce comparable results to state-of-the-art methods in image deblurring [9]. *Image deblurring* is the process of sharpening a blurry image whose blur can be caused by camera shake or moving objects within the scene during the exposure time. Nevertheless, this is a highly ill-posed problem as there are many sharp frames that can generate the given blurry image. *Video interpolation* aims at generating intermediate frames from sharp inputs, generally driven by optic flow [11]. The problem is that original frames are usually blurry in the presence of fast moving objects hindering the optic flow estimation, and subsequently the interpolation process. A naive solution might be to apply a deblurring stage before interpolation. However, this yields to sub-optimal solutions as the temporal information hidden in the blur has been removed. Instead of addressing deblurring and interpolation independently, they are both embedded to the established model of temporal super resolution, *i.e.* to reduce the blur shortcoming by increasing the time resolution (shortening the exposure time), jointly.

To the best of our knowledge, no deep-learning-based solution has been previously proposed for VTSR. Perhaps, the most similar work to ours is the one proposed by Jin *et al.* [13]. They proposed a two-network architecture for joint deblurring and video frame interpolation, increasing the time resolution 10 or 20 times. On one hand, a deblurring network is responsible of estimating some sharp key frames for the output video. On the other hand, an interpolation network computes the missing frames by using information of the blurry inputs and the sharp key frames. However, no VTSR modeling is done in this work, on the contrary, it uses a deblur-interpolation strategy.

The main goal of this thesis is to propose a neural network architecture for *Video Temporal Super-Resolution*. Specifically, the network takes two consecutive frames exposured for τ seconds and expands the time resolution as they were captured at $\tau/2$ seconds. By recursively applying this method, it can reach the point where everything becomes motionless. The only assumption we make here is that the camera has a shutter nearly always open, namely one frame period equals the exposure time. Extensive experiment with real data demonstrate the power of VTSR. In particular, we are able to restore static appearance of fast moving objects and deblur burst sequences, yielding to a joint solution for video interpolation and deblurring.

The reminder of this thesis is divided as follows. First, a literature review of traditional methods for VTSR and the deep learning background are presented in chapter 2. Then, the proposed deep-learning approach for VTSR and the performance quality metrics used for assessment are described in chapter 3. Furthermore, experimental settings and corresponding results are outlined in chapter 4. At last, chapter 5 draws the final conclusions.

2 BACKGROUND

2.1 Video Formation Model

First and foremost, it is imperative to establish how videos are produced by camera devices. Initially, let us consider how a single image is captured and then we extend this process for videos. A digital image is basically a multi-dimensional array that records the colorimetric information at discrete units called pixels. Usually, each pixel stores three color components: red, green and blue. Such cameras are referred as RGB devices. Internally, they are composed of a 2D grid of coupled devices that sense the incoming radiation [14]. The sensor response at each pixel $p = (x, y) \in \mathbb{R}^2$ is:

$$z(p) = \int_{\lambda \in \Lambda} E(p, \lambda) S_{k(p)}(\lambda) d\lambda \tag{2.1}$$

where $k(p) \in \{1, \ldots, m\}$ ($m = 3$ for RGB devices) denotes the color filter associated to the sensor p, $E(p, \lambda)$ is the input spectral radiance, $S_k(p)(\lambda)$ is the spectral sensitivity of the sensor, and Λ is the spectral domain in the range [400, 700] nm – this range corresponds to the portion that is visible to the human eye. Figure 2.1 depicts the color acquisition process, noting that raw measurements only captures one color component for each pixel. Particularly, it illustrates the case of a camera with a Bayer color filter array [15], which is broadly adopted in commercial RGB cameras. In addition, Figure 2.1(b) exemplifies the spectral sensitivities that can be found in RGB cameras.

In practice, sensors need to be exposed for certain time such that they capture enough

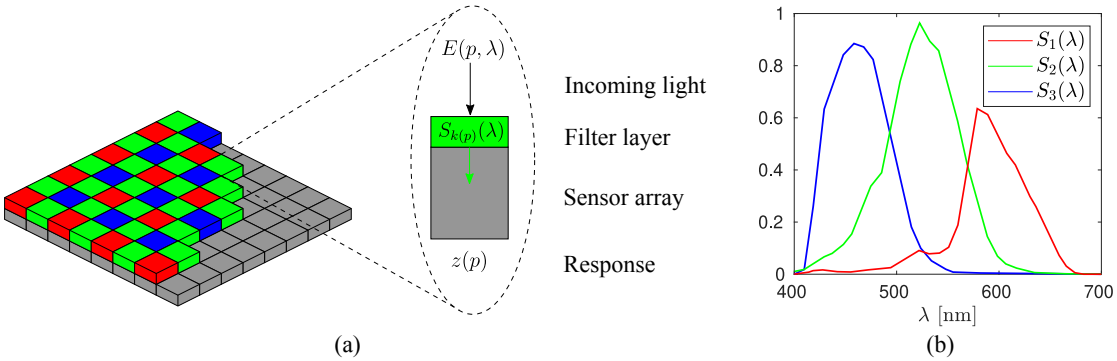


Figure 2.1. *Color acquisition in cameras. (a) Interaction of light with Bayer color filter array and grid of sensors, (b) Example of spectral sensitivity for RGB cameras.*

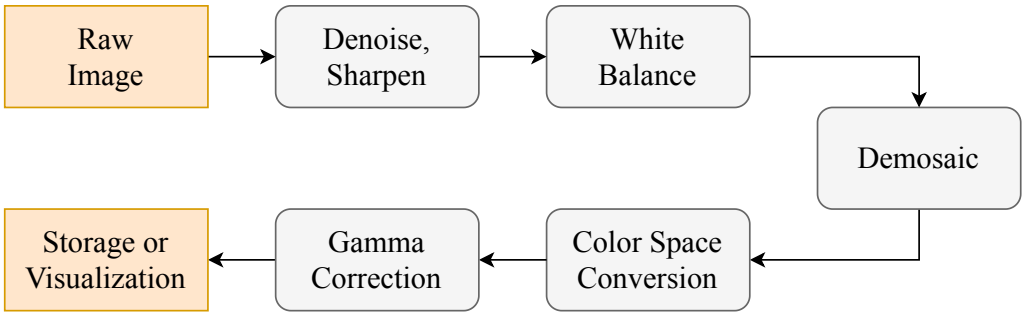


Figure 2.2. *Traditional ISP pipeline.*

light from the scene. Besides, they are sensible to noise sources. Therefore, a more accurate expression for the raw image is given by:

$$z_\tau(p) = \kappa \int_{t=0}^{\tau} z(p,t)dt + \eta(z(p,t)) \tag{2.2}$$

where $z(p,t)$ stands for the instantaneous sensor response, $\eta(z(p,t))$ denotes the signal-dependent noise component, κ is a scaling factor proportional to τ^{-1} [16], and τ is the exposure time. Since the scene and/or the camera are not static, we obtain blurry images of the scene. Intuitively, the longer is the exposure time, the more blur is visible in the final image.

Once the raw measurements are obtained, they go through an Image Signal Processing (ISP) pipeline, internally inside the camera, before the digital image is ready for viewing. Figure 2.2 illustrates a simplified block diagram of the main stages that take place inside a traditional ISP [17, 18]. First, some preprocessing tasks are executed to remove the noise and focus problems. Then, *White balance* aims at mapping "white" measurements to a true sensation of white, even when the light conditions change in the scene. Afterwards, all the m color components are estimated for each pixel through *demosaicing*. Recall that color filter arrays only allow to sense one color component per pixel. Subsequently, color space conversion is performed in order to match the human and camera spectral sensitivities. In other words, this makes pixel values to be seen as humans perceive colors. Finally, a gamma correction applies a nonlinear Camera Response Function (CRF) that maps the radiance to image intensities, namely the output digital values.

Regarding that the ISP involves many steps, it is quite convenient to focus on certain blocks for research purposes. Concretely in this thesis, we only deal with the blur that is consequence of the exposure time τ. Thus, by discarding the gamma correction block and assuming the other tasks of the ISP are performed in optimal conditions, our image formation model is simplified to:

$$z_\tau = \frac{1}{\tau} \int_{t=0}^{\tau} z(t)dt = z(t) * w_\tau(t) \tag{2.3}$$

where $z(t)$ is the instantaneous latent color image (as it has gone throughout the ISP), and $w_\tau(t)$ is equivalent temporal blur kernel, modeled as a rectangular window.

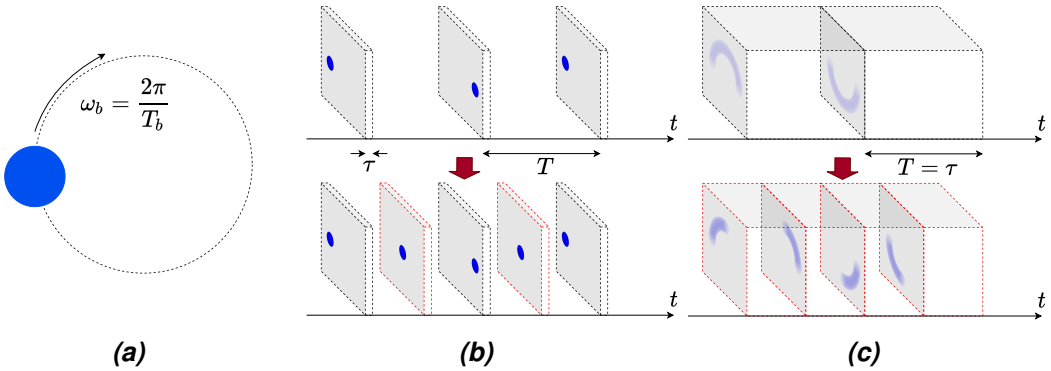


Figure 2.3. *VFI vs. VTSR. (a) Scene to be recorded (a rotating ball), (b) LTR video with sub-exposure time (upper), HTR video generated by VFI (lower), (c) LTR video with full-exposure time (upper), HTR video generated by VTSR (lower). Example replicated from [6].*

For the case of videos, we simply consider the sequential application of the previous image formation model at a given frame rate $r = 1/T$, being T the frame time. Then, we denote a discrete frame of a video as:

$$z[n]|_0^\tau = \frac{1}{\tau}\int_{t=nT}^{nT+\tau} z(t)dt \tag{2.4}$$

Particularly, when the shutter is fully open, then $\tau = T$.

2.2 Video Temporal Super-Resolution

Capturing videos under the presence of very fast dynamic objects is a challenging task as it may happen that they move faster than the actual frame rate. This causes some issues compromising the quality of videos such as jerkiness, motion blur and/or temporal aliasing. The presence of those shortcomings depends on the exposure time and the frame rate, which in turn, control the temporal resolution. *Video Temporal Super-Resolution* (VTSR) is the task that aims at recovering the rapid motion details which are not clearly seen in a recorded video sequence [6, 19]. In practice, VTSR estimates a set of high temporal resolution (HTR) frames from the captured low temporal resolution (LTR) frames.

Accordingly, VTSR is a mechanism to increase the frame-rate of the input sequence. Since *Video Frame Interpolation* (VFI) also aims at video frame rate up-conversion, some researchers refers to VTSR as VFI [20, 21]. However, despite having this common goal, there are methodological differences due to conditions in which the video is recorded. For the sake of clarity, let us consider the toy example illustrated in Figure 2.3 . We consider the process of capturing a video of a rapid ball following a circular trajectory with constant angular velocity $\omega_b = \frac{2\pi}{T_b}$, T_b being the period of the movement (Figure 2.3(a)). Assuming a low frame-rate camera with frame time $T = \frac{T_b}{2}$ and the capability to set the exposure time, there are two main settings in which the video can be shot: 1) *sub-exposure time*, where the exposure time τ is shorter than the frame time T; and 2) *full-exposure time*,

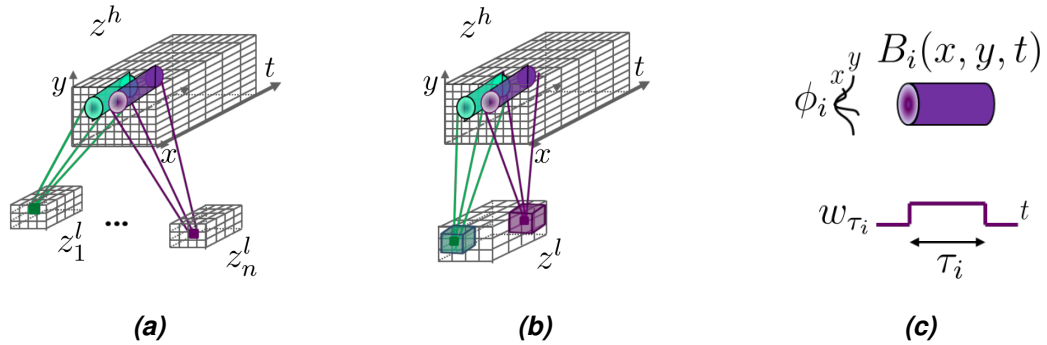


Figure 2.4. *STSR approaches. (a) Multi-video SR: measurements from multiple low-resolution videos impose linear constrains on the high-resolution video, (b) Single-video SR: similar patches within the low-resolution video can be interpreted as taken from different low-resolution videos, again inducing linear constrains on the high-resolution video, (c) The space-time blur kernel $B_i(x, y, t)$ is the composition of the spatial PSF ϕ_i and the temporal rectangular window w_{τ_i} with exposure time τ_i.* ***Source: [19]***

establishing $\tau = T$. In the first case, discontinuous motion is observed since the camera shutter remains open at short intervals (upper part of Figure 2.3(b)), loosing details of the real ball trajectory. To improve the temporal resolution, intermediate frames can be computed by using VFI methods, as illustrated in the red-colour frames in the lower part of Figure 2.3(b). Nevertheless, the true movement is not successfully recovered, in this specific example, because the time sampling rate is very low and VFI methods typically assume linear displacement for the interpolation. In the second case, the right trajectory of the ball is implicitly obtained, paying the price of motion blur in each frame (upper part of Figure 2.3(c)). Under this scenario, VFI methods are not applicable as they do not allow to resolve frame into two frames. Additionally, the motion blur makes difficult to establish the dense correspondence for the motion estimation, necessary in VFI approaches. On the contrary, VTSR solutions are suitable for full-exposed videos since they estimate a high-resolution video as if it was captured using shorter exposure time. Consequently, the temporal resolution is expanded.

2.2.1 Previous work

Multi-video spatio-temporal super-resolution

Initially, the VTSR was partially tackled in the more general context of Space-Time Super-Resolution (STSR), where the resolution is increased in, both, temporal and spatial domain [7, 8]. Shechtman *et al.* [7] proposed a method for producing a high space-time resolution video z^h from a set of low-resolution video sequences $\{z_i^l\}_{i=1}^N$ recording the same dynamic scene (Figure 2.4(a)). Regarding a set of space-time transformations $\mathcal{T}_i$ that aligns the coordinate system between z^h and z_i^l for $i = 1, \ldots, N$, then every space-time pixel $p = (x, y, t)$ in the high-resolution video is projected to $p_i^l = \mathcal{T}_i(p)$, p_i^l being pixels into the ith low-resolution video. More precisely, the relation between the measurements

$z_i^l(p_i^l)$ and $z^h(p)$ is given by the video observation model:

$$\begin{aligned} z_i^l(p_i^l) &= (z^h * B_i)(p) \\ &= \int_{q=(x,y,t)\in \mathrm{supp}(B_i)} z^h(q) B_i(q-p) dq \end{aligned} \tag{2.5}$$

where $B_i = \phi_i * w_{\tau_i}$ is the space-time blur operator composed by the Point Spread Function (PSF) ϕ_i and temporal kernel blur w_{τ_i} of the ith camera (Figure 2.4(c)). By stacking those relations for every measurement in discrete form, a linear system of equations in terms of the unknown high resolution elements of z^h can be constructed:

$$A\mathbf{h} = \mathbf{l} \tag{2.6}$$

where $\mathbf{h}$ is a column vector containing all the unknown elements of the high-resolution video z^h, $\mathbf{l}$ is the column vector with the measurements taken from all the low-resolution sequences $\{z_i^l\}_{i=1}^N$, and A denotes the matrix with the relative contributions of the unknown elements to each low-resolution measurements defined by equation 2.5.

Naturally, the size of z^h is bigger than the size of a single sequence z_i^l, but when there is access to an enough number of low-resolution videos, there are more equations than unknowns in equation 2.6. Hence, a least-square solution can be computed for the linear system. Nonetheless, Shechtman *et al.* additionally added a regularization term for numerical stability and smoothness purposes, such that their STSR solution is given by:

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}} \|A\mathbf{h} - \mathbf{l}\|^2 + \lambda_s R_s(\mathbf{h}) + \lambda_t R_t(\mathbf{h}) \tag{2.7}$$

where $R_s(\cdot)$ and $R_t(\cdot)$ are the regularization functions in spatial and temporal domain, respectively, while λ_s and λ_t are their corresponding weights. In particular, Shechtman *et al.* used directional regularizers that smooth the values along the space-time edges.

