Video Super-Resolution via Dynamic Local Filter Network

by

Yang Zhou

Thesis submitted to the University of Ottawa
in Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Video super-resolution (VSR) aims to give a satisfying estimation of a high-resolution (HR) image from multiple similar low-resolution (LR) images by exploiting their hidden redundancy. The rapid development of convolutional neural network (CNN) techniques provide numerous new possibilities to solve the VSR problem. Recent VSR methods combine CNN with motion compensation to cancel the inconsistencies among the LR images and merge them to an HR images. To compensate the motion, pixels in input frames are warped according to optical-flow-like information. In this procedure, trade-off has to be made between the distraction caused by spatio-temporal inconsistencies and the pixel-wise detail damage caused by the compensation.

We proposed a novel VSR method with the name, Video Super-Resolution via Dynamic Local Filter Network, and its upgraded edition, Video Super-Resolution with Compensation in Feature Extraction. Both methods perform motion compensation via a dynamic local filter network, which processes the input images with dynamically generated filter kernels. These kernels are sample-specific and position-specific. Therefore, our proposed methods can eliminate the inter-frame differences during feature extractions without explicitly manipulating pixels. The experimental results demonstrate that our methods outperform the state-of-the-art VSR algorithms in terms of PSNR and SSIM and recover more details with superior visual quality.

Acknowledgements

I would like to owe my deepest gratitude to my supervisor, Professor Jiying Zhao, for the patient guidance, encouragement and advice he has provided throughout my time as his student. The door to his office was always open whenever I ran into a trouble spot or had a question about my research or writing. It is extremely lucky for me to have a supervisor who cared so much about my work.

I would also like to offer my best regards and blessing to Lei Chen, Xiaohong Liu and all of my colleagues in our lab who supported me during my study here. This accomplishment would not have been possible without them.

Dedication

This thesis work is dedicated to my wife who has been a constant source of support and encouragement during all of the ups and downs of my research and life. I am truly thankful for having you in my life. This work is also dedicated to my parents who have always loved me unconditionally and given me confidence to achieve what I want.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AdaGrad	Adaptive gradient algorithm
Adam	Adaptive moment estimation algorithm
BayesSR	A Bayesian approach to adaptive video super resolution
CNN	Convolutional neural network
EDSR	Enhanced deep super-resolution network
ELU	Exponential linear unit
ESPCN	Efficient sub-pixel convolutional neural network
GAN	Generative adversarial network
HR	High-resolution
LR	Low-resolution
MDSR	Multi-scale deep super-resolution network
MSE	Mean absolute error
PSNR	Peak signal-to-noise ratio
ReLU	Rectified linear unit
ResNet	Residual network
RMSProp	Root mean square propagation algorithm
SGD	Stochastic gradient decent
SISR	Single image super-resolution
SPMC	Sub-pixel motion compensation
SR	Super-resolution
SRCNN	Super-resolution convolutional neural network

SRResNet	Super-resolution residual network
SRGAN	Super-resolution generative adversarial network
SSIM	Structural similarity
STN	Spatial transform network
VESPCN	Video efficient sub-pixel convolutional neural network
VSR	Video super-resolution
VSRnet	Video super-resolution with convolutional neural networks

Chapter 1

Introduction

Image super-resolution (SR) refers to the technique that recovers a high-resolution (HR) image from low-resolution (LR) images without changing the hardware sensors. As a long-standing fundamental research topic in image processing field, it has wide applications in medical imaging [26] [52] [62] [63] [64], satellite imaging [15] [34] [41] [59] [60], surveillance [10] [27] [32] [43] [71] , etc. It can also be used to facilitate image/video enhancement [5] [16] [17] [53] and text/object recognition [8] [29] [45]. There are mainly two branches: single image super-resolution (SISR) and video super-resolution (VSR). We will discuss them separately in the following sections.

1.1 Single image super-resolution

The SISR estimates the HR image from a single LR image. It can be viewed as a sophisticated improvement of traditional image interpolation methods. Traditional interpolation methods, like the nearest neighbor, bilinear and bicubic, usually re-sample the images with simple averaging algorithms to ensure that the up-sampled images have smooth edges, as shown in Figure 1.1. However high frequency details of original images are lost in this process. The interpolated image often looks blurry.

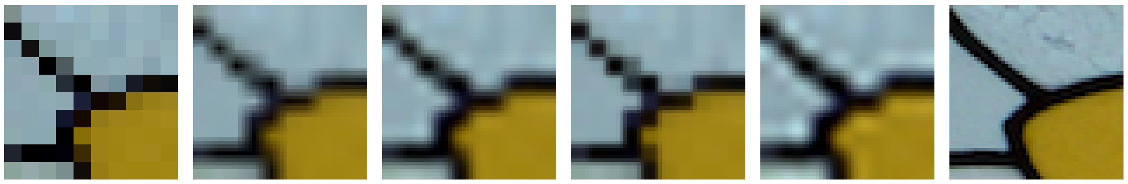


Figure 1.1: Various image interpolation methods.