Alternatively, Mudenagudi *et al.* [8] extended the aforementioned work by adding non-linear constraints, allowing them to achieve higher magnification factors. By formulating the STSR reconstruction problem using the Maximum a posteriori-Markov Random Field, they found a resembling optimization problem:

$$\min_{z^h} \sum_{p\in\Omega} \sum_{i=1}^{N} \alpha_i(p, p_i^l) \left[(z^h * B_i)(p) - z_i^l(p_i^l)\right]^2 + \lambda_s R_s(z^h) + \lambda_t R_t(z^h) \tag{2.8}$$

where Ω is the set of space-time pixels in the high-resolution video and $\alpha_i(p, p_i^l)$ denotes the non-linear constraints that selectively determine whether a low-resolution pixel p_i^l contributes to the reconstruction of the pixel p in the high-resolution video. In such work, truncated linear functions are considered for the regularizers $R_s(\cdot)$ and $R_t(\cdot)$. Moreover, they used graph-cut optimization to find the final solution.

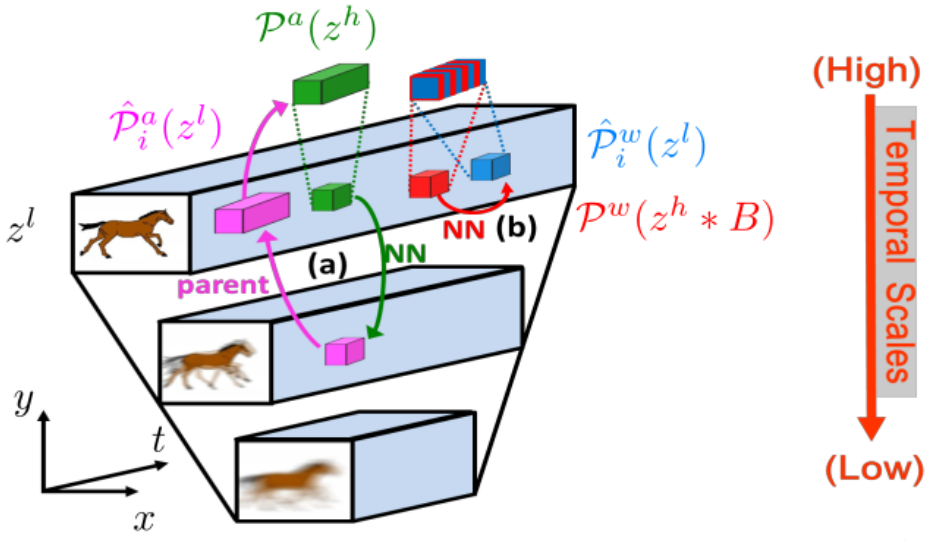


Figure 2.5. *Self-similarity within and across temporal scales. (a) across-scale similar patches provide "Example-based" constraints, i.e., $\mathcal{P}^a(z^h)$ might look like $\hat{\mathcal{P}}_i^a(z^l)$, (b) within-scale similar patches impose "Classical" constraints as additional linear constraints can be added: $\mathcal{P}^w(z^h * B) = \hat{\mathcal{P}}_i^w(z^l)$.* ***Source:****[19]*

Single-video spatio-temporal super-resolution

In the case of a single low-resolution video z^l, our video observation model in equation 2.5 reduces to $z^l(\mathcal{T}(p)) = (z^h * B)(p)$, where $\mathcal{T}$ stands for the space-time decimation operator. As a result, the construction of the linear system in equation 2.6 remains undetermined due to the higher number of elements in z^h. Notwithstanding, *self-similarity* can be exploited to add more constraints. The idea is inspired by the pioneer work of Glasner *et al.* [22] which shows that small patches in a natural image tend to recur many times inside the image, within and across multiple scales. This means that we can consider similar patches as if they were extracted from the same high-resolution patch, which leads to multiple constraints on the unknown elements of z_h (Figure 2.4 (b)).

To be precise, self-similar patches can induce two types of constraints depending on where the similar patches are taken from, as illustrated in Figure 2.5. Recurrence of small patches *across* coarser spatio-temporal scales introduces *"Example-based"* constraints since it provides some "guesses" for the high-resolution video. The principle is illustrated in Figure 2.5(a). Let us assume a reference patch in z^l (small green) "recurs" in a coarser scale (small pink). Thereby, the parent of the similar patch $\hat{\mathcal{P}}_i^a(z^l)$ (large pink) serves as an estimation of how the slow-motion version of the reference $\mathcal{P}^a(z^h)$ (large green) might look like. On the other hand, recurrence of small patch *within* the same video scale induces *"Classical"* constraints since they can be considered as if they were captured by different cameras. In Figure 2.5(b), a reference patch in the low-resolution video $\mathcal{P}^w(z^l)$ (small red) has a similar patch within the same scale $\hat{\mathcal{P}}_i^w(z^l)$ (small blue). Taking advantage of this similarity, we can introduce the constraint $\mathcal{P}^w(z^h * B) = \hat{\mathcal{P}}_i^w(z^l)$. Overall, the optimization problem that includes the *self-similarity* priors can be written as:

$$\min_{z^h} \sum_{p\in\Omega} \left[(z^h * B)(p) - z^l(p^l)\right]^2 + \lambda_a \sum_{\mathcal{P}^a} \sum_{i=1}^{M} \|\mathcal{P}^a(z^h) - \hat{\mathcal{P}}_i^a(z^l)\|^2 + \lambda_w \sum_{\mathcal{P}^w} \sum_{i=1}^{N} \|\mathcal{P}^w(z^h * B) - \hat{\mathcal{P}}_i^w(z^l)\|^2 \tag{2.9}$$

whose second and third term respectively refer to the *"Example-based"* and *"Classical"* constraints, $\{\hat{\mathcal{P}}_i^a\}_{i=1}^M$ is the set of across-scale similar patches for each reference patch $\mathcal{P}^a$, $\{\hat{\mathcal{P}}_i^w\}_{i=1}^N$ is the set of within-scale similar patches for each reference patch $\mathcal{P}^w$, and λ_a, λ_w are the weighting parameters.

In this way, Shimano *et al.* [6] proposed a VTSR method from a single video by incorporating the *"Example-based"* constraints from self-similar image patches across spatio-temporal scales. In addition, a smoothness term $R_t(z_h)$, based on the Laplacian filter, was included to avoid flickering effects, similarly as in equations 2.7, 2.8. As opposed to 2D image patches, Shahar *et al.* [19] extended the idea to 3D ST-patches along with the use of the *"Classical"* constraints for the general case of STSR. Notably, they proposed an efficient way to find similar ST-patches at sub-frame accuracy. Their visual results reveal the capability to resolve severe motion aliasing and motion blur, especially for the case of VTSR. More recently, Maggioni and Dragotti [23] presented a two-stage approach for VTSR. Each stage starts by computing motion-compensated 3D patches, *i.e.*, a stack of 2D blocks following a motion trajectory. In the first stage, a set of similar 3D patches are matched to the references, registered at sub-pixel level, and aggregated at the pertinent location in the high-resolution video. In the second stage, registration artifacts are fixed by using an error-correcting linear operator, which is learned from self-similar patches across temporal scales.

It is noteworthy to remark that VTSR is reduced from STSR by considering that the temporal blur window only acts in the blur function, *i.e.*, $B(t) = w_\tau(t)$, and the transformation $\mathcal{T}$ that aligns the coordinate systems between the high and low resolution videos corresponds to the temporal decimation operator. That is why STSR methods are equally applicable to VTSR.

2.3 Deep learning

Deep learning is considered a subfield of machine learning that includes methods for the data-driven learning of a hierarchically organized representation [24]. Commonly, deep learning models are referred as Deep Neural Networks (DNNs) since their structure were originally inspired by how the exchange of information, that yields to learning, occurs among neurons inside the brain. To understand what DNNs really do, it is appropriated to recall *machine learning*. Traditional machine learning methods aim at approximating mapping rules through experience, which is achieved by providing data samples. Remarkably, the performance heavily depends on the representation (features) of data they are given. For instance, let us consider a machine learning system for face recognition. Human beings can easily recognize faces by their oval shape comprised of eyes, mouth

and hair. Nonetheless, it can be challenging to define a computerized set of features that matches with those high-level concepts, such that the machine learning algorithm can find a decision rule based on those input feature values. For many years, part of the research was dedicated to devise hand-crafted and application-oriented features that help machine learning algorithms to solve a particular task. In contrast, deep learning methods discover, both, the representation and the mapping rule from raw data by building complex concepts from simpler ones in a hierarchical way.

Although most of the principles and basic methods of deep learning were already seeded back in the 1980s [25, 26, 27], it was not until after 2012 when they became popular and successful in different computer-aided applications [28, 29, 30]. The increase in performance can be attributed to three aspects. First, the access to larger datasets allows DNNs to reach a generalized mapping rule at the end of the training phase [31]. Second, the possibility to implement bigger models as the computational resources have improved along the time. Nowadays, we rely on GPUs to train models that were excessively "deep" in size to be stored and trained with computers in the past. At last, few advances in regularization and optimization techniques enable the speed-up in convergence and to reach more optimal solutions [32, 33].

In the beginning of the deep-learning boom, several architectures, specifically Convolutional Neural Networks (CNNs) [27], were mainly used for image classification, wherein an image is taken as input and produces a binary vector associated with an image label [28, 34, 35]. Soon after, CNNs were extended to image-to-image problems, what means that the network is able to output an entire image. In particular, end-to-end CNN-based models have demonstrated to outperform traditional reconstruction algorithms for image restoration problems [36, 37]. Despite of the fact that there is not a strong mathematical proof of how DNNs are able to restore the image, some researchers have found relations between CNN-based models and traditional restoration algorithms. Notably, Jain and Sebastian [38] showed the connection between CNNs and Markov random field (MRF) methods in image denoising. Dong *et al.* [36] found that sparse-coding-based SR methods can be interpreted as a particular CNN. Zhang *et al.* [37] pointed out that their proposed CNN is a generalization of a one-stage trainable nonlinear reaction diffusion (TNRD) model for image denoising.

Certainly, end-to-end CNNs for image restoration represent basis for the deep-learning-based approach for VTSR that is presented in chapter 3. Thus, we introduce the reader to the relevant deep-learning background for image restoration in the reminder of this section. Concretely, the basic CNN architectures for image-to-image tasks and their main components are described in section 2.3.1. Then, the mathematical foundations of the learning process for DNNs are exposed in section 2.3.2.

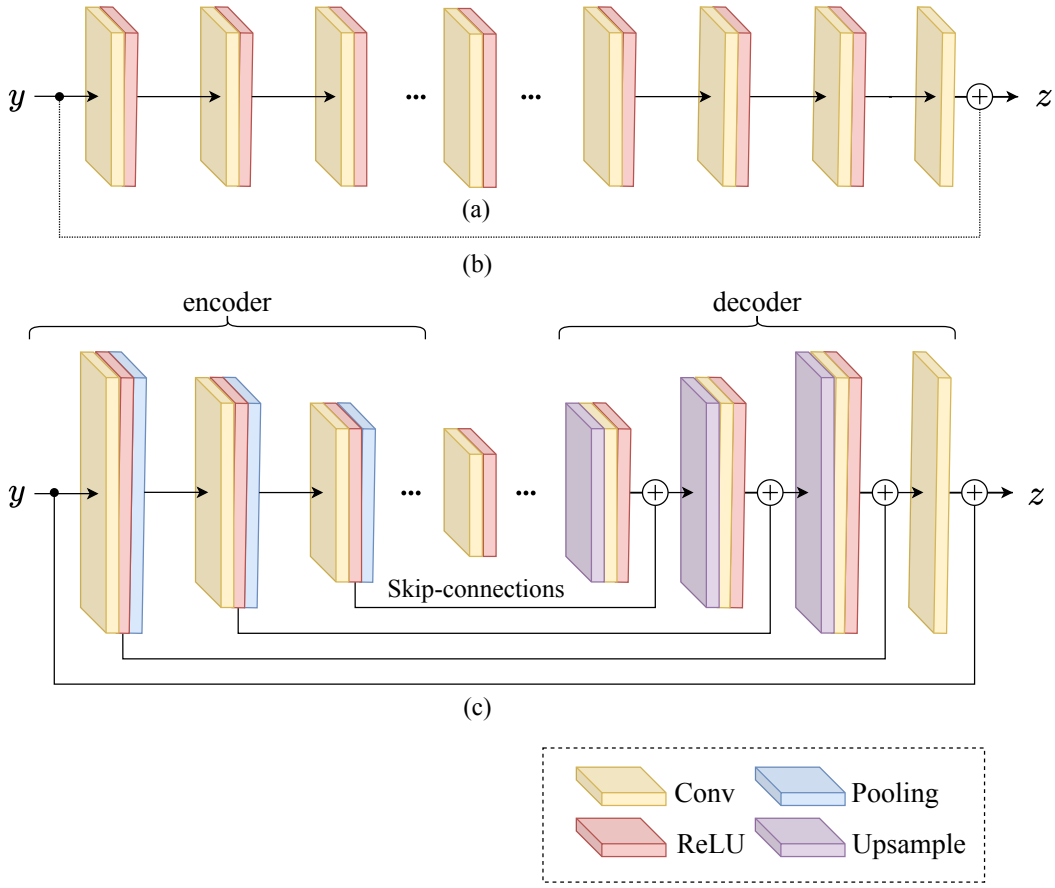


Figure 2.6. *Standard CNN architectures for image restoration. (a) Fully-convolutional architecture, (b) Fully convolutional architecture with residual connection, (c) Encoder-decoder architecture*

2.3.1 Standard CNNs for image restoration

The goal of any image restoration method is to recover a clean image z provided its corresponding observed image y, which is the result of a degradation function ϕ, *i.e.*, $\mathrm{y} = \phi(z)$. Generally, this is an ill-posed problem, so that it is not easy to define an inverse mapping ϕ^{-1} to restore the latent image. In a deep-learning framework, we use a large set of data samples $\{\mathrm{y}_i, z_i\}_{i=1}^{N}$ to optimize the parameters Θ of a DNN f such that it approximates to the inverse degradation function ϕ^{-1}:

$$z \approx f(\mathrm{y}; \Theta^*)$$

where Θ^* denotes the optimized parameters of the network that are achieved once the training process is executed. The way how the parameters Θ^* are computed is explained in section 2.3.2. Here, we merely focus on the basic architectures of f and their components.

Figure 2.6 illustrates three standard DNNs that can be used for this purpose. To be specific, they are CNNs as they include the convolution layer (Conv) as their main building block. The very basic architecture is only a finite cascading connection of a convolu-

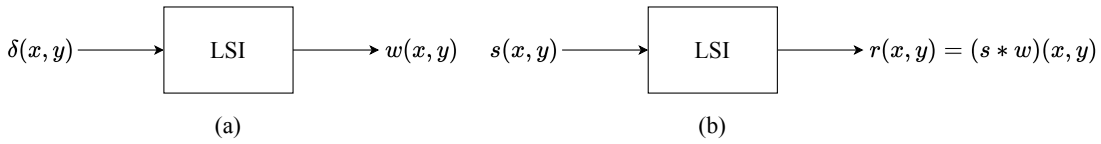


Figure 2.7. *Convolution in LSI systems. (a) Response to the unitary impulse signal δ, (b) Response to an arbitrary signal s*

tion layer (Conv) and a Rectified Linear Unit (ReLU), as shown in Figure 2.6(a). The intuition behind this architecture is that more complex representations are computed after each Conv+ReLU operation until the last Conv layer that takes a regression role to obtain the desired output. This architecture has been used, for example, by Dong *et al.* [36] for single-image SR. The second architecture has practically the same structure but it adds a residual connection to the input, *i.e.*, the output is the summation between the input and the result of the last Conv layer (Figure 2.6(b)). This strategy is advantageous when the input and the output are highly correlated since the convolutional block only has to estimate the residual image instead of the full clean target image. This architecture has demonstrated faster convergence and superior performance compared to the non-residual structure in single-image SR [39] and image denoising [37]. The third architecture is based on the popular U-net [40] that was initially proposed for image segmentation. This network (Figure 2.6(c)) is comprised of: (i) an encoder that extracts the primary elements of the image while removing corruptions, (ii) a decoder encharged of recovering image details from the encoded features, and (iii) skip connections that help the decoder to restore a cleaner image. This network architecture has been used in many image restoration methods [41, 42, 43] and other image-to-image tasks [44].

Convolution layer

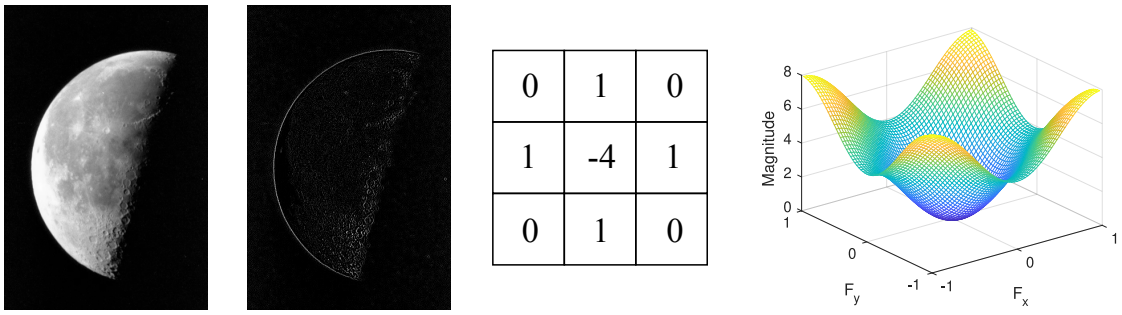
The convolution layer is the core of the representation extraction for DNNs in image-related applications. As its name suggests, this layer is based on the convolution operator. For 2D discrete signals, as in the case of images, the convolution is defined by:

$$r(x,y) = (s * w)(x,y) = \sum_m \sum_n s(m,n) w(x-m, y-n) \tag{2.10}$$

where $w(x,y)$ denotes the kernel filter, while $s(x,y)$ and $r(x,y)$ are the input and output signals, respectively.

From linear system theory, it is well known that a Linear and Spatially-Invariant (LSI) system can be characterized by its response w to the unitary impulse signal δ, as shown in Figure 2.7(a). Being δ defined as:

$$\delta(x,y) = \begin{cases} 1 & x,y = 0 \\ 0 & \text{otherwise} \end{cases}$$



(a) (b) (c) (d)

Figure 2.8. *Filtering illustration. (a) Input image of the moon, (b) Output obtained by convolution, (c) Laplacian kernel, (d) Frequency response of kernel in (c)*

Besides, the output r of the system, to any input s, can be computed by the convolution with its impulse response w using equation 2.10 (Figure 2.7(b)). This suggest that the convolution layer can be interpreted as a signal s that goes through a LSI system with response w. When the impulse response w is analyzed in frequency domain, it actually emphasizes some spatial frequency components. For instance, let us consider the laplacian kernel and its frequency response in Figure 2.8(c-d), respectively. We can notice that the magnitude of its frequency response keeps a high value for high frequencies ($|\mathsf{F_x}|, |\mathsf{F_y}| \approx 1$) and it progressively reduces when it gets closer to the origin. Thus, the net effect of this kernel in the output is to accentuate abrupt changes of intensity, as it occurs in the edges, while suppressing or filtering constant values (low frequencies). That is the reason why kernels are referred as *filters*, whereas the output is named *feature map* since it highlights certain information.

In fact, multiple feature maps are extracted at once in a convolutional layer. To be precise, the output signal r of size (C', H', W')[1] of a convolutional layer with an input s of size (C, H, W) is described by:

$$r(k) = b(k) + \sum_{l=1}^{C} w(k,l) \star s(l), \quad k = 1, \ldots, C' \tag{2.11}$$

where $\star$ denotes the valid 2D cross-correlation operator [2], b is bias vector of length C', w is the kernel of size (C', C, K, K), C denotes the number of channels (maps), and H, W denote the height and width of the respective discrete signals. Interestingly, w and b belong to the set of learnable parameters Θ of the network f. In other terms, the network learns itself a set of filters which extract features that contribute for a better reconstruction. Since convolution layers are placed on top of previously computed feature maps, more sophisticated and abstract features are extracted as we go through the deeper layers of the network [46].

[1] In the case of unitary *stride*, no *padding* nor *dilation*, the output dimensions are $H' = H - K + 1, W' = W - K + 1$. To keep the dimensions equal, one can add a total padding of $K - 1$ for each axis. For more complex cases, the reader is referred to [45].

[2] The cross-correlation operator is equivalent to the convolution with the only difference that it does not need the flipping operation and therefore is less computational expensive [24].

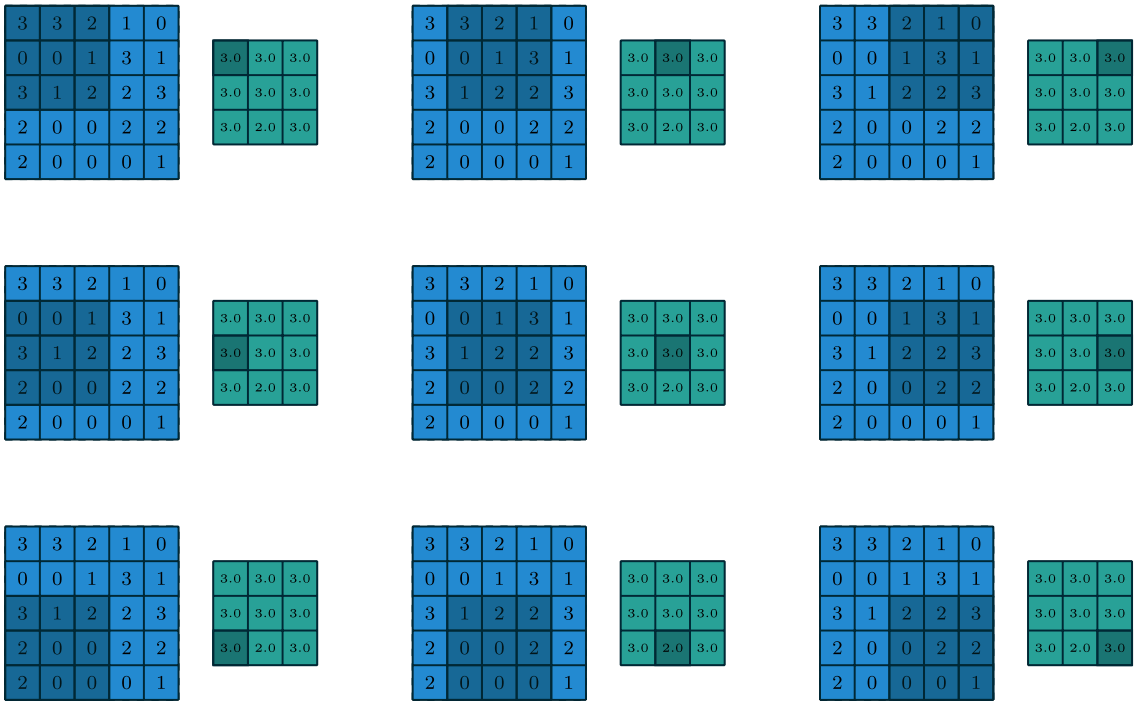


Figure 2.9. *Computation of a max-pooling operation. This example takes a patch size 3×3 and stride 1×1, wherein the largest value in the shaded blue region is copied to the highlighted location in the output green matrix.* ***Source:*** *[45]*

Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) is type of activation function that is used by default for the hidden layers of a DNN. Typically, an activation function is added on top of an affine operation (as is the case of a convolution layer) with the purpose of integrating a non-linear behaviour in the network. The ReLu is mathematically defined as:

$$g(u) = \max\{0, u\}$$

where $u \in \mathbb{R}$. In simple terms, the ReLu is linear except that it outputs zero whenever u is negative. It turns out that those small non-linearities are enough to produce complex non-linear mappings in the network when it consist of many hidden layers. Likewise, the ReLu become the default activation function for hidden layers in DNNs because it allows them to accomplish better convergence and avoid the so-called problem of vanishing gradients. In section 2.3.2, we go back to this issue, such that the reader understands the important role of the gradients in the learning process. At this point, it is sufficient to know that if the unit is active ($u > 0$), the gradients remain large and consistent.

Pooling layer

The pooling layer is used as a sub-sampling operator that incorporates a translation-invariant property. This implies that if the input is translated a small amount, the pooled features keep the same value [24]. To put it differently, the presence of a feature tends

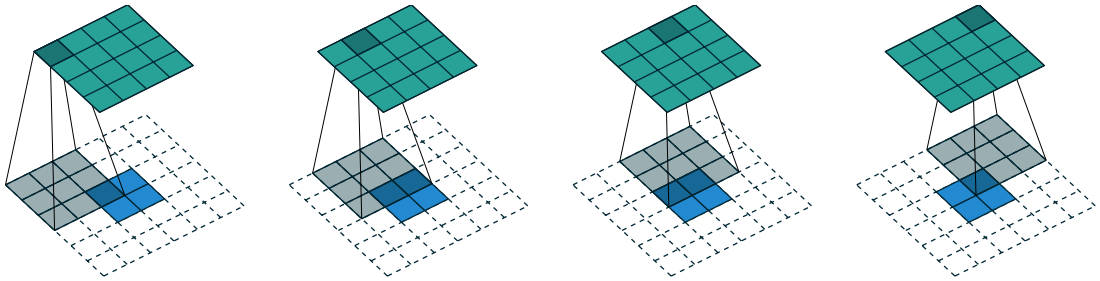


Figure 2.10. *Computation of transposed convolution. A 2×2 input (blue) is transposed-convolved with a 3×3 kernel (gray), which turns into a 4×4 output (green).* ***Source:*** *[45]*

to be more important than its specific location. Consequently, pooling layers helps to filter out noise and corruptions in the encoder, while maintaining meaningful and coarser features. Perhaps, the most widely used pooling operator in CNNs is the *max-pooling* as it has exhibited better performance [47]. Basically, it extracts the maximum value over a patch in the input feature map, similarly as shown in Figure 2.9. In this example, a patch of size 3×3 is moving along the input matrix with stride 1×1. In every moving step, the maximum value within the patch is concatenated into an output array in the same way the patch is shifted along the input.

Upsampling layer

Upsampling layers are essential in DNN architectures where feature maps have to be projected back to higher-dimensional spaces. Regarding the encoder block of the DNN presented in Figure 2.6(c), one needs to up-sample the abstract feature maps of the last encoded layer such that the dimensions of the input image and the network output match each other. Otherwise, an end-to-end image restoration DNN could not be implemented. For this purpose, several interpolation methods can be utilized such as nearest neighbor, bilinear, and bicubic.

Transposed convolution constitutes another upsampling method for decoders in image-to-image DNN [40, 45]. Its name comes from an analogy of matrix transposition. In fact, the convolution operation can be unrolled and expressed as a matrix multiplication. For instance, let us consider the valid convolution between a 4×4 array s and a 3×3 kernel w, whose result is a 2×2 array r. Equivalently, the convolution can be executed by the matrix multiplication of a 4×16 sparse matrix W with a 16-element column vector $\mathbf{s}$, which ends up with a 4-element column vector $\mathbf{r}$, *i.e.*, $\mathrm{W}\mathbf{s} = \mathbf{r}$. By applying the transposed matrix over the 4-element column vector $\mathbf{r}$, we have $\mathrm{W}^T\mathbf{r} = \widetilde{\mathbf{s}}$, being $\widetilde{\mathbf{s}}$ a 16-element column vector. Hence, W^T allows to project a feature map into a higher dimensional space. Notwithstanding, transposed convolution is not implemented as a matrix multiplication but in an algorithmic manner as exemplified in Figure 2.10. Particularly, this is equivalent to the convolution of a 3×3 kernel with of a 2×2 input padded with a 2×2 border of zeros using unitary strides. Similarly as in convolution layers, the kernel belongs to the set of parameters Θ of the network f, what makes transposed convolution a learnable mapping.

Skip connections

Skip connections play an important role for the successful training of the encoder-decoder network that fits the target restoration mapping. Firstly, pooling layers tend to remove too much details that complicates the recovery task of decoder. Nevertheless, passing-by the feature maps of the encoder towards the decoder makes to re-incorporate the missed feature details, and so, it helps the decoder to restore a cleaner image. Secondly, skip connections promote the convergence to a better solution in the optimization process. As hypothesized by He *et al.* [35], residual mappings, referenced to the input of previous stacked layers, are easier to learn than the whole mapping without reference. Under the hood, the gradients, required in the learning process, often vanishes for deep architectures. However, the skip connections automatically passes backwardly the gradients to bottom layers, preventing the vanishing gradient problem to happen. Again, we go back to this issue in section 2.3.2 once the back-propagation algorithm is presented.

2.3.2 Learning process

The learning stage concerns the methods for training a DNN f. In essence, we need to adjust the set of parameters Θ of f that makes the DNN to produce the target mapping. For this purpose, a cost function $J^*(\Theta)$ is primarily specified – generally as a minimization cost. Then, we use an optimization algorithm to minimize:

$$J^*(\Theta) = \mathbb{E}_{p(\mathrm{y},z)}[\mathcal{L}(f(\mathrm{y};\Theta), z)] \tag{2.12}$$

where $\mathbb{E}$ denotes the expectation operator across the data distribution $p(\mathrm{y}, z)$, $\mathcal{L}$ is the per-example loss function, $f(\mathrm{y};\Theta)$ is the prediction for the input y, and z its corresponding target. In practice, we do not know what is the true data generating distribution $p(\mathrm{y}, z)$, instead we use the empirical distribution $\hat{p}(\mathrm{y}, z)$ defined by the trained set $\{\mathrm{y}_i, z_i\}_{i=1}^N$ as an approximation:

$$J(\Theta) = \mathbb{E}_{\hat{p}(\mathrm{y},z)}[\mathcal{L}(f(\mathrm{y};\Theta), z)] = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f(\mathrm{y}_i;\Theta), z_i) \tag{2.13}$$

where N is the number of samples in the training set. Contrary to many traditional machine learning models, the DNN f includes many non-linear elements that makes J a nonconvex function. Hence, we must adopt iterative gradient-based optimizers that do not guarantee convergence in global sense, namely they can lead to a very low cost but never achieve the global minimum. The basics of this type of optimizers are described in the next subsection.

After the optimization process, it is critical to examine how the DNN f with optimized parameters Θ^* behaves in the presence of unseen data, called as *test set*. In this regard, a performance metric P is formulated and evaluated under the test set. One may wonder, then, why P is not used a cost function as we are ultimately interested to minimize – or

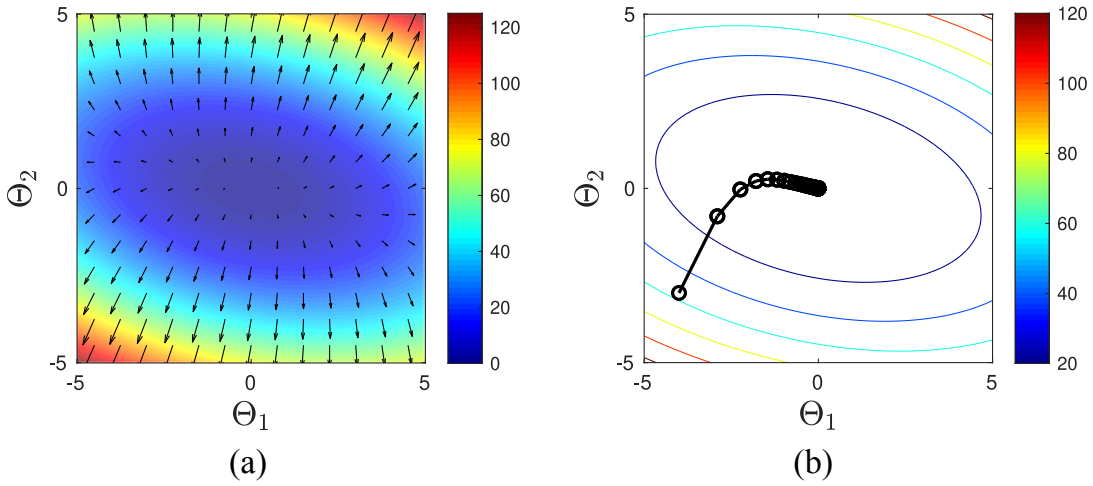


Figure 2.11. *Illustration of gradient descent. (a) Bivariate function and their gradients evaluated at several points (black arrows), (b) Iterations of the gradient descent algorithm*

maximize depending on how P is defined – its value. The answer is simply that P is commonly intractable for the optimization problem, so J is used as a surrogate with the hope to optimize P.

Gradient descent optimization

In multi-variate calculus, the gradient of a scalar-valued function is a vector that contains all the partial derivatives of the function. When the gradient is evaluated at a particular point, the resulting vector points to the direction wherein the function increases the fastest. Figure 2.11(a) illustrates a simple bivariate function as a heatmap and their gradients at various points. Since we are aiming to minimize a cost function $J(\Theta)$, the parameters Θ can be adjusted by moving a small step towards the *opposite* direction of the gradient. Thus, the updating rule of the gradient descent is:

$$\Theta^{new} = \Theta^{old} - \epsilon \nabla_\Theta J(\Theta^{old}) \tag{2.14}$$

where $\epsilon \in \mathbb{R}^+$ is the *learning rate* that determines the size of the step that is taken. Figure 2.11(b) shows every iteration of the gradient descent for a straightforward function and how this yields to the minimum value.