As most of the information has been discarded in the down-sampling process, it is impossible to recover the original HR image perfectly. However, there are still left resources which SISR can exploit to reconstruct the HR image much more accurately than plain interpolations. While traditional SISR algorithms [14] [22] [23] [30] [68] usually focus on inherent similarities of the LR images, machine learning based methods [18] [19] [35] [54] [66] comprehend high-level knowledge by inspecting large amount of external examples and use this knowledge to super-resolve target images. Figure 1.2 illustatres comparisons between the bicubic interpolation and the enhanced deep super-resolution network (EDSR) [42], one of the state-of-the-art SISR methods. The images are from the original paper of EDSR.

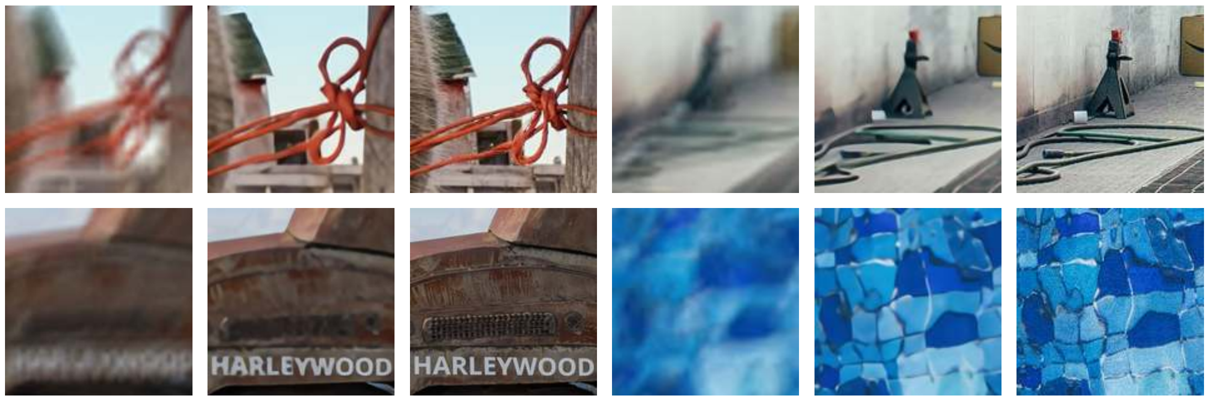


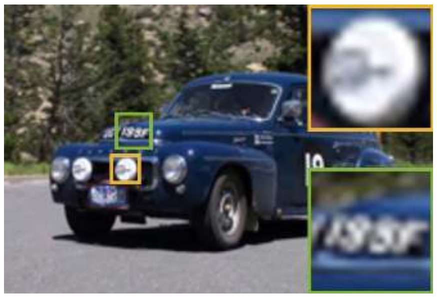
Figure 1.2: Comparison between bicubic interpolation and EDSR [42].

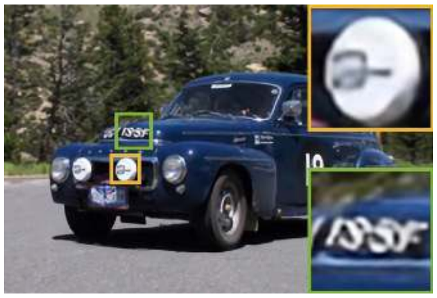
1.2 Video super-resolution

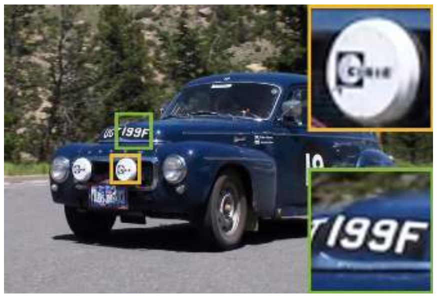
VSR problem assumes multiple LR observations of an HR image are available in the form of video frames. Abundant explicit redundancy existing in the frames can be naturally utilized to construct HR images. Despite the redundancy is worthy of exploitation, how to combine and assemble this information has become a key challenge, which limits extensive explorations of VSR.

One of the main obstacles is the inter-frame inconsistencies which are caused by the emergence, vanishing and transformation of the various objects in the frames. Recent methods mostly employ motion compensation to address this issue. For the LR frames are assumed to be consecutive and aligned along the time dimension, it is possible to estimate the motion of objects, according to which transformations can be applied to reduce the inconsistencies. Then the compensated frames can be easily combined into an HR image. Figure 1.3 shows the experimental results of a state-of-the-art method, detail-revealing deep video super-resolution. The result in Figure 1.3(d) is from 3 consecutive frames, representing VSR. Because the neural network is trained to accept exact 3 frames as input, 3 identical images are fed to the network to simulate the behavior of SISR for comparison. The result is shown in Figure 1.3(b). It is obvious that the VSR generates more accurate results than the SISR.