Recalling the expression for the cost function in equation 2.13, the needed gradient can be expanded and entails to an empirical mean of the per-example loss gradients over the whole training set:

$$\nabla_\Theta J(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla \mathcal{L}(f(\mathrm{y}_i; \Theta), z_i) \tag{2.15}$$

Considering the sizes of the dataset in deep learning problems, computing such a gradient is highly expensive. Alternatively, we can just estimate the true gradient – the gradient computed for the whole dataset – by the average over a randomly sampled mini-batch of size $m < N$. Due to this random selection, the gradient computed over the mini-batch

Algorithm 1: Stochastic gradient descent (SGD) update

Result: Optimized parameters Θ^*
Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter Θ
$k \leftarrow 1$;
while *stopping criterion not met* **do**
  Sample a minibatch $\{\mathrm{y}_i, z_i\}_{i=1}^{m}$ from the training set $\{\mathrm{y}_i, z_i\}_{i=1}^{N}$ at random;
  Compute gradient estimate: $\hat{\mathrm{g}} \leftarrow \frac{1}{m}\nabla_\Theta \sum_i \mathcal{L}(f(\mathrm{y}_i; \Theta), z_i)$;
  Apply update: $\Theta \leftarrow \Theta - \epsilon_k \hat{\mathrm{g}}$;
  $k \leftarrow k + 1$;
$\Theta^* \leftarrow \Theta$;

deviates from the true one. Nevertheless, the error in the gradient estimation reduces at lower rate compared to the computational resources as $m \to N$ [24]. Therefore, it is not needed to have a large mini-batch size. Equally interesting, the noise introduced by the random sampling promotes generalization in the test set and helps to escape from bad local minima [48].

At bottom, this random sampling is the main principle of the so-called Stochastic Gradient Descent (SGD), which is pinpointed in Algorithm 1. Noteworthy, the sampling noise does not vanish when approaching a good local minimum, so the learning rate must be gradually decreased over time for convergence. That is why the learning rate at iteration k it is denoted by ϵ_k in the algorithm. The way how the the learning rate is reduced is defined by a learning rate scheduler. Common learning rate schedulers are linear decay, multi-step decay or exponential decay. Those are already implemented in deep-learning libraries such as PyTorch [49].

Certainly, SGD is the most basic gradient-based algorithm that is used to train deep models. Several extensions that regularize the updating rule in some way are commonly used, ADAM being the most popular since it incorporates the ideas of the momentum and AdaGrad [32].

Back-propagation

In practice, the set Θ is easily composed of thousands of parameters that are distributed along the layers of the network f. For this reason, it is extremely challenging to define and evaluate an analytical expression of the gradient for each individual parameter. Back-propagation is thus an algorithm that solves this problem by efficiently computing the required gradients [25]. In short, an input y that is processed by the network f until computing the per-example loss $\mathcal{L}$ can be viewed as a forward pass of a computational graph. This computational graph is comprised of *nodes* that represents the tensors, matrices, vector or scalars that are computed throughout the hidden layers of the network, and *edges* symbolizing the operations that are applied from one node to the other. Back-propagation processes the graph in backward direction and computes the gradients by recursively applying the chain rule. This substantially reduces the runtime because avoids

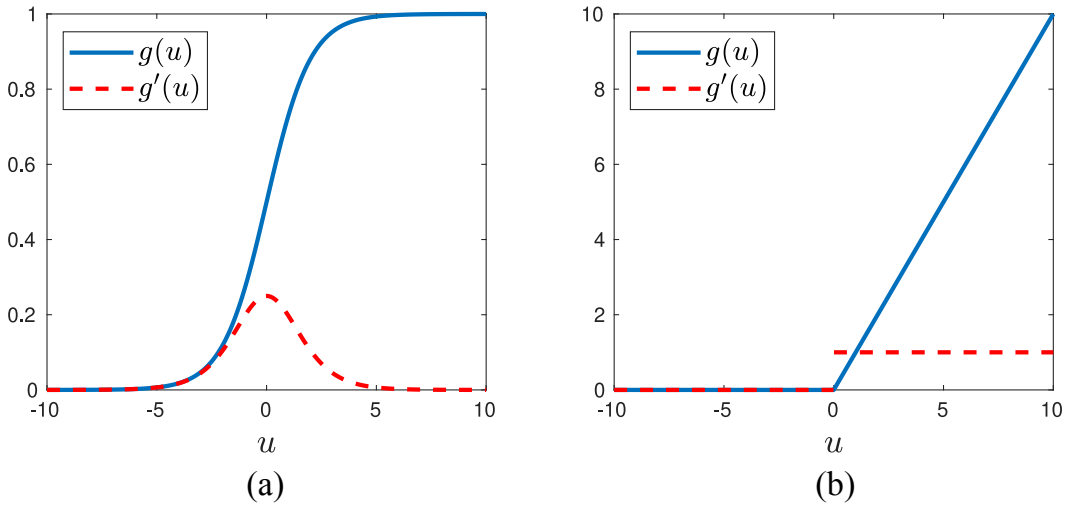


Figure 2.12. *Comparison of activation functions and their derivatives. (a) Sigmoid function, (b) ReLU*

the computation of common subexpressions that are previously computed for higher layers of the network.

Intuitively, the gradients for the bottom layers of the network involve the product of numerous intermediate terms. Since gradients tend to be small, *i.e.*< 1, the total gradient for the parameters placed in such layers approaches 0. Regarding that the parameters move proportional to the gradient, learning is much slower at first layers compared to the parameters placed in last layers. This is known as the *vanishing gradient problem* that was mentioned in previous sections of this chapter. Specifically, we introduced skip connections as a mechanism to reduce the gradient vanishing. The net effect of skip connections in the backward pass is to aggregate the gradients of the last layers, which are larger in general. Therefore, learning in first layers is boosted and yields to higher performance.

The gradient vanishing problem can be caused by a bad selection of activation functions as well. Figure 2.12 depicts the sigmoid and ReLU functions along with their derivatives. It can be seen that when u is around the saturation zone in the sigmoid function, the gradients are nearly zero causing the vanishing gradient problem (Figure 2.12(a)). Conversely, the derivative of the ReLU function is 1 whenever $u > 0$ (Figure 2.12(b)). That explains why the introduction of ReLU units for the hidden layers fosters higher performances of the DNNs.

3 METHODS

In this chapter, the proposed methodology and the techniques utilized for experimentation are described. In section 3.1, the main principle of the deep-learning based approach for VTSR is presented along with its training scheme variants. Then, a full description of the network architecture that has been used for experimentation is provided in section 3.2. Additionally, section 3.3 specifies the supervised loss function that is utilized for training the network. Finally, the selected performance metrics for evaluation are listed in section 3.4.

3.1 Data-driven VTSR approach

The deep-learning based approach proposed in this work consists of training a DNN f that learns to transform two consecutive frames as they were captured half the exposure time. This supervised traning scheme is represented in Figure 3.1. Expressively, the DNN f is trained to learn the ideal mapping function f^*:

$$\left(z[n]|_0^T, z[n+1]|_0^T\right) \xrightarrow{f^*} \left(z[n]|_{T/2}^T, z[n+1]|_0^{T/2}\right) \tag{3.1}$$

For the sake of clarity, every frame captured by the camera device $z[n]|_a^b$ is denoted in equation 3.1 as:

$$z[n]|_a^b = \frac{1}{b-a}\int_{nT+a}^{nT+b} z(t)dt$$

where T is the time frame of the input video, $[a, b]$ is the exposure interval, $n \in \mathbb{Z}$ and $z(t)$ the latent continuous-time varying scene. Therefore, the ideal mapping f^* takes two

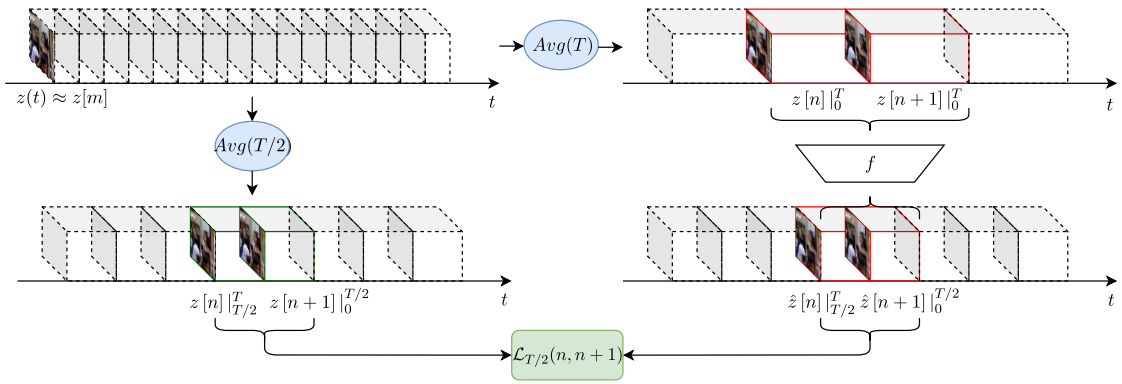


Figure 3.1. *VTSR learning framework.*

consecutive frames $z[n]|_0^T$ and $z[n+1]|_0^T$, integrated over the time intervals $[nT, nT+T]$ and $[(n+1)T, (n+1)T+T]$, respectively, and produces two frames $z[n]|_{T/2}^T$ and $z[n+1]|_0^{T/2}$ exposed during $[nT+T/2, nT+T]$ and $[(n+1)T, (n+1)T+T/2]$. In other words, the frames captured using exposure time T are expanded to frames captured by $T/2$. As it can be inferred from Figure 3.1, the whole video is processed by applying the DNN f in a sliding fashion.

In practice, the signal $z(t)$ is not accessible, which makes difficult to construct the pairs of inputs and targets needed for training in a supervised scheme. This is because one camera cannot synchronously shoot two videos along with different frame rate. Nevertheless, the technological advances of digital cameras in the recent years, make even possible to capture 240-fps videos with cell-phone devices. Having then access to a recording $z[m]$ of the scene $z(t)$ with a high-speed camera, the pairs of inputs and ground-truths can be approximated by the discretization of time with $T = M\tau$; being M positive and even, and τ the frame time of the high-speed video whose frame rate is $r = 1/\tau$. Thus, each one of the terms in the equation 3.1 are approximated by:

$$\begin{aligned}
z[n]|_0^T &= \frac{1}{T}\int_{nT}^{nT+T} z(t)dt \approx \frac{1}{M}\sum_{m=nM}^{nM+M} z[m] \\
z[n+1]|_0^T &= \frac{1}{T}\int_{(n+1)T}^{(n+1)T+T} z(t)dt \approx \frac{1}{M}\sum_{m=(n+1)M}^{(n+1)M+M} z[m] \\
z[n]|_{T/2}^T &= \frac{1}{T/2}\int_{nT+T/2}^{nT+T} z(t)dt \approx \frac{1}{M/2}\sum_{m=nM+M/2}^{nM+M} z[m] \\
z[n+1]|_0^{T/2} &= \frac{1}{T/2}\int_{(n+1)T}^{(n+1)T+T/2} z(t)dt \approx \frac{1}{M/2}\sum_{m=(n+1)M}^{(n+1)M+M/2} z[m]
\end{aligned} \tag{3.2}$$

To put it simply, equation 3.2 says that the pairs of inputs and ground-truths are computed by averaging M and $M/2$ consecutive frames, respectively. Notably, the only assumption is made for this to work, is that the high-speed camera has a shutter nearly always open, *i.e*, one frame period equals the exposure time. With this computational mechanism, it is then possible to impose a loss $\mathcal{L}_{T/2}(n, n+1)$ to train the DNN f that outputs an estimation of the temporally super-resolved frames $\hat{z}[n]|_{T/2}^T$ and $\hat{z}[n+1]|_0^{T/2}$, as depicted in Figure 3.1. More generally, $\mathcal{L}_{T/2^N}(n, n+1)$ refers to the loss function that takes the groundtruth and output frames at indices $n, n+1$, being super-resolved at a frame rate of $2^N/T$. The choice of this loss $\mathcal{L}_{T/2^N}(n, n+1)$ is independent of the proposed learning framework and it is, in fact, comprised of different terms. The actual supervised loss terms utilized along with the VTSR methodology are pinpointed in section 3.3.

Bearing in mind the basic principle of the deep-learning method for VTSR, more complex training procedures can still be added on top of it to avoid possible artifacts in the testing phase. Furthermore, the way how VTSR is presented here can be exploited in a recursive way to accomplish a methodologically-ingenious technique to deblur and interpolate

frames, altogether. These further ideas are disclosed in the reminder of this section.

3.1.1 Recursive VTSR

In essence, the above-mentioned VTSR approach aims at building a DNN f that increases the frame rate and reduces the exposure time in a factor of 2. Then, the recursive application of this model yields to the following result:

$$\begin{aligned}\left(z[n]|_0^T, z[n+1]|_0^T\right) &\xrightarrow{f} \left(\hat{z}[n]|_{T/2}^T, \hat{z}[n+1]|_0^{T/2}\right) \\ &\xrightarrow{f^2} \left(\hat{z}[n]|_{3T/4}^T, \hat{z}[n+1]|_0^{T/4}\right) \\ &\vdots \\ &\xrightarrow{f^N} \left(\hat{z}[n]|_{T-T/2^N}^T, \hat{z}[n+1]|_0^{T/2^N}\right)\end{aligned} \tag{3.3}$$

where N corresponds the number of times that f has been recursively applied, *i.e.*, $f^N = f \circ f^{N-1} = f \circ f \circ f^{N-2} = f \circ \cdots \circ f$, denoting $\circ$ as the composition operator. By making N big enough, the exposure interval of the obtained output frames turns to be infinitesimally small, which implies to achieve the level when even the fastest motion is frozen and the frames become spatially sharp. Overall, the recursive application of the VTSR method allows to increase the time resolution and reduce the blur, simultaneously. Thus, this mechanism corresponds to a novel method to tackle the problem of joint deblurring and frame interpolation. Effectiveness of recursive method, compared also to state-of-the-art techniques in the aforementioned task, is evaluated in section 4.4.

3.1.2 Advanced training schemes

The supervised training approach proposed in section 3.1 simply consists of providing examples of the target frames. We refer to this scheme as *basic* training. Anyway, it might still be a weak regularization to accomplish a good approximation of the temporal super resolution function in equation 3.1, in broader sense. For this reason, more complex training schemes are designed such that they fulfill some of the properties we expect from our VTSR method. Specifically, two more schemes are unveiled here: *reconstruction* and *multilevel* training. The performance of the provided schemes are compared in section 4.3.

Reconstruction training

Since the target mapping function in equation 3.1 only works with a pair of frames, sliding processing is required to fully expand the time resolution of an input video by a factor of 2. This procedure is represented in Figure 3.2. As it is illustrated, the summation of the resulting frames, that are in-between the action of consecutive VTSR models f, equals

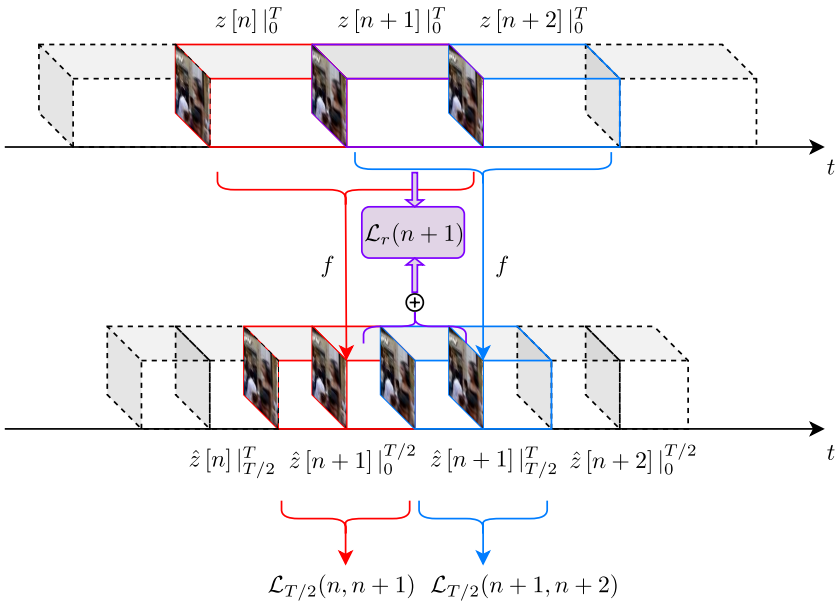


Figure 3.2. *Reconstruction training scheme.*

the middle input frame exposed from 0 to T. Mathematically, it is found that:

$$
\begin{aligned}
z[n+1]|_0^T = & \frac{1}{T}\int_{(n+1)T}^{(n+1)T+T} z(t)dt \\
& \frac{1}{T}\left[\int_{(n+1)T}^{(n+1)T+T/2} z(t)dt + \int_{(n+1)T+T/2}^{(n+1)T+T} z(t)dt\right] \\
& \frac{1}{2}\left[\frac{1}{T/2}\int_{(n+1)T}^{(n+1)T+T/2} z(t)dt + \frac{1}{T/2}\int_{(n+1)T+T/2}^{(n+1)T+T} z(t)dt\right] \\
& \frac{1}{2}\left(z[n+1]|_0^{T/2} + z[n+1]|_{T/2}^T\right)
\end{aligned}
\tag{3.4}
$$

This dictates a useful constraint to guide the training phase. Accordingly, we can instead take triplets of consecutive frames and enforce a reconstruction constraint based on equation 3.4 during training. Thus, the global loss turns to be a sum of the following terms:

$$\mathcal{L} = \mathcal{L}_{T/2}(n, n+1) + \mathcal{L}_{T/2}(n+1, n+2) + \lambda_r \mathcal{L}_r(n+1) \tag{3.5}$$

where $\mathcal{L}_{T/2}(n, n+1)$, $\mathcal{L}_{T/2}(n+1, n+2)$ are the supervised loss terms computed with the respective ground-truth and output frames, λ_r a weighting hyper-parameter, and $\mathcal{L}_r(n+1)$ is the reconstruction loss term given by:

$$\mathcal{L}_r(n+1) = \|\frac{1}{2}\left(\hat{z}[n+1]|_0^{T/2} + \hat{z}[n+1]|_{T/2}^T\right) - z[n+1]|_0^T\|_1 \tag{3.6}$$

Roughly speaking, this scheme allows the network to produce outputs coherent with the input and promotes temporal consistency.