Neural network technique, especially the convolutional neural network, has achieved a rapid development in recent years. It has provided numerous new possibilities and made significant performance improvements to the VSR. The VSRnet proposed by Kappeler *et al.* [33] uses optical flow technique to estimate the motion information among the input frames. The frames are then warped according to the motion information and fed to an SRCNN [19] inspired network for super-resolving. The video efficient sub-pixel convolutional neural network (VESPCN) proposed by Caballero *et al.* [7] uses a motion compensation module, which is inspired by the spatial transform network (STN) [31], to replace the optical flow


(a) Bicubic


(b) SR with 3 identical frames


(c) Ground Truth

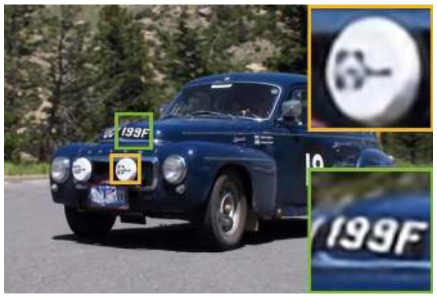

(d) SR with 3 consecutive frames

Figure 1.3: Detail-revealing deep video super-resolution.

calculating procedure. In [54], an efficient sub-pixel convolutional neural network (ESPCN) based module is used for super-resolving. Tao *et al.* [58] improved the VESPCN with a sub-pixel motion compensation (SPMC) layer, which combines motion compensation and sub-pixel up-sampling into one operation. Detailed review on these works will be given in Chapter 3.

1.3 Thesis contributions

In our proposed methods, there are two contributions which improve the performance and efficiency of the VSR task.

The first contribution is introducing dynamic local filter network for motion compensation in VSR. To our best knowledge, this is the first time that the dynamic local filter network is applied for VSR problem. Traditional CNN based VSR methods rely on pixel manipulation based motion compensation which might damage the original pixel information and make SR results suboptimal. We propose to use dynamic local filter networks for estimating and compensating the motion among video frames. Two novel VSR networks are proposed based on this idea and experimental results demonstrate that the proposed networks outperform the state-of-the-art methods.

The second contribution is introducing multi-scale up-sampling structure to VSR for the first time. Traditional VSR networks are separately trained for different scale factors, which unnecessarily consumes large amount of time for redundant training and storage space. In the proposed VSR network, we introduce a multi-scale up-sampling structure which utilizes the inter-scale correlation and super-resolves images at multiple scale factors simultaneously. A considerable time and space can be saved.

1.4 Thesis structure

We focus on video super-resolution in this thesis and proposed two novel methods: video super-resolution via dynamic local filter network (DLVSR) and video super-resolution with compensation in feature extraction (VSR-CFE). Both methods share the same inspiration which introduces dynamic filter network with locally-connected layers to VSR. The rest of this thesis are organized as follows. Chapter 2 introduces the basic concepts of neural network and some SR-related components. Chapter 3 illustrates three SISR methods and three VSR methods in detail. Chapter 4 presents the proposed SR methods. Chapter 5 gives our experimental results and Chapter 6 concludes this thesis.

Chapter 2

Related concepts

2.1 Quality evaluation

To estimate the performance of algorithms, quality evaluation is an essential procedure in image processing area. There are many approaches proposed to compare the processed images with the ground truth. In the research of SR, the most commonly used quality evaluation metrics are Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM), by which, the super-resolved HR images are compared against the ground truth.

First, the mean squared error (MSE) is formulated as

$$\mathrm{MSE}(I_s, I_g) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I_g(i,j) - I_s(i,j)]^2, \tag{2.1}$$

where I_s is the super-resolved image, I_g is the corresponding ground truth. Both of the images are of size $M \times N$. Based on the MSE, PSNR can be defined as

$$\mathrm{PSNR}(I_s, I_g) = 10 \cdot log_{10} \left[\frac{(\mathrm{MAX} - \mathrm{MIN})^2}{\mathrm{MSE}(I_s, I_g)} \right], \tag{2.2}$$

where MAX and MIN are the maximum and minimum possible pixel values of the image respectively. For 8-bit gray-scale images, the MAX is $2^8 - 1$ which equals to 255 and the MIN is 0. In the scenario of image SR, the related images are usually colored and represented in RGB or YCbCr color space. For an RGB image, the PSNR is measured on R, G, B channel individually and averaged afterwards. For a YCbCr image, due to most of its information has been concentrated in the luminance (Y) channel, PSNR is usually only applied on the luminance channel. The larger the PSNR is, the better the image is recovered.

Although PSNR generally indicates the quality of images, it is also known to perform poorly when there are differences in translation, contrast, brightness or other factors between the measured image and the ground truth. SSIM proposed by Wang *et al.* [65] has been proved to be more consistent with human visual perception than PSNR. It is a combined comparison on the luminance, contrast and structure differences between two images and can be formulated as

$$\mathrm{SSIM}(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}, \tag{2.3}$$

where α, β, γ are weights. The $l(x,y)$, $c(x,y)$, (x,y) are the comparison measurements on luminance, contrast and structure respectively, and are defined as

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}, \tag{2.4}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}, \tag{2.5}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}, \tag{2.6}$$

where μ_x and μ_y are the average of x and y respectively. σ_x^2 and σ_y^2 are the variance of x and y respectively. σ_{xy} is the covariance of x and y. $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$, $c_3 = \frac{1}{2}c_2$ are variables to stabilize the division with weak denominator, where L is the dynamic range of the pixel-values, $k_1 = 0.01$ and $k_2 = 0.03$ by default. SSIM is usually only taken on luminance channel. The larger the SSIM is, the better the image is super-resolved.