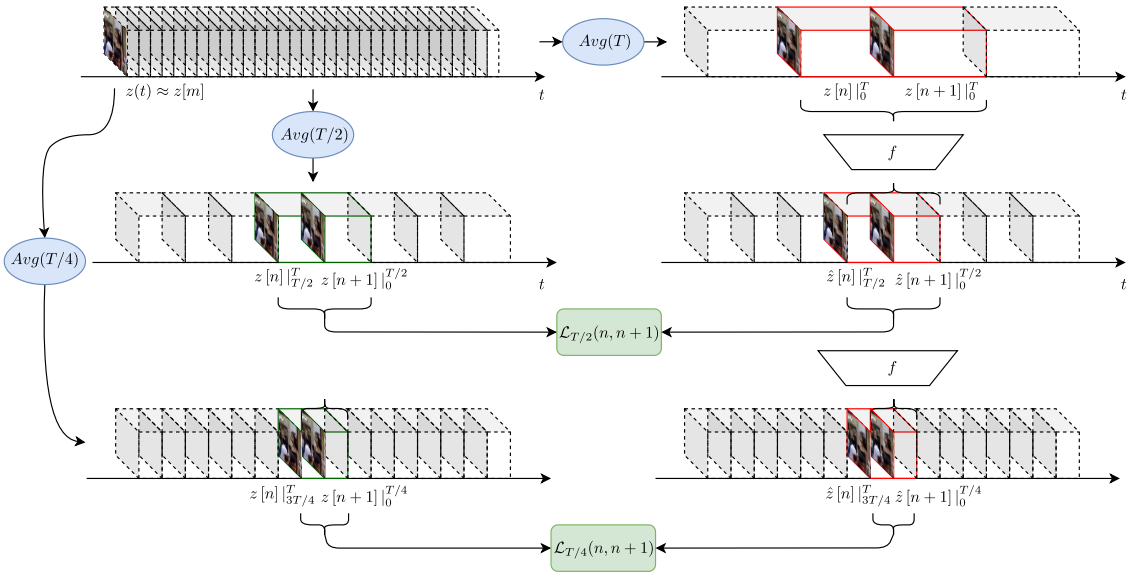


Figure 3.3. *Multilevel training scheme.*

Multilevel training

The ultimate goal of VTSR is to find such a mapping function f that expands the time resolution no matter what is the frame rate in the input. Secondly, we want to reach the point of motionless video by recursion. Nonetheless, even if the VTSR network is trained under several time expansion levels, it is clear that the space of input images differs in recursive settings because of the possible artifacts produced by the network itself. In fact, this difference is more noticeable for deeper time expansions as the amount of artifacts increases in every recursive application. To deal with this issue, we can supervisely train the VTSR network f regarding multiple resolution levels that result from the recursive application of f. To prevent a huge overload in training, we only consider the expansion up to two higher levels as shown in Figure 3.3. The global loss function is thereby computed as $\mathcal{L} = \mathcal{L}_{T/2}(n, n+1)+\mathcal{L}_{T/4}(n, n+1)$, where $\mathcal{L}_{T/2}(n, n+1)$ and $\mathcal{L}_{T/4}(n, n+1)$ correspond to the supervised loss terms when the time is expanded by 2 and 4, respectively. In this way, the network has at least a mechanism to correct inaccuracies produced after a recursion.

3.2 Neural network architecture

The general architecture for the VTSR network is illustrated in Figure 3.4. This structure is mainly inspired by state-of-the-art deblurring neural networks: DeblurGANv2 [10] and EDVR [9]. First of all, the proposed network architecture takes a pair of consecutive frames $z[n]|_0^T$ and $z[n+1]|_0^T$ and by residual-learning produces the estimated target frames $\hat{z}[n]|_{T/2}^T$ and $\hat{z}[n+1]|_0^{T/2}$. It means that the network only learns the needed pixel-wise changes that are applied to the inputs. In fact, the residual scheme has demonstrated more accurate results than the standard reconstruction in different image restoration task [39, 41] and that is why is also used in many deblurring networks [3, 9, 10, 42]. The architecture is composed of the four main components. The Feature Pyramid

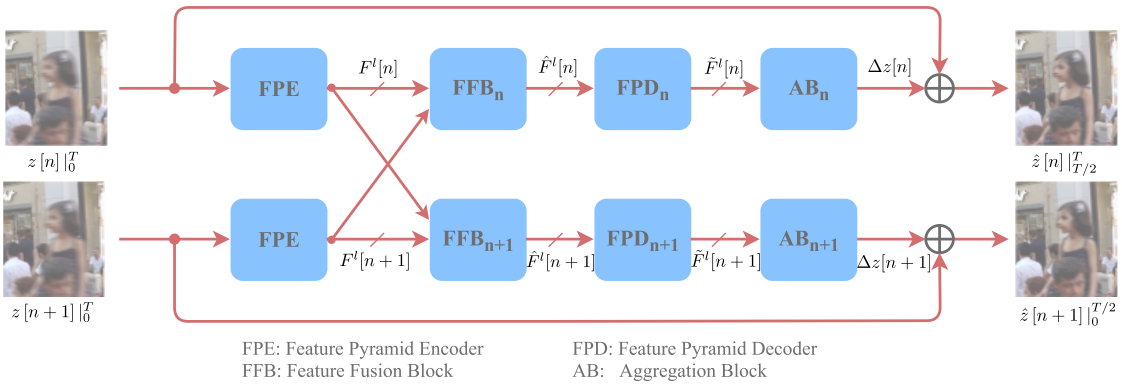


Figure 3.4. *Overview of the VTSR pipeline architecture.*

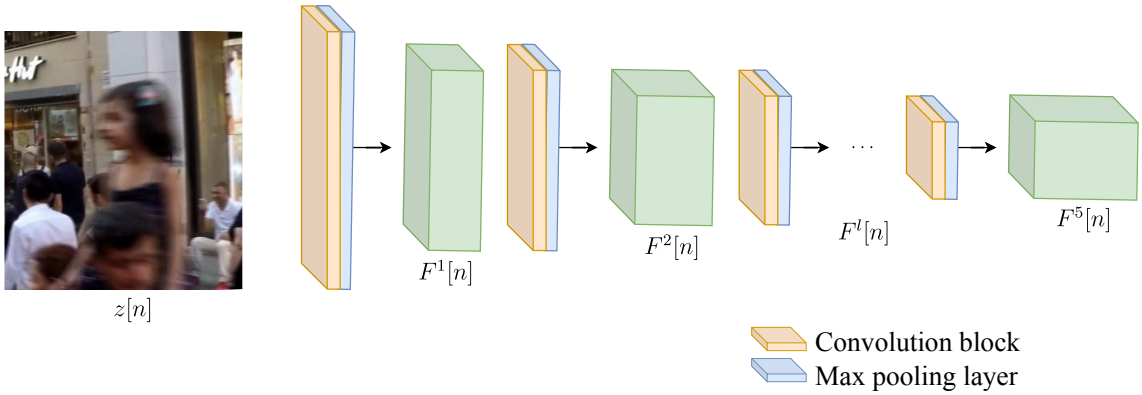


Figure 3.5. *Structure of the FPE block.*

Encoder (FPE) and Feature Pyramid Decoder (FPD) blocks are familiar from U-Net-like structures [40]. The Feature Fusion Block (FFB) combines features extracted from the two frames for the decoder and Aggregation Block (AB) aggregates features from multiple resolutions. Typically, the weights of each one of the convolutional filters in those blocks are different, except in the FPE blocks which share the same coefficients. A detailed explanation of these blocks is found below.

3.2.1 Feature Pyramid Encoder (FPE)

This processing block extracts a multi-scale feature representation for a given image. Towards this end, we use a DNN structure as it is illustrated in Figure 3.5. Technically, this DNN is comprised of convolutional blocks and maxpooling layers that downsample the features. Unlike the encoder structure of the image restoration networks presented in section 2.3.1, convolutional block involve more complex layers. Those blocks are based on the backbone networks used in image classification problems since the trained models have demonstrated to successfully extract more semantic information of the input images. Inspired by the work of Kypyn *et al.* [10], the MobileNetv2 backbone network [50] is used in our network since it provides a good trade-off between good contextual representation and computational resources required. In total, two deep pyramid feature representation

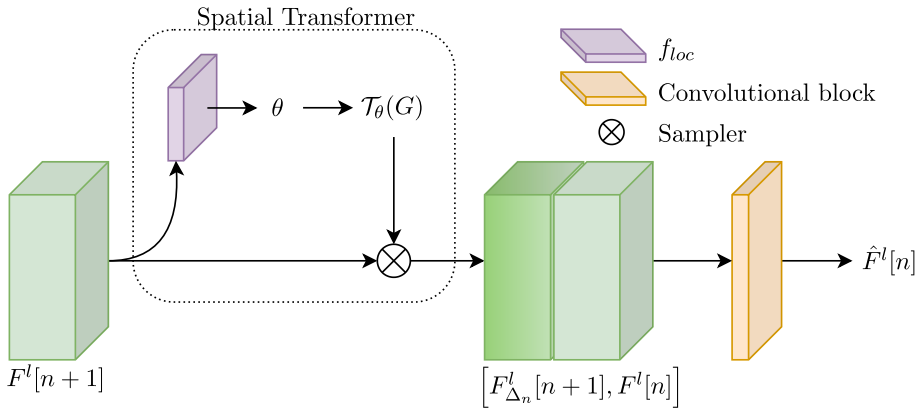


Figure 3.6. *Fusion block with pre-alignment of features.*

$F^l[n]$ and $F^l[n+1]$, such that $l = 1, \ldots, 5$, are extracted from the input frames $z[n]$ and $z[n+1]$, respectively.

3.2.2 Feature Fusion Block (FFB)

Extracted features from both frames are fused to incorporate relevant information that is found in the other frame. At a l-level, fused features $\hat{F}^l[n]$ and $\hat{F}^l[n+1]$ can be simply obtained by fusion convolution, namely:

$$\begin{aligned} \hat{F}^l[n] &= g_n\left(\left[F^l[n], F^l[n+1]\right]\right) \\ \hat{F}^l[n+1] &= g_{n+1}\left(\left[F^l[n+1], F^l[n]\right]\right) \end{aligned} \tag{3.7}$$

where g is a function consisting some convolutional layers and $[\cdot,\cdot]$ denotes the concatenation operation.

Notwithstanding, fusion convolution in equation 3.7 may not easily infer the inter and intra-frame complexities that are caused by the presence of occlusions, paralax problems and the misalignment of semantic elements among the given frames. Based on the work by Wang *et al.* [9], two modules can be incorporated in this block for an effective and efficient aggregation of the relevant information found in the given frames: *Spatial pre-alignment of features* and *Spatio-temporal attention mechanism*. These modules are described below and the benefit of adding them in the FFB is analized in our ablation studies – section 4.2.

Spatial pre-alignment of features

One of the issues that challenges the fusion among the given frames is the misalignment due to the motion of the camera or objects in the scene. Inspired by the networks EDVR [9] and TDAN [51], this module allows supportive frames to be spatially aligned to a reference, at feature level across the pyramid encoders. Being precise, $F^l[n+1]$ is firstly aligned to the reference $F^l[n]$ to produce the fused map $\hat{F}^l[n]$, while $F^l[n]$ is aligned to

$F^l[n+1]$ before generating $\hat{F}^l[n+1]$. In this work, instead, we assume that the alignment of features can be achieved by an affine transformation conditioned to the input by using spatial transformer networks [52].

Considering, for instance, the case in which $F^l[n]$ is the reference feature map, $F^l[n+1]$ is aligned and fused as shown in Figure 3.6. Inside the spatial transformer, there is a *localization network* f_{loc} that estimate the parameters θ of the affine transformation that should be applied to the feature map $F^l[n+1]$, *i.e.*, $\theta = f_{loc}(F^l[n+1])$. Then, the *grid generator* $\mathcal{T}_\theta(G)$ defines the set of points where the input feature map $F^l[n+1]$ should be sampled to produce the desired transformation, such that the output pixels lie on a regular grid $G = \{G_i\} = \{(x_i^t, y_i^t)\}$. In this case, the point-wise transformation is:

$$\begin{bmatrix} x_i^s \\ y_i^s \end{bmatrix} = \mathcal{T}_\theta(G_i) = A_\theta \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix} \tag{3.8}$$

where (x_i^t, y_i^t) are coordinates of the regular grid in the transformed feature domain, (x_i^s, y_i^s) are the corresponding sampling points in the input feature map, and A_θ is the 2-by-3 affine transformation matrix. For computational reasons, both pairs of coordinates are normalized in the range $[-1, 1]$. At last, the *sampler* determines the value for every i pixel in the spatially modulated feature map $F^l_{\Delta_n}[n+1] = \{V_i\}$ by the bilinear interpolation of the pixel values in the input feature map $F^l[n+1] = \{U_{jk}\}$ as follows:

$$V_i = \sum_j^H \sum_k^W U_{jk} \max(0, 1 - |x_i^s - k|) \max(0, 1 - |y_i^s - j|) \tag{3.9}$$

where U_{jk} is the pixel value of the input feature map $F^l[n+1]$ at location (k, j), V_i is the output value for the pixel i of the modulated feature map $F^l_{\Delta_n}[n+1]$ that is located at (x_i^t, y_i^t), and H, W are the spatial dimensions of the input feature map $F^l[n+1]$. Likewise, the modulated feature map $F^l_{\Delta_{n+1}}[n]$ is the result of aligning $F^l[n]$ to the reference $F^l[n+1]$. Subsequently, $\hat{F}^l[n]$ and $\hat{F}^l[n+1]$ are fused through equation 3.7 with $\left[F^l_{\Delta_n}[n+1], F^l[n]\right]$ and $\left[F^l_{\Delta_{n+1}}[n], F^l[n+1]\right]$, respectively.

Spatio-temporal (ST) attention mechanism

The relevant information from temporally separated frames is conveyed through soft-attention maps, similarly as shown in Figure 3.7. Following the strategy of Wang *et al.* [9], the attention weights a_w for the reference frame n are computed as:

$$a_w(F^l[n], F^l[m]) = \sigma\left(\psi(F^l[n])^T \phi(F^l[m])\right) \tag{3.10}$$

where $m \in \{n, n+1\}$, $\psi(F^l[n])$ and $\phi(F^l[m])$ are two embeddings obtained by a convolution block, and $\sigma(\cdot)$ denotes the sigmoid function that keeps the maps in the range

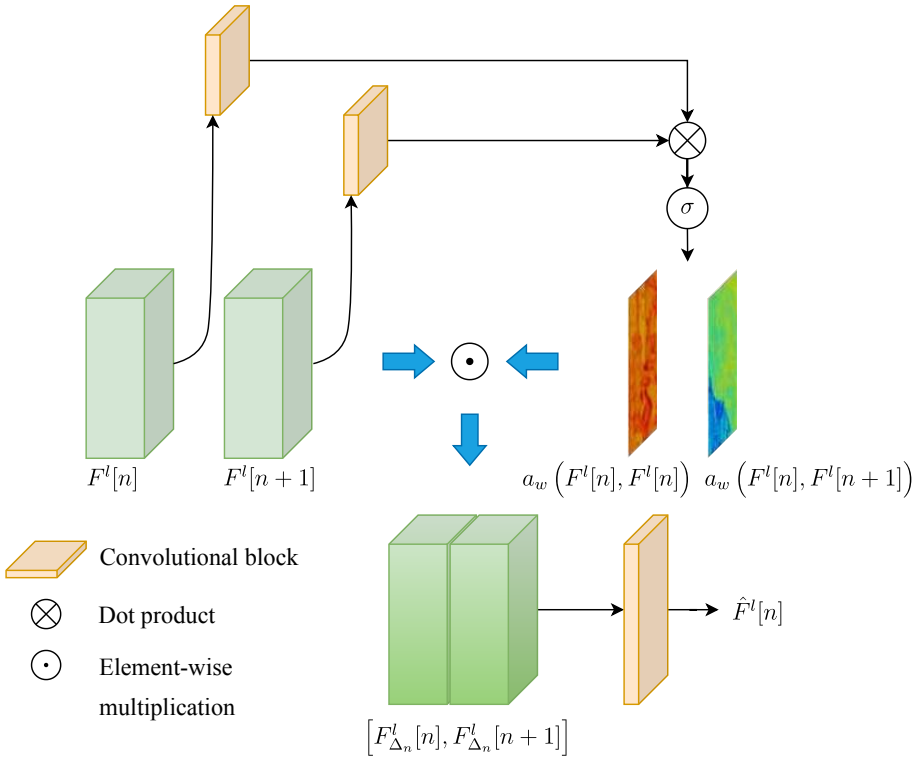


Figure 3.7. *Fusion block with spatio-temporal attention module.*

$[0, 1]$:

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

Concretely, equation 3.10 measures the spatially-specific similarity distance between a supporting frame m and the reference n, in an embedding space.