2.2 Neural networks

Neural networks refer to a family of computer programs with biological neural network inspired architectures. As the quintessential deep learning models, the neural networks aim to approximate some function $f(\cdot)$. For instance, an image SR function can be defined as

$$I_{SR} = f(I_{LR}; \theta), \tag{2.7}$$

where I_{LR} is the low-resolution image and I_{SR} is the super-resolved image. The parameter θ is learned to be the value which results in the best function approximation. With properly designed architectures and enough training data, neural networks can learn to approximate complex function mappings.

2.2.1 Neuron

Neurons are elementary units in a neural network. They are mathematical functions which receive one or more inputs x_i and produce an output y. Usually, each x_i is assigned with a corresponding weight w_i. Then the weighted sum of all inputs, added with an optional bias b, is passed to an activation function $f()$, which performs a non-linear transformation to generate the output. The output is sometimes called activation as well. The behavior of a neuron with k inputs can be formulated as

$$y = f\left(\sum_{i=1}^{k} w_i x_i + b\right). \tag{2.8}$$

An illustration of a neuron is given in Figure 2.1. In addition, when the inputs and the output are represented as vectors, the neuron can also be defined as

$$y = f\left(\mathbf{w}^\top \mathbf{x} + b\right). \tag{2.9}$$

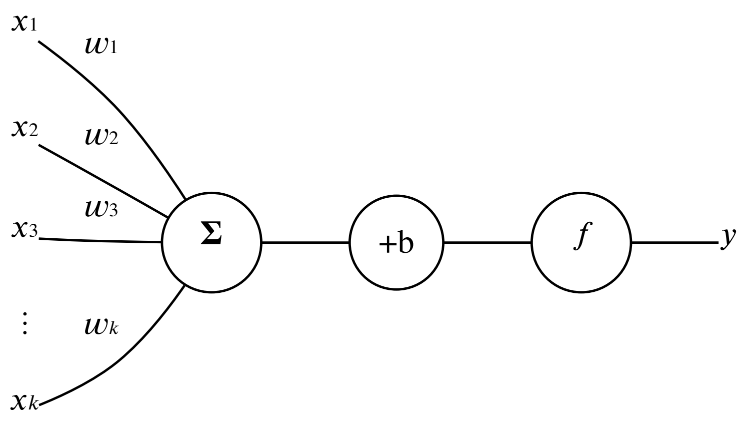


Figure 2.1: A basic neuron with k inputs.

2.2.2 Neural network

The output generated by a neuron can be passed to multiple other neurons as an input. Multiple neurons can thus be connected together to form a network. From the functional view, neuron connection is equivalent to function composition and a neural network is a function composited by smaller functions. Inside the network, neurons are aligned in hierarchy which is referred as layers in neural network context. The layers are functions as well. For example, there might be three layers cascaded together, denoted as $f_1(\cdot)$, $f_2(\cdot)$ and $f_3(\cdot)$. The equivalent function of the network $f(x)$ can be formulated as

$$f(x) = f_3(f_2(f_1(x))). \tag{2.10}$$

This chain-like structure is the most commonly used architecture of neural networks. The neurons in the first layer, which is the $f_1(\cdot)$ in this case, receive inputs directly from the inputs of the network. Therefore the first layer is also called input layer. The layer $f_2(\cdot)$ is named the second layer intuitively and the last layer $f_3(\cdot)$ is the third layer, also called the output layer. The neurons of the output layer generate outputs as the output of the whole network. The layers between the input layer and the output layer, like $f_2(\cdot)$, are called hidden layers. The number of layers gives the depth of a network. It is this terminology from which the name "deep learning" comes. Figure 2.2 illustrates a 4-layer neural network with 2 hidden layers.