Subsequently, the spatio-temporal maps a_w are used to emphasize the important features that boost the fusion process. Thus, attention-modulated features $F^l_{\Delta_n}[m]$, referenced to the frame n, are calculated by pixel-wise multiplication with the original features $F^l[m]$ as:

$$F^l_{\Delta_n}[m] = F^l[m] \odot a_w(F^l[n], F^l[m]) \tag{3.11}$$

Finally, the fused features $\hat{F}^l[n]$ and $\hat{F}^l[n+1]$ are obtained by using the fusion convolution in equation 3.7 with $\left[F^l_{\Delta_n}[n], F^l_{\Delta_n}[n+1]\right]$ and $\left[F^l_{\Delta_{n+1}}[n+1], F^l_{\Delta_{n+1}}[n]\right]$, respectively.

3.2.3 Feature Pyramid Decoder (FPD)

As typically done in U-net based architectures, higher spatial resolution features are reconstructed as:

$$\tilde{F}^l[n] = g\left(\hat{F}^l[n] + (\hat{F}^{l+1}[n])^{\uparrow 2}\right) \tag{3.12}$$

where $(\cdot)^{\uparrow s}$ refers to the upsampling operator by a factor s, and $l = 1, \ldots, 4$. As in any decoder structure, spatially higher levels of the pyramid are reconstructed from semantically richer features $\hat{F}^{l+1}[n]$, along with the image details contained in $\hat{F}^l[n]$.

3.2.4 Aggregation Block (AB)

The goal of this component is to aggregate the features at different resolutions such that produces the required residual changes $\Delta z[n]$ and $\Delta z[n+1]$ that need to be applied to the inputs $z[n]|_0^T$ and $z[n+1]|_0^T$, respectively. For this purpose, high levels of the feature pyramid are firstly upsampled and fusion convolution layers are used for the aggregation. For instance:

$$\Delta z[n] = h_n\left(g_n\left(\left[\tilde{F}^1[n], (\tilde{F}^2[n])^{\uparrow 2}, \ldots, (\tilde{F}^l[n])^{\uparrow 2^{l-1}}\right]\right)^{\uparrow 2}\right)$$

where h denotes a general convolution operation followed by a $tanh$ activation layer to keep the output in the range of $[-1, 1]$.

3.3 Loss function

Given a pair of consecutive outputs $\hat{z}[n]|_{T-T/2^N}^T$ and $\hat{z}[n+1]|_0^{T/2^N}$ – obtained by applying f^N– and their corresponding ground-truth frames, the supervised loss function is a linear combination of the following terms:

$$\mathcal{L}_{T/2^N}(n, n+1) = \lambda_1\mathcal{L}_1 + \lambda_p\mathcal{L}_p + \lambda_s\mathcal{L}_s \tag{3.13}$$

where $\lambda_1, \lambda_p, \lambda_s$ correspond to the weighting values which are set by experimentation.

Pixel-wise loss $\mathcal{L}_1$: This term incorporates the widely used pixel-wise error between the estimation and the groundtruth in L_1 sense. This term is computed as:

$$\mathcal{L}_1 = \|\hat{z}[n]|_{T-T/2^N}^T - z[n]|_{T-T/2^N}^T\|_1 + \|\hat{z}[n+1]|_0^{T/2^N} - z[n+1]|_0^{T/2^N}\|_1$$

Perceptual loss $\mathcal{L}_p$: In order to encourage more visually convincing frames to human eye, we use the perceptual distances that makes estimated frames sharper. This loss is defined as:

$$\mathcal{L}_p = \|\phi(\hat{z}[n]|_{T-T/2^N}^T) - \phi(z[n]|_{T-T/2^N}^T)\|_2 + \|\phi(\hat{z}[n+1]|_0^{T/2^N}) - \phi(z[n+1]|_0^{T/2^N})\|_2$$

where ϕ refers to the `conv4_3` feature maps of the ImageNet pretrained VGG16 model [34].

Sharpness loss $\mathcal{L}_s$: This term is added to emphasize sharpness at the edges. It is defined as:

$$\mathcal{L}_s = \|\nabla^2\hat{z}[n]|_{T-T/2^N}^T - \nabla^2 z[n]|_{T-T/2^N}^T\|_1 + \|\nabla^2\hat{z}[n+1]|_0^{T/2^N} - \nabla^2 z[n+1]|_0^{T/2^N}\|_1$$

where ∇^2 denotes the laplacian operator computed by the convolution with the laplacian kernel k:

$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

3.4 Image quality metrics

For the quantitative assessment of the proposed VTSR method, frame-wise quality is evaluated by computing two widely-used full-reference metrics for image restoration problems: *Peak Signal-to-Noise Ratio* (PSNR) and *Structural Similarity Index Measure* (SSIM). In this context, "full-reference" metrics means that a distortion-free image (ground-truth) is used as a reference to evaluate the quality of the method estimation.

Peak Signal-to-Noise Ratio (PSNR): It is an error-based metric that has a clear physical meaning. Specifically, it measures the ratio between maximum power of the signal and the power of the noise, in logarithmic scale. Understanding the noise as the error image, the PSNR is mathematically expressed as:

$$PSNR(z, \hat{z}) = 10 \log_{10} \left(\frac{\max^2(z)}{\frac{1}{M} \|z - \hat{z}\|_2^2} \right) \tag{3.14}$$

where M denotes the image size, while z and $\hat{z}$ are the reference and estimation, respectively.

Alternatively, the Signal-to-Noise Ratio (SNR) differs from PSNR in the numerator of the ratio since it considers the power of the signal, instead of its maximum. Nevertheless, we only use the PSNR as quality metric because it is less content-dependent. For instance, let us consider two images, one brighter than the other in average. Assuming the estimation achieves the same mean squared error (equivalent to the power of the noise) for both images, the SNR is higher for the brighter image. This is because the power of the signal is correlated to the image brightness. Conversely, the PSNR keeps the same as the maximum is usually 255 per color channel in a 8-bit representation. Therefore, the PSNR value lean on the estimation accuracy rather than the image content.

Structural Similarity Index Measure (SSIM): In contrast to the PSNR, SSIM is a perceptual-based quality metric. In other words, the quality of the image is evaluated by modeling how the human visual system perceive images. Wang *et al.* [53] devised this metric by assuming that the change in structural information provides a suitable measurement of the perceived distortion. Figure 3.8 depicts the block diagram for its computation. First, luminance and contrast measurements are extracted and compared to each other by computing local means and standard deviations, respectively. Then, the

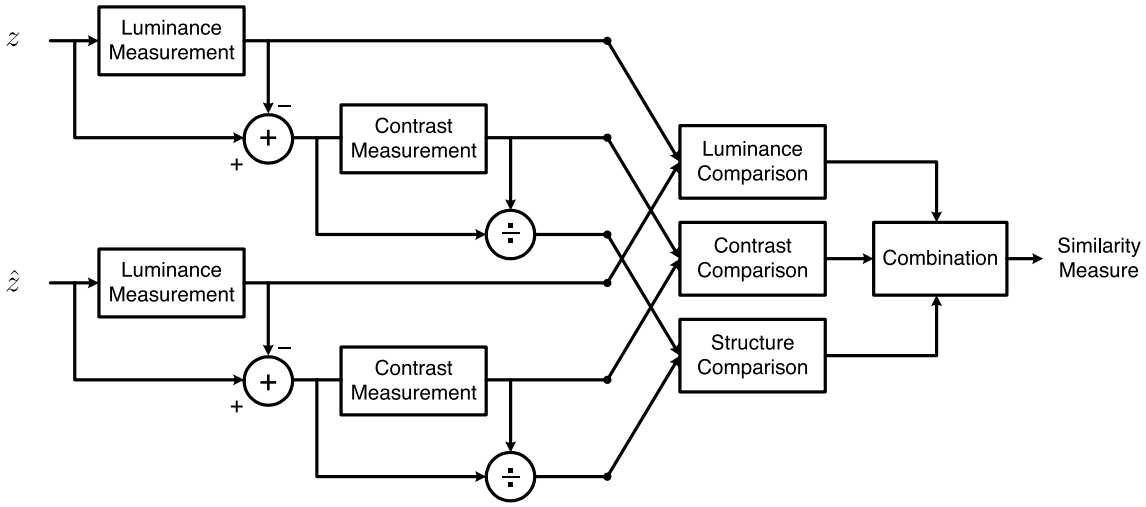


Figure 3.8. *Diagram block of SSIM measurement system.* ***Source:*** *[53]*

structural similarity is extracted by comparing local patterns of pixel intensities that have been normalized for luminance and contrast, as shown in the diagram. Ultimately, the SSIM is combination of the luminance, contrast and structure similarities. For a detailed description of the involved expressions in the SSIM computation, the reader is referred to the authors' work [53].

4 EXPERIMENTS

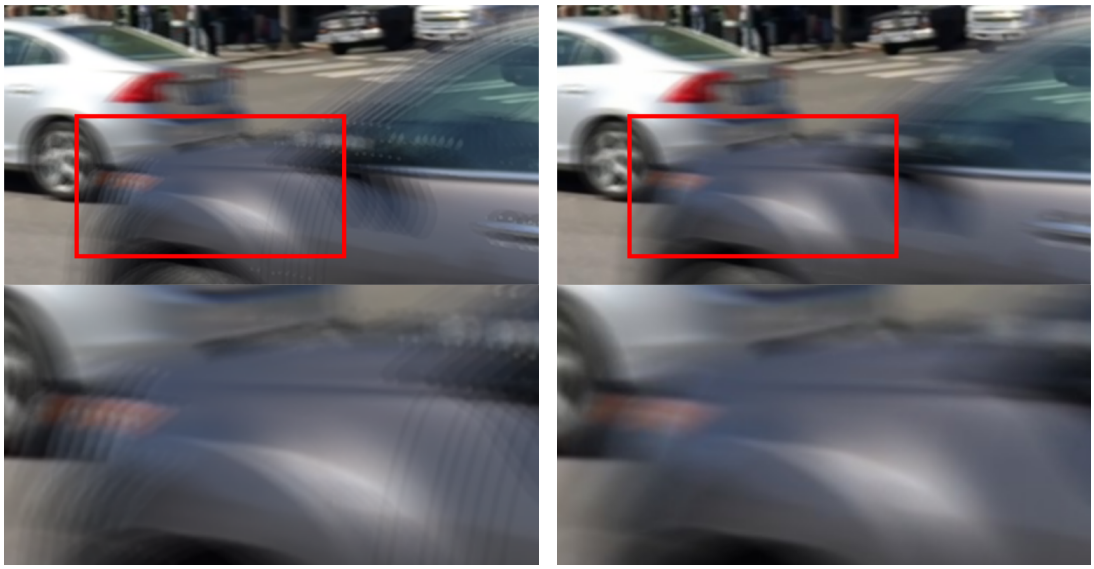
This chapter describes the experiments and their corresponding results in order to study the capabilities of the proposed deep-learning solution for the VTSR problem. First, we describe the general settings throughout all the performed experiments in section 4.1. Secondly, some ablation studies in the network architecture are presented in section 4.2. Then, section 4.3 introduces the comparison results between our the different training schemes that are designed for the network. Finally, a comparison against recent methods for joint deblurring and frame interpolation are outlined in section 4.4.

4.1 Experimental settings

4.1.1 Datasets

For video deblurring there are several datasets which contain fast frame rate videos: GOPRO [54], DVD [42] and Sony Low-Motion (Sony) [13]. The *GOPRO* dataset is comprised of 33 720p-videos captured by a GoPro Hero 4 camera at a frame rate of 240 frames per second (fps). We adopted the train/test split suggested by the authors, namely, 22 and 11 video sequences for training and testing, respectively. The *DVD* dataset contains 133 high-speed videos recorded at 240fps and 720p resolution. Those videos were captured by several cameras such as iPhone 6s, GoPro Hero 4 and Nexus 5x. Since many videos involve strong artifacts in this dataset, we only included 68 videos which were randomly split to the training (56) and testing (12) sets. The *Sony* dataset was collected by using a Sony RX V camera at 250 fps. In total, it contains 63 video sequences split into 46 and 17 sets for training and testing, respectively. Roughly, each video is comprised of 1000 frames at a 1080p resolution. Qualitatively, all datasets include dynamic objects and camera movements from outdoor and indoor scenes.

With the purpose of avoiding dataset bias, we included also 2 videos captured by a RED camera at 300 fps and 2K resolution. Those videos are generously provided by Huawei company in Tampere, Finland. Such videos are exclusively used in evaluation for the fair comparison between our VTSR method and joint deblurring and frame interpolation methodologies. Thus, we compare performances with data unseen during training or validation. We refer to those videos as the *HuaweiRED* dataset.



(a) ***(b)***

Figure 4.1. *Visual effect of pre-interpolation step for the blur generation. (a) Without frame-interpolation. (b) With frame-interpolation*

4.1.2 Data preparation

It must be pointed out that datasets differ in terms of resolution. To avoid performance differences caused by the input image resolution, datasets are converted at 720p. Obviously, this only applies for Sony and Huawei datasets which have higher resolution. In practice, Sony is converted to such resolution by using the FFmpeg tool [55], whereas REDLINE [56] is used for Huawei videos because this is the associated software tool to the RED camera that has been used for their acquisition.

Furthermore, we realized by visual inspections that video frames do not fully satisfy the assumption of shutter almost open, particularly for GOPRO and DVD videos. To alleviate this issue, we interpolate 7 frames and average them including the reference frame as a surrogate of true full-exposed frames. To this aim, we utilized the frame interpolation method proposed by Jiang *et al.* [11] which allows the computation of multiple intermediate frames in one step. This pre-processing step also helps in the generation of more real blur when averaging consecutive frames as it is illustrated in Figure 4.1. In Figure 4.1(a), it can be noticed some jerkiness problem in the obtained blur near strong edges, especially around the nearest window car. On the contrary, the produced blur looks smoother when pre-interpolation of frames is applied before the blur generation (Figure 4.1(b)). In this manner, videos look like they were full-exposed.

4.1.3 Implementation details

The proposed network and its model variations are implemented with Pytorch [49]. In the training phase, we keep 90% of the video clips for actual training, the rest is used for validation. Moreover, we generate the pairs of inputs and outputs on the fly. Specifically,

Recursion level	FFB variation	PSNR	SSIM
f	*baseline*	**28.473**	**0.8732**
	+ST attention	28.344	0.8719
	+pre-alignment	27.567	0.8637
	+ST attention+pre-alignment	27.582	0.8619
f^2	*baseline*	**26.387**	**0.8301**
	+ST attention	25.386	0.8121
	+pre-alignment	25.466	0.8109
	+ST attention+pre-alignment	25.108	0.8052
f^3	*baseline*	**24.201**	**0.7684**
	+ST attention	23.681	0.7564
	+pre-alignment	23.866	0.7591
	+ST attention+pre-alignment	23.616	0.7548

Table 4.1. *Ablation studies on DVD dataset of the Feature Fusion Block (FFB). The fusion of features is mainly performed by fusion convolution of equation 3.7 (baseline). Yet, we can incorporate the Spatio-temporal attention module (+ST attention) and/or Spatial pre-alignment of features (+pre-alignment).*

we randomly average 8, 4, or 2 consecutive frames for the inputs and the outputs are then accordingly averaged by 4, 2, or 1 frames. This mechanism forces the network to increase the frame rate by 2 no matter what is the input frame rate. However, we only sample 20 pairs of inputs and outputs per video sequence for faster computation. Also, frames within the same video are highly correlated, then the network does not need to sample every single frame to learn from that data. Subsequently, images are rescaled to an image size of 512x512 and then randomly cropped to a size of 224x224. Random horizontal flipping is the only data augmentation mechanism included. The batch-size is set to 8 and models are trained for 100 epochs using Adam optimizer with a starting learning rate of 1e-4. We incorporate a multi-step scheduler that change the learning rate with gamma 0.7 every 10 epochs. In addition, we freeze for one epoch the backbone in the FPE with weights pretrained on ImageNet [31]. For the subsequent epochs, those weights are freely optimized in the training phase.