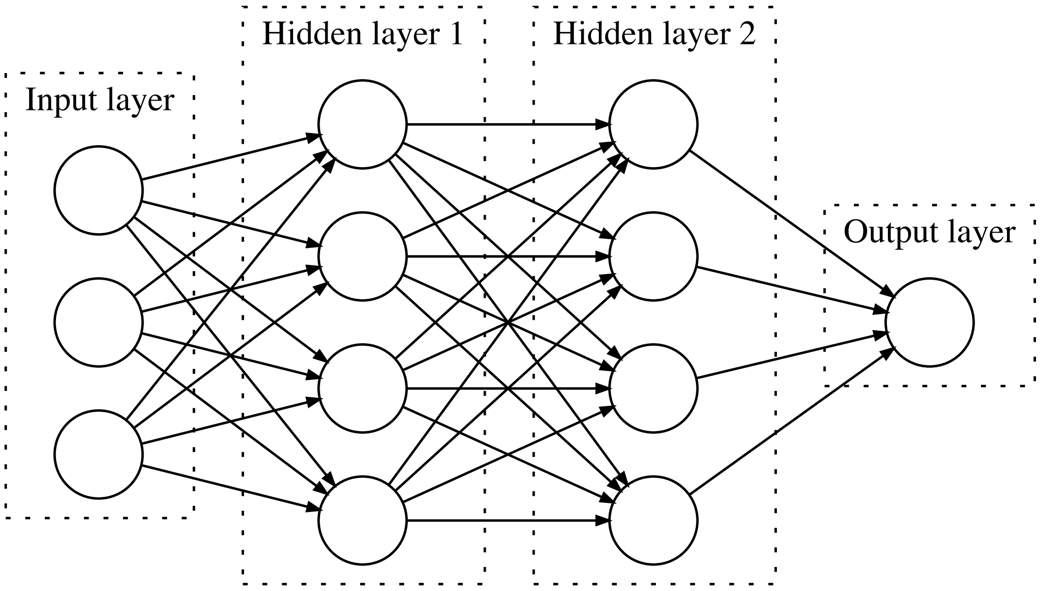


Figure 2.2: A 4-layer neural network.

2.2.3 Activation function

Assuming we have a 3-layer neural network formulated as $f(x) = f_3(f_2(f_1(x)))$, if $f_1(\cdot)$,$f_2(\cdot)$ and $f_3(\cdot)$ were all linear functions, then the whole network $f(x)$ would remain a linear function, which makes the composition meaningless. To approximate more complex functions, non-linearity must be introduced. As we mentioned in Section 2.2.1, the weighted sum of the inputs of a neuron is passed through an activation function. The activation function plays an important role in neural network, because it brings the non-linearity.

Quite an amount of activation functions have been invented. In this section, we will give a brief introduce to three of the most commonly used activation functions: sigmoid, tanh and ReLU.

Sigmoid function. A sigmoid function, also called a logistic function or a logistic curve, is a "S" shape function which has applications in a wide range of fields, including probability, machine learning, statistics and so on. In the context of neural network, it is usually defined as

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}. \tag{2.11}$$

The corresponding graph is illustrated in Figure 2.3. The function takes a real number input x in the range of $[-\infty, \infty]$ and outputs a real number value between 0 and 1. Besides introducing non-linearity, the sigmoid function is used to clamp outputs to within $[0, 1]$ as well. For example, when a neural network is supposed to work as a binary classifier, its target class is often denoted as 0 or 1. Hence a sigmoid activation function will probably be used on the last layer to ensure the output values fall in a valid range.

Hyperbolic tangent function. The hyperbolic tangent function, or tanh function for short, is an old mathematical function which was first used in 1774. It is defined as the ratio between the hyperbolic sine and hyperbolic cosine function, as the formulation below

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}, \tag{2.12}$$

where $\sinh(x)$ and $\cosh(x)$ are defined as

$$\sinh(x) = \frac{1-e^{-2x}}{2e^{-x}}, \tag{2.13}$$

$$\cosh(x) = \frac{1+e^{-2x}}{2e^{-x}}. \tag{2.14}$$

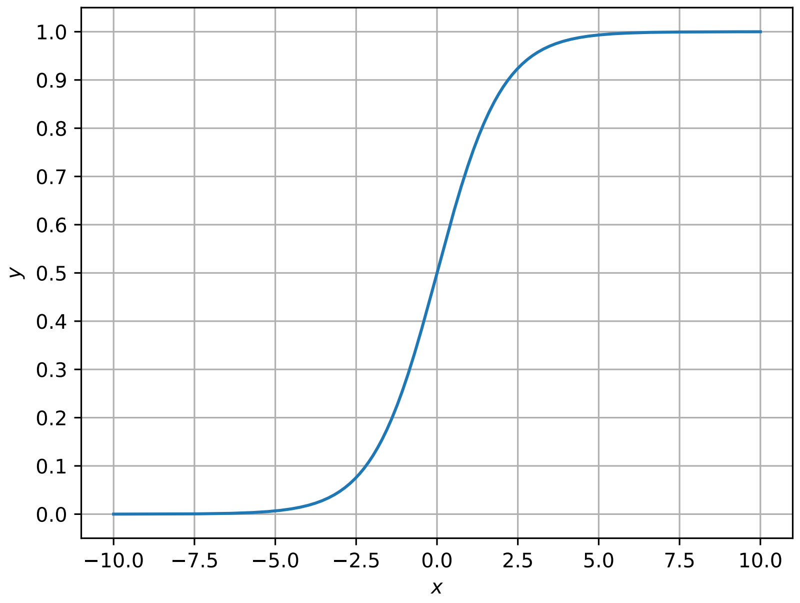


Figure 2.3: The sigmoid function.