4.2 Ablation studies

An ablation study on the Feature Fusion Block (FFB), presented in section 3.2.2, is performed to assess the impact of including the *Spatio-temporal attention module* and the *Spatial pre-alignment of features*. In this experiment, we only used the DVD dataset and considered the *basic* training scheme, which is refered in section 3.1.2. For training, the weights of the supervised loss function are set to $\lambda_1 = 0.6, \lambda_s = 0.15, \lambda_p = 0.05$. For testing, original videos with frame rate r are consecutively averaged every 8 frames and

take them as the input – input videos then have a frame rate of $r/8$. As our network is trained to double the time resolution, we recursively apply the method to recover the original video. Expressively, we apply $f^3 = f \circ f \circ f$ to an input video with frame rate $r/8$ in order to produce a video with frame rate r, which matches to the original video.

Table 4.1 summarizes the PSNR and SSIM values obtained for each model variation and for each one of the three recursion levels. In this table, *baseline* stands for to the fusion that is performed by the fusion convolution of equation 3.7. The addition of the *Spatio-temporal attention module* and the *Spatial pre-alignment of features* are denote by *+ST attention* and *+pre-alignment*, respectively. Interestingly, performance always drops by the recursive application of the proposed method, no matter the model variation. This is easily expected since ground-truths of higher time-expansion levels become less similar to the input which makes things more challenging for the networks. Besides, artifacts surely accumulate by recursion. Yet we hypothesized that either the *Spatial pre-alignment of features* or the *Spatial pre-alignment of features* may aid the fusion process, we failed to demonstrate such idea since the *baseline* outperforms any other model variation in terms of, both, PSNR and SSIM. On one hand, scenes contain objects that move in many different ways, and so features do too. Hence, the affine model that was assumed for *Spatial pre-alignment of features* might be quite rigid to properly align the features coming from different frames. Perhaps, less restrictive models such as deformable convolution blocks [57] or flow-based alignment modules would suit better for this task. On the other hand, the baseline architecture might be complex enough that it does not gain much flexibility by adding the concerned modules in the FFB. Thus, we choose the *baseline* model of the FFB for our network architecture and use it for the rest of the experiments.

4.3 Comparison of training schemes

In section 3.1.2, three different schemes have been devised for the training of the proposed deep-learning-based VTSR method: *basic*, *reconstruction*, and *multilevel* training. In this experiment, we aim at comparing the effectiveness of those strategies by using the GOPRO and Sony datasets. We keep the same weights of supervised loss function of the previous experiment ($\lambda_1 = 0.6, \lambda_s = 0.15, \lambda_p = 0.05$). Additionally, we set $\lambda_r = 0.15$ that is used in the reconstruction training scheme. For testing, we again average 8 consecutive frames to construct the inputs and use the recursion to recover the original video – applying f^3.

Table 4.2 presents the PSNR and SSIM results obtained. Indeed, the advanced schemes (*reconstruction*, *multilevel* training) tend to improve the performance as they are conceived to regularize the temporal super-resolution problem. Notwithstanding, it is not clear which of them turn to be better. Although the multilevel scheme overcomes the others in GOPRO dataset, the reconstruction scheme gets some of the best metrics in Sony dataset. Moreover, the gap in the quantitative results seems not to be significant. Regarding then the training time for each one of the schemes that are listed in Table

		GOPRO		Sony	
Recursion level	Scheme	PSNR	SSIM	PSNR	SSIM
f	Basic	28.341	0.8795	35.664	0.9621
	Reconstruction	28.316	0.8768	**35.699**	0.9612
	Multilevel	**28.783**	**0.8905**	35.274	**0.9652**
f^2	Basic	26.014	0.8346	33.611	0.9470
	Reconstruction	26.033	0.8345	**33.617**	0.9466
	Multilevel	**26.220**	**0.8426**	33.264	**0.9481**
f^3	Basic	23.889	0.7780	32.326	0.9337
	Reconstruction	23.913	0.7791	**32.339**	**0.9388**
	Multilevel	**23.975**	**0.7835**	32.186	0.9340

Table 4.2. *Quantitative results for training schemes on GOPRO and Sony. Such training schemes are described in section 3.1.2.*

	Basic	Reconstruction	Multilevel
Training time	19h 0m 16s	26h 41m 38s	54h 23m 7s

Table 4.3. *Training times on GOPRO and Sony.*

4.3, we conclude that the reconstruction scheme has the best trade-off between training computational cost and performance. It is observed from Table 4.3 that the multilevel scheme almost doubles the training time of the reconstruction scheme, but it does not gain a substantial improvement.

Curiously, what is more noticeable from Table 4.2 is the performance gap between the GOPRO and Sony datasets. For instance, the PSNR is more than 7 dB lower in GOPRO compared to Sony. That is explained by the dataset quality. As mentioned in section 4.1, the frame rates for GOPRO and Sony datasets are respectively 240 and 250 fps. Then, as inputs are equally generated by averaging 8 consecutive frames, blur becomes stronger in GOPRO due to longer temporal distances between frames. Likewise, GOPRO contains mostly blur produced by drastic camera movements, which we found more challenging for our method. In contrast, sequences in Sony tend to include more blur produced by dynamic objects which is local and easier to inferred for the network.

For qualitative assessment, we show some visual examples in Figure 4.2. Examples include the input and the subsequent time-expanded frames until recovering the original frame rate, *i.e.*, apply f^3 to recursively accomplish $r/8 \rightarrow r$. Overall, our method progressively makes frames sharper under the recursive application of the network. Especially, Figure 4.2(a) shows a successful recovery under the presence of a smooth camera movement. Edges and persons contour increasingly get sharp, even the left-side letters. Similarly, satisfactory temporal super-resolution is demonstrated in Figure 4.2(b)

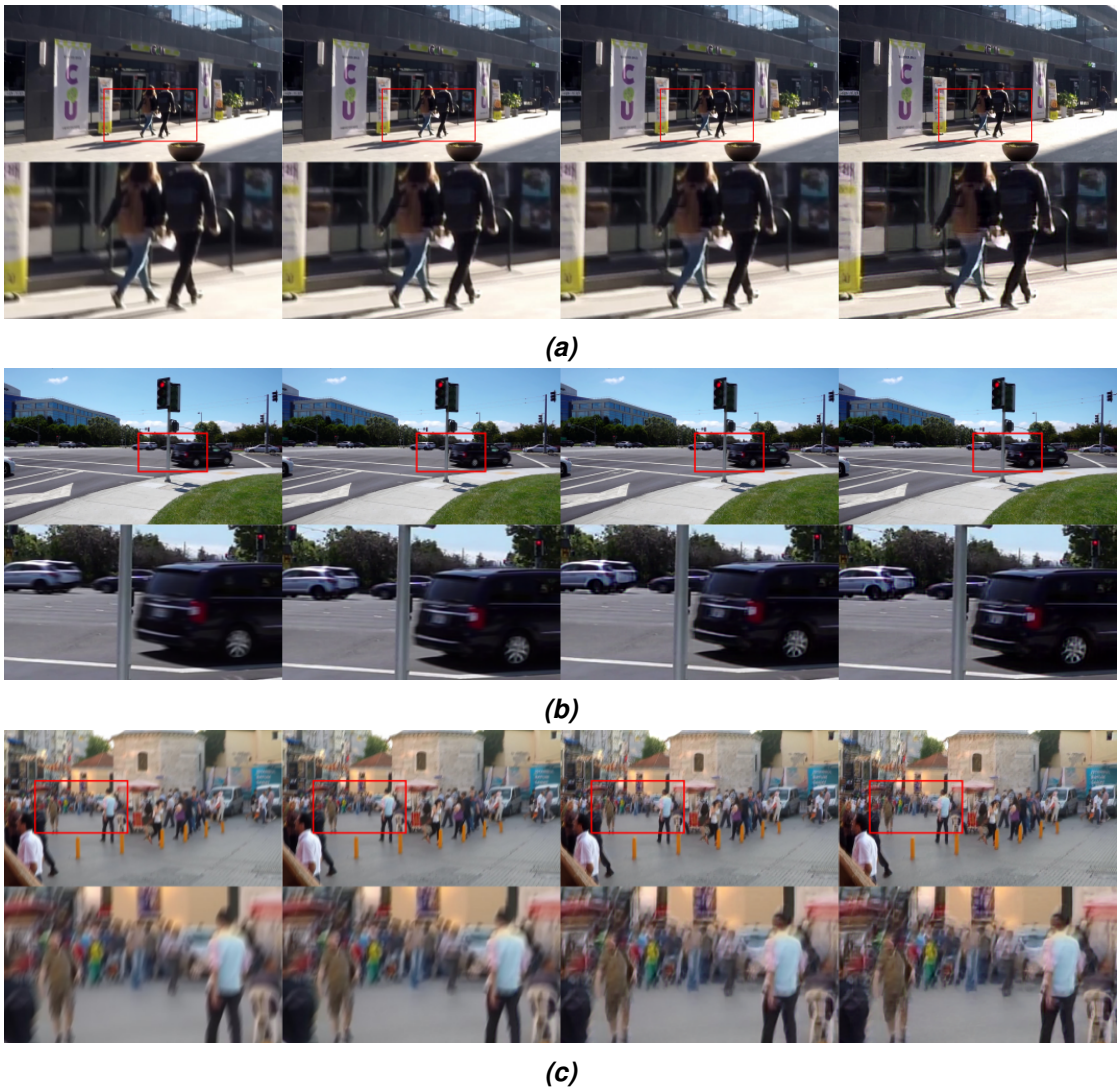


Figure 4.2. *Examples of visual results on GOPRO and Sony. From left to right: input* $(r/8)$*,* f $(r/8 \rightarrow r/4)$*,* f^2 $(r/4 \rightarrow r/2)$*,* f^3 $(r/2 \rightarrow r)$*. (a) Successful results on GOPRO, (b) Successful results on Sony, (c) ineffective result on GOPRO.*

that involves an input with dynamic objects as cars are. It is observed the gradual edge enhancement for the background cars and for the front wheel. Nevertheless, we found the proposed method to be sensible towards strong global motion as shown in Figure 4.2. Although edges turn sharper and sharper, an artistic-like appearance is obtained, which is not what one expects from a temporal super-resolution system. We argue that the multi-scale feature pyramid strategy is not enough to deal with strong global blur. A good solution would be to incorporate a proper image pyramid as in the work proposed by Nah *et al.* [54]. An image pyramid allows to tackle different levels of blur in a coarse-to-fine manner.

		Clip 1		Clip 2	
Time expansion	Method	PSNR	SSIM	PSNR	SSIM
$r/16 \rightarrow r/8$	*doing nothing*	41.524	0.9961	39.076	0.9959
	Ours (f)	**42.9883**	**0.9965**	**40.443**	**0.9961**
$r/8 \rightarrow r/4$	*doing nothing*	40.236	**0.9936**	37.614	**0.9932**
	Ours (f^2)	**40.272**	0.9933	**38.038**	0.9928
$r/4 \rightarrow r/2$	*doing nothing*	**39.448**	**0.9920**	**36.818**	**0.9913**
	Ours (f^3)	38.512	0.9904	36.590	0.9899
$r/2 \rightarrow r$	*doing nothing*	38.929	0.9905	36.413	0.9897
	Ours (f^4)	37.193	0.9874	35.591	0.9871
$r/15 \rightarrow r$	DeblurGAN-v2+SloMo	27.900	0.9222	28.332	0.9323
$r/20 \rightarrow r$	Jin *et al.*	**39.822**	**0.9915**	**37.448**	**0.9902**

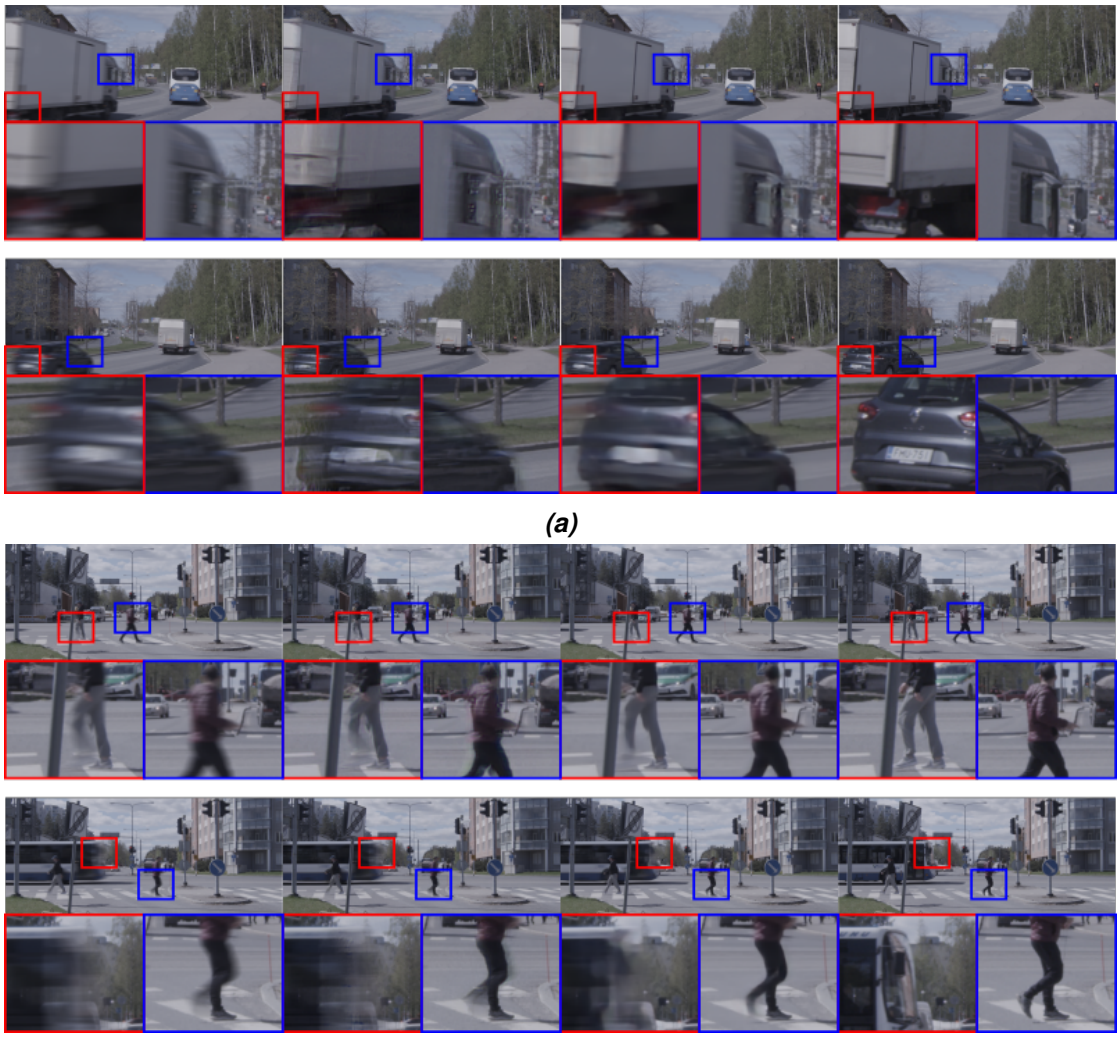
Table 4.4. *Method comparison. Whereas the original video with frame rate r is recovered by recursion in our methodology, DeblurGan-v2+SloMo and Jin et al. are fixed for a time expansion of x15 and x20, respectively.*

4.4 Joint deblurring and frame interpolation

Our VTSR method is ultimately compared to recent methodologies for joint deblurring and frame interpolation. In this experiment, we utilize the two-video dataset HuaweiRED for evaluation, exclusively. Since those videos have a very high temporal resolution (300 fps), averaging by 8 frames does not create significant blur for the input frames of our system. Therefore, we use instead 16 consecutive frames as inputs to our system, *i.e.*, we are required to apply f^4 to recover the original frame rate r. Furthermore, we use the model that was trained with the *reconstruction* training scheme for testing. As competitors, two methods are considered:

1. The naive combination of deblurring and frame interpolation. In particular, we use the DeblurGAN-v2 [10] for deblurring and SloMo [11] for frame interpolation. We refer to this method as DeblurGAN-v2+SloMo. Bering in mind that DeblurGAN-v2 outputs the middle frame from the averaged input, an odd number for averaging is more appropriate. Specifically, we use 15 frames. Then, SloMo [11] is used to interpolate the missing 14 frames. With this methodology an input video sequence with frame rate $r/15$ is recovered to the initial one r.
2. The system proposed by Jin *et al.* [13] that is explicitly designed for joint deblurring and frame interpolation. This method only allows to expand the temporal axes x10 or x20. We therefore use x20 because is closer to the time expansion of our method (x16). Thus, we average 20 consecutive frames to generate the corresponding inputs.

Noteworthy, any of the itemized methods recovers the original frame rate at a single



(a)

(b)

Figure 4.3. *Visual examples on HuaweiRED videos. From left to right: doing nothing, Ours (f^4), Jin et al. , ground-truths. (a) Clip 1, (b) Clip 2*

step, in contrast to our method that needs recursion. To assess the effectiveness of the recursion, we compare also the performance of *doing nothing*. To be precise, by *doing nothing* we mean to duplicate the input frame every recursive step. In this way, we can see how well our method behaves compared to the simple solution of repeating frames.

Table 4.4 condenses the performance metrics we obtained for the aforementioned methodologies. Firstly, our method just surpasses the performance of *doing nothing* in the two first recursive steps. However, the performance of our method drops in the next recursions, turning *doing nothing* to be quantitatively better. In second place, comparing against the single-step solution for joint deblurring and frame interpolation, we observe that our method overcomes with good margin the naive solution of DeblurGan-v2+SloMo. Conversely, the method by Jin *et al.* outperforms ours as well. In fact, Jin *et al.* achieves the best metrics in the recovery of the original videos. Anyhow, since our methodology is trained to double the input frame rate, VTSR is versatile, in theory, to many different time expansions.

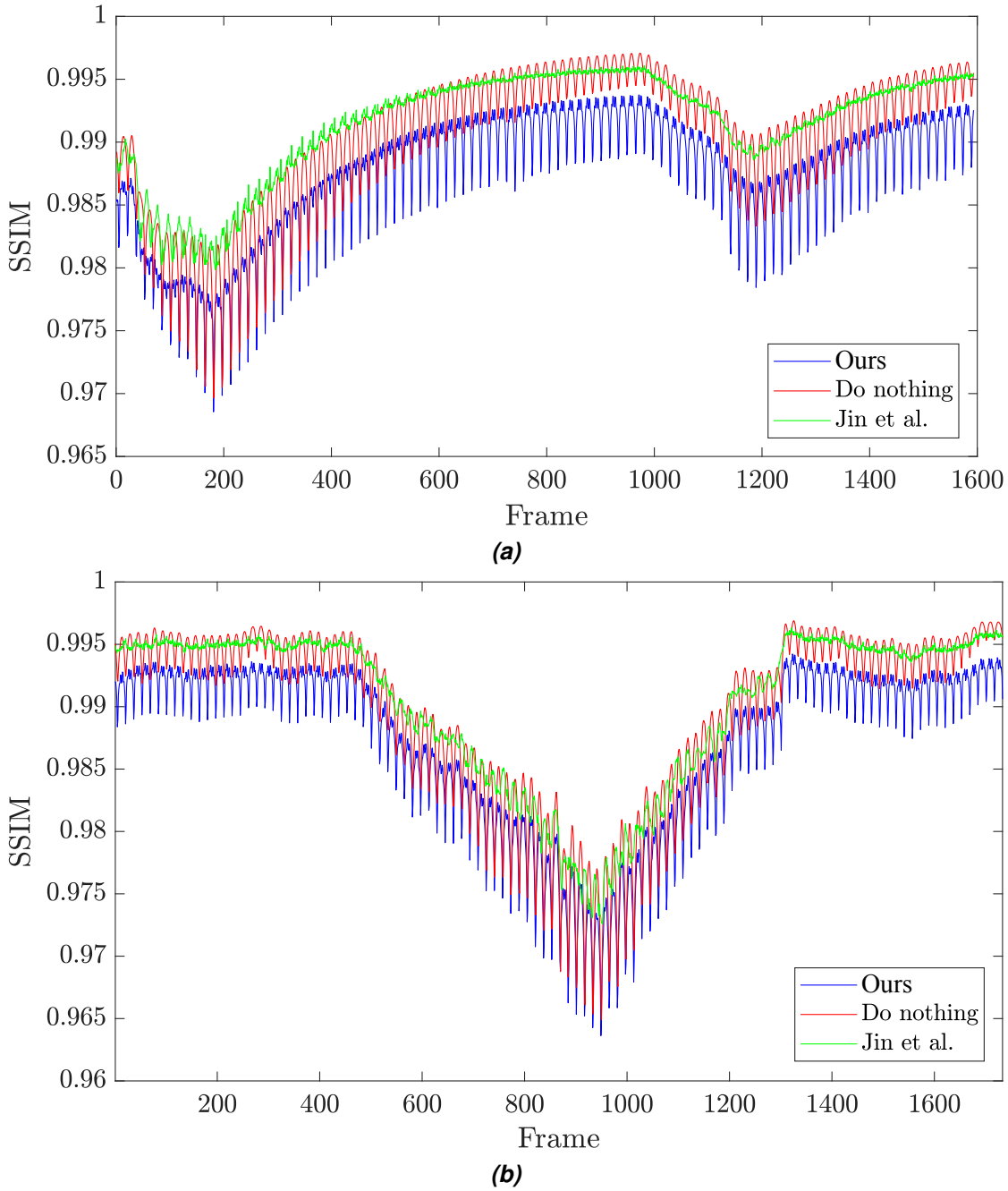


Figure 4.4. *Frame-wise performance on HuaweiRED videos. (a) Clip 1, (b) Clip 2*

Surprisingly, the PSNR and SSIM results in Table 4.4 are much higher compared to the performances presented in the other experiments. By inspection on visual results, we realize that a larger proportion of the scene remains static meanwhile few dynamic elements are corrupting the video. That might explain why *doing nothing* seems to be better than the proposed method. Figure 4.3 exhibits some examples of the recovered video at frame rate r from the two clips in the dataset. Specifically, it depicts results for *doing nothing*, our methodology, and Jin *et al.* – compared to the ground-truths. Examining the zoom areas, more clear distortions are seen for our method. Yet one may think it looks sharper than *doing nothing*. Consequently, PSNR and SSIM could not be the best metrics for evaluation. Perhaps, by masking the corrupting elements, it is still possible to

use then but only under the masked regions. On the other side, Jin *et al.*, again, manifests better qualitative results though the blur of the dynamic objects is quite strong to accomplish perfect reconstruction.

By reproducing the video results, we observe a flickering effect every certain time. With the aim of analyzing such issue, we construct a plot of the frame-wise SSIM per each clip, which is depicted in Figure 4.4. Indeed, an oscillation pattern was found for *doing nothing* and our method, whose minima match to each other. To be specific, this happens every 16 frames that corresponds to the number of frames that has been used for averaging in the construction of the inputs. We believe that the perceptual loss, which is the bigger responsible for the sharp appearance, makes borders to be attached at the moment they are more visible and does not allow them to move smoothly. Despite adding a reconstruction loss to promote temporal consistency, it seems this regularization is not enough to produce temporal smoothness. One solution to this might be to come up with a stronger loss that guarantees a smooth behaviour along the time. Also, the introduction of recurrent neural units in the network architecture would let the system have a memory, but such models are harder to train. Another aspect that is probably causing weakness in our method is that averaging 16 frames may yield to stronger local blur than the one presented during training. Although the recursive mechanism is intended to progressively expand the time until even the fastest motion is frozen and the frames become spatially sharp. In practice, there might be limitations in the number of recursive steps due to the artifacts that are highlighted every step.

Conversely, Jin *et al.* does not reveal a high oscillation pattern as shown in Figure 4.4. Anyhow, it is impacting that at some instances *doing nothing* overcomes Jin *et al.*, although the last one is undoubtedly better by visual inspection. This is another reason that makes us believe PSNR and SSIM are not the best performance metrics for this task. A final comment about those plots is that the substantial declining trends are caused by the appearance of corrupting elements in the scene – objects moving. Otherwise, the performance would be perfect as SSIM≈ 1.

5 CONCLUSION

In this thesis, a deep-learning-based methodology was proposed for the problem of video temporal super-resolution. In spite of the fact that temporal super-resolution has been previously studied, this approach had not been considered in recent deep learning solutions to extract slow-motion videos from blurry ones. We demonstrated that the recursive application of the proposed method, which expands the time by a factor of 2, gradually reduces the blur. Notwithstanding, the appearance of artifacts can limit, in practice, its use. There is uncertainty of how many times the recursive mechanism can be applied until artifacts exploit in the resulting video.

We also found that this method suits better for local blur produced generally by dynamic objects rather than strong global blur caused by substantial camera movement. A downside of this method is the lack of temporal smoothness that generates a flickering effect in the video results. To solve this issue, we incorporated a temporal consistency regularizer in the training phase of our network, but further regularization is still needed to ensure smoothness. In our experiments for the task of joint deblurring and frame interpolation, the method by Jin *et al.* [13] overweight any other method. Interestingly enough, our deep learning system is never trained, explicitly, for such a task. Instead, our network aims at super-resolve the temporal domain, yet our method inherently tackles the deblurring and frame interpolation problem. Furthermore, a miss-match between quantitative and visual results was observed in some cases. Thus, we argue that an evaluation based of PNSR and SSIM might not be appropriate for the problem of temporal super-resolution.

REFERENCES

[1] Apple. *iPhone X - Technical Specifications*. URL: https://support.apple.com/kb/SP770?viewlocale=en%7B%5C_%7DUS%7B%5C&%7Dlocale=es%7B%5C_%7DES.

[2] Gsmarena. *Sony Ericsson C510 - Full phone specifications*. URL: http://www.gsmarena.com/new%7B%5C_%7Dfrom%7B%5C%%7D3Cbr%20/sony%7B%5C_%7Dericsson%7B%5C_%7Dc510-2640.php.

[3] Kupyn, O., Budzan, V., Mykhailych, M., Mishkin, D. and Matas, J. Deblurgan: Blind motion deblurring using conditional adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 8183–8192.

[4] Kotera, J., Rozumnyi, D., Šroubek, F. and Matas, J. Intra-frame Object Tracking by Deblatting. *arXiv preprint arXiv:1905.03633* (2019).

[5] Park, S. C., Park, M. K. and Kang, M. G. Super-resolution image reconstruction: a technical overview. *IEEE signal processing magazine* 20.3 (2003), 21–36.

[6] Shimano, M., Okabe, T., Sato, I. and Sato, Y. Video temporal super-resolution based on self-similarity. *Asian Conference on Computer Vision*. Springer. 2010, 93–106.

[7] Shechtman, E., Caspi, Y. and Irani, M. Space-time super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.4 (2005), 531–545.

[8] Mudenagudi, U., Banerjee, S. and Kalra, P. K. Space-time super-resolution using graph-cut optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2010), 995–1008.

[9] Wang, X., Chan, K. C., Yu, K., Dong, C. and Change Loy, C. Edvr: Video restoration with enhanced deformable convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019, 0–0.

[10] Kupyn, O., Martyniuk, T., Wu, J. and Wang, Z. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. *Proceedings of the IEEE International Conference on Computer Vision*. 2019, 8878–8887.

[11] Jiang, H., Sun, D., Jampani, V., Yang, M.-H., Learned-Miller, E. and Kautz, J. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 9000–9008.

[12] Niklaus, S., Mai, L. and Liu, F. Video frame interpolation via adaptive separable convolution. *Proceedings of the IEEE International Conference on Computer Vision*. 2017, 261–270.

[13] Jin, M., Hu, Z. and Favaro, P. Learning to Extract Flawless Slow Motion From Blurry Videos. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, 8112–8121.

[14] Nyström, D. Colorimetric and multispectral image acquisition. PhD thesis. Institutionen för teknik och naturvetenskap, 2006.

[15] Bayer, B. E. *Color imaging array*. US Patent 3,971,065. July 1976.

[16] Boracchi, G. and Foi, A. Modeling the performance of image restoration from motion blur. *IEEE Transactions on Image Processing* 21.8 (2012), 3502–3517.

[17] Ramanath, R., Snyder, W. E., Yoo, Y. and Drew, M. S. Color image processing pipeline. *IEEE Signal Processing Magazine* 22.1 (2005), 34–43.

[18] Nikkanen, J. Computational color constancy in mobile imaging. PhD thesis. Ph. D. dissertation, Tampere University of Technology, 2013.

[19] Shahar, O., Faktor, A. and Irani, M. Space-time super-resolution from a single video. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* IEEE Computer Society, 2011, 3353–3360. ISBN: 9781457703942. DOI: 10.1109/CVPR.2011.5995360.

[20] Nah, S., Son, S., Timofte, R., Lee, K. M., Siyao, L., Pan, Z., Xu, X., Sun, W., Choi, M., Kim, H. et al. AIM 2019 Challenge on video temporal super-resolution: methods and results. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE. 2019, 3388–3398.

[21] Park, B., Yu, S. and Jeong, J. Robust Temporal Super-Resolution for Dynamic Motion Videos. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE. 2019, 3494–3502.

[22] Glasner, D., Bagon, S. and Irani, M. Super-resolution from a single image. *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, 349–356.

[23] Maggioni, M. and Dragotti, P. L. Video temporal super-resolution using nonlocal registration and self-similarity. *2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE. 2016, 1–6.

[24] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. MIT press, 2016.

[25] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors. *nature* 323.6088 (1986), 533–536.

[26] Hinton, G. E. et al. Learning distributed representations of concepts. *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. Amherst, MA. 1986, 12.

[27] LeCun, Y. et al. Generalization and network design strategies. *Connectionism in perspective* 19 (1989), 143–155.

[28] Krizhevsky, A., Sutskever, I. and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, 1097–1105.

[29] Girshick, R., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, 580–587.

[30] Sun, Y., Chen, Y., Wang, X. and Tang, X. Deep learning face representation by joint identification-verification. *Advances in neural information processing systems*. 2014, 1988–1996.

[31] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, 248–255.

[32] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[33] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[34] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[35] He, K., Zhang, X., Ren, S. and Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778.

[36] Dong, C., Loy, C. C., He, K. and Tang, X. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2015), 295–307.

[37] Zhang, K., Zuo, W., Chen, Y., Meng, D. and Zhang, L. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing* 26.7 (2017), 3142–3155.

[38] Jain, V. and Seung, S. Natural image denoising with convolutional networks. *Advances in neural information processing systems*. 2009, 769–776.

[39] Kim, J., Kwon Lee, J. and Mu Lee, K. Accurate image super-resolution using very deep convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 1646–1654.

[40] Ronneberger, O., Fischer, P. and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, 234–241.

[41] Mao, X., Shen, C. and Yang, Y.-B. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in Neural Information Processing Systems*. 2016, 2802–2810.

[42] Su, S., Delbracio, M., Wang, J., Sapiro, G., Heidrich, W. and Wang, O. Deep Video Deblurring for Hand-held Cameras. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, 1279–1288.

[43] Mustaniemi, J., Kannala, J., Matas, J., Särkkä, S. and Heikkilä, J. LSD $\_2$-Joint Denoising and Deblurring of Short and Long Exposure Images with Convolutional Neural Networks. *arXiv preprint arXiv:1811.09485* (2018).

[44] Isola, P., Zhu, J.-Y., Zhou, T. and Efros, A. A. Image-to-image translation with conditional adversarial networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, 1125–1134.

[45] Dumoulin, V. and Visin, F. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).

[46] Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. *European conference on computer vision*. Springer. 2014, 818–833.

[47] Scherer, D., Müller, A. and Behnke, S. Evaluation of pooling operations in convolutional architectures for object recognition. *International conference on artificial neural networks*. Springer. 2010, 92–101.

[48] Wilson, D. R. and Martinez, T. R. The general inefficiency of batch training for gradient descent learning. *Neural networks* 16.10 (2003), 1429–1451.

[49] PyTorch. URL: https://pytorch.org/.

[50] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, 4510–4520.

[51] Tian, Y., Zhang, Y., Fu, Y. and Xu, C. TDAN: Temporally deformable alignment network for video super-resolution. *arXiv preprint arXiv:1812.02898* (2018).

[52] Jaderberg, M., Simonyan, K., Zisserman, A. et al. Spatial transformer networks. *Advances in neural information processing systems*. 2015, 2017–2025.

[53] Wang, Z., Bovik, A. C., Sheikh, H. R. and Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13.4 (2004), 600–612.

[54] Nah, S., Kim, T. H. and Lee, K. M. Deep Multi-Scale Convolutional Neural Network for Dynamic Scene Deblurring. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[55] FFmpeg. URL: https://ffmpeg.org/.

[56] RED. *Use REDLINE*. URL: http://docs.red.com/955-0004_v42/REV-A/HTML/955-0004_V42%5C%20Rev-A%5C%20%5C%20%5C%20RED%5C%20PS,%5C%20REDCINE-X%5C%20PRO%5C%20Operation%5C%20Guide/Content/11_REDLINE/1_Intro_REDLINE.htm.

[57] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H. and Wei, Y. Deformable convolutional networks. *Proceedings of the IEEE international conference on computer vision*. 2017, 764–773.