Therefore the formulation of $\tanh(x)$ can be simplified as

$$\tanh(x) = \frac{e^x + e^{-x}}{e^x + e^{-x}}. \tag{2.15}$$

The graph is shown in Figure 2.4. Similar to the sigmoid function, the tanh function takes a real number input within $[-\infty, \infty]$, but outputs a real number value between -1 and 1. Actually, the tanh function is a rescaled sigmoid function as the equation below

$$\tanh(x) = 2 \cdot \text{sigmoid}(2x) - 1. \tag{2.16}$$

Although the tanh function and the sigmoid function seem to be interchangeable, practical differences do exist, especially when the training data is normalized.

- Because of the larger output value range than the sigmoid function, the tanh function has stronger gradients around $x = 0$ which might help network converge faster.

- The tanh function can reduce bias in the gradients due to being symmetric with respect to the origin.

Explanation in detail can be found in [70].

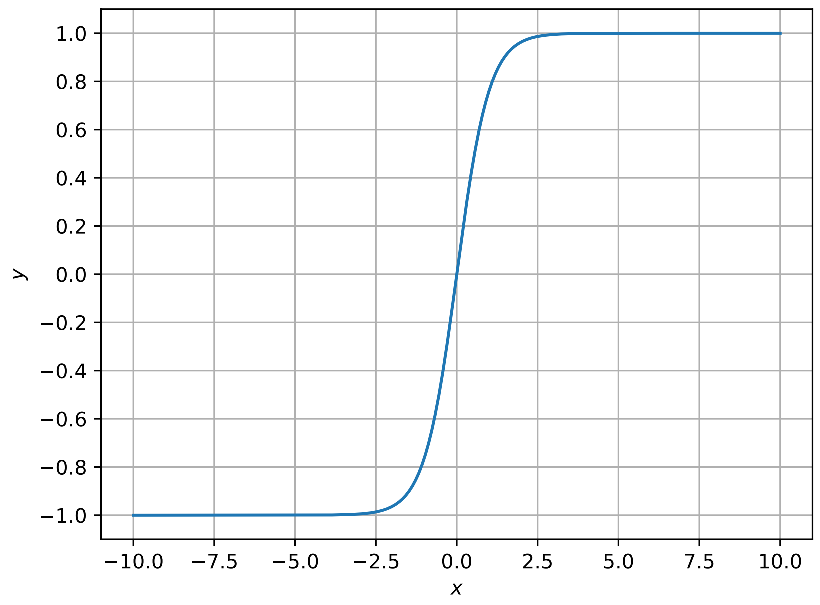


Figure 2.4: The tanh function.

Rectified linear unit. A rectified linear unit (ReLU) is defined as

$$\text{ReLU}(x) = max(0, x), \tag{2.17}$$

which means it outputs the positive part of its argument as shown in Figure 2.5. ReLU is currently the default recommended activation function for use with most neural network [48]. Although ReLU applies non-linear transformation, the function remains very close to linear ones. It can be viewed as a concatenation of two linear functions. Due to ReLU is almost linear, it has properties which make both linear and non-linear models easy to optimize. The ReLU(x) has zero gradient for $x < 0$ and is non-zero centered, non-differentiable at $x = 0$. These properties can lead to problem in some specific situations. Therefore, some variants of the ReLU are introduced.

The leaky ReLU is a variant which allows a small, non-zero gradient when the unit is

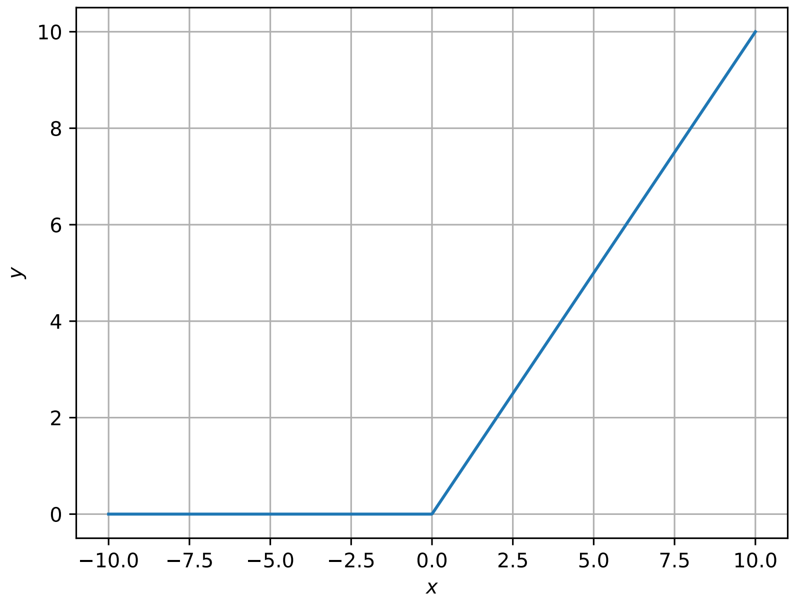


Figure 2.5: ReLU function.

not active. It is formulated as

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} . \tag{2.18}$$

Illustration can be found in Figure 2.6.

The exponential linear units (ELUs) [13] make the mean of its outputs closer to zero and have been shown to be able to obtain higher classification accuracy than the standard ReLU. The ELU is defined as

$$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{otherwise} \end{cases} , \tag{2.19}$$

where a is a positive hyper-parameter to be tuned. Its graph is shown in Figure 2.7.

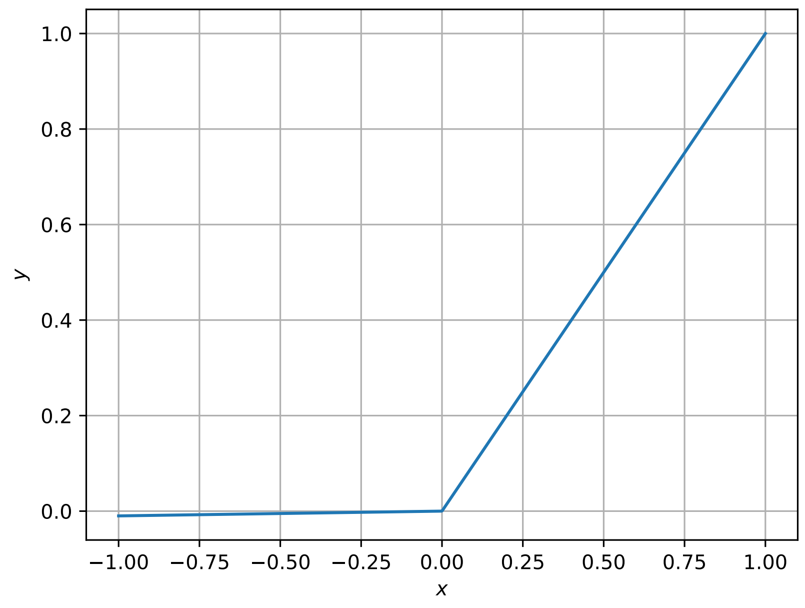


Figure 2.6: The leaky ReLU function.

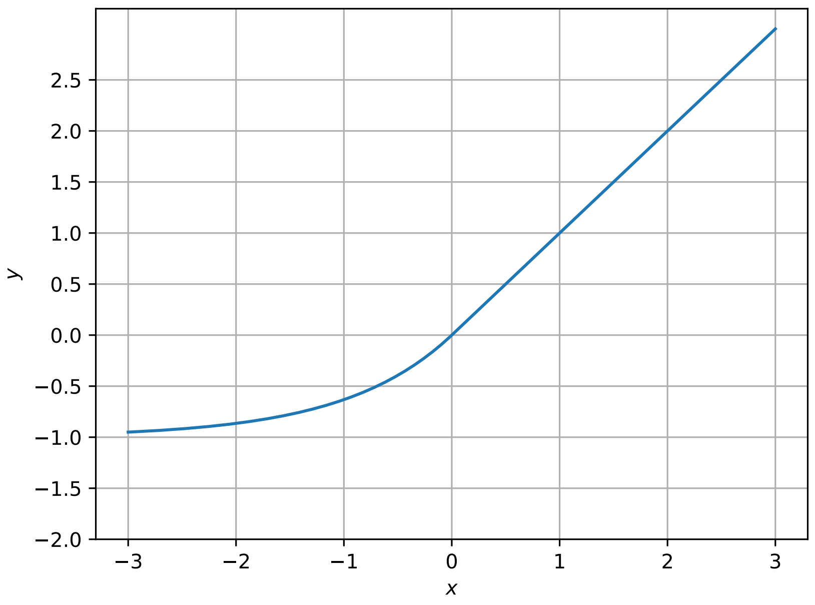


Figure 2.7: The ELU function.

2.3 Training neural networks

Neural networks is a family of machine learning models. The machine learning is defined by Mitchell [47] as

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

Machine learning methods can typically be classified into two categories: supervised learning and unsupervised learning. In supervised learning, example inputs and corresponding desired outputs (labels) are shown to the model. For unsupervised learning, no label is given. In this thesis, we only focus on supervised learning.

2.3.1 Parameter and hyper-parameter

The behavior of a machine learning model is usually parameterized by a set of tunable parameters. The procedure to train the model is to find the optimal parameters with the help of the training dataset. Besides the parameters, there are other configurations which control behaviors of the model but are not adapted by the learning process. To distinguish from the parameters, these configuration are call hyper-parameters.

2.3.2 Dataset

Machine learning models learn and make predictions on data. The data used to support this work-flow are usually split as three datasets: training, test and validation datasets. Initially, the training dataset is shown to model. Learning algorithm tunes the model parameters to make it fit to the training data. The fitted model then makes prediction on the validation dataset. The validation dataset gives an unbiased estimation on the training progress. Finally, the test dataset is used to measure the performance of the trained models.

2.3.3 Loss function

A loss function is a function that maps one network evaluation to a real number, representing a loss value associated with this specific evaluation. The network training or optimization task seeks to minimize the loss value by tuning parameters. We will introduce several commonly used loss functions in this section.

MSE and L2 loss. MSE is short for the mean squared error, which is defined as

$$\mathrm{MSE}(\hat{I}, I_{gt}) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \Big[I_{gt}(i,j) - \hat{I}(i,j) \Big]^2, \tag{2.20}$$

where $\hat{I}$ is the predicted image, I_{gt} is the ground truth. Both of them are of size $m \times n$. MSE loss is usually used when the outputs are images and the PSNR is the primary metric. L2 loss is the square of the L2 norm of the difference. In brief, it is just the sum of squared errors whose definition is

$$\mathrm{L2}(\hat{I}, I_{gt}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \Big[I_{gt}(i,j) - \hat{I}(i,j) \Big]^2. \tag{2.21}$$

However, due to pixel values are mostly normalized within $[0, 1]$, the quadratic property of MSE/L2 loss may impede the convergence speed.

MAE and L1 loss. MAE, short for the mean absolute error, is formulated as

$$\mathrm{MAE}(\hat{I}, I_{gt}) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \Big| I_{gt}(i,j) - \hat{I}(i,j) \Big|, \tag{2.22}$$

where $|\cdot|$ denotes the absolute value. Similarly, L1 loss is the sum of absolute errors, defined as

$$\mathrm{L1}(\hat{I}, I_{gt}) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left|I_{gt}(i,j) - \hat{I}(i,j)\right|. \tag{2.23}$$

Because their linear property, large errors have relatively smaller influence than MSE and L2 loss. Therefore, MAE and L1 loss is more robust to outliers.

Cross entropy. The cross entropy is a measurement of the divergence between two probability distributions in information theory. It can be used as the loss function for classification tasks. Let p_i be the true probability distribution and q_i is the predicted one. Considering a binary classification task, we assume $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$. The cross entropy loss is then defined as

$$\mathrm{CrossEntropy}(p, q) = -y\mathrm{log}\hat{y} - (1-y)\mathrm{log}(1-\hat{y}). \tag{2.24}$$

Cross entropy is the most popular loss function for classification tasks, but not commonly used for image processing problems.

2.3.4 Optimization

Optimizing a neural network aims to minimize the loss function $L(x)$. The absolute lowest value of $L(x)$ is a call global minimum. Single or multiple global minima may exist for $L(x)$. It is also possible that there are local minima which are not globally optimal. Moreover, the input of $L(x)$ is multidimensional, which means the optimization task will be quite challenging. In the context of neural network, we are usually satisfied with finding a value of $L(x)$ which is very low, but not necessarily a minimum.

Almost all neural network optimizations are powered by the stochastic gradient decent (SGD) algorithm [6]. The SGD is an extension of gradient decent algorithm. In brief, the gradient decent algorithm treats each tunable parameter as one dimension of a high-dimensional space. The graph of the loss function will be a surface in this high-dimensional space. The algorithm drives an imaginary particle to move down the loss surface from a random-initialized position according to the gradient at its position. The iterative update procedure can be formulated as

$$w := w - \eta \nabla Q_i(w), \tag{2.25}$$

where w denotes the parameters, $Q_i(w)$ is the value of loss function at i-th iteration, η is the step size of update operation and is more frequently called the learning rate. The coordinates where the particle finally settles will be the optimized parameters.

In SGD, the true gradient is approximated by a gradient of a batch of sampled data. Thus it is possible to make several passes over the training dataset. For each pass, the data can be shuffled to prevent periodic arrangements. The SGD algorithm can be presented in pseudo-code as below.

```
Input: Training data, learning rate $\eta$
Output: Model parameters $w$
Choose an initial vector of parameters $w$ and learning rate $\eta$;
repeat
    Randomly shuffle examples in the training set;
    for $= 1, 2, ..., n$ do
        $w := w - \eta \nabla Q_i(w)$;
    end
until an approximate minimum is obtained;
```

In machine learning, the learning rate of SGD has been considered problematic. Setting learning rate too high makes the training diverge; setting it too low slows down the convergence. Because of these issues, several extensions of SGD have been proposed.

SGD with momentum. SGD with momentum [57] determines the next update as a linear combination of the gradient and the previous update. The behavior can be formulated as

$$\Delta w := w + \alpha \Delta w - \eta \nabla Q_i(w). \tag{2.26}$$

The name momentum derives from an analogy to momentum in physics. The particle in the parameter space gains acceleration from the gradient. Compared with classical SGD, SGD with momentum prevents oscillations by attempting to keep traveling in the same direction.

AdaGrad. AdaGrad (short for adaptive gradient) algorithm [21] introduces per-parameter learning rate to classical SGD. This property potentially decreases the learning rate for less sparse parameters and increases the learning rate for sparse ones. For each parameter w_j, the update procedure is defined as

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j, \tag{2.27}$$

where η is a base learning rate, $G_{j,j}$ is defined as

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2, \tag{2.28}$$

where g_τ is the gradient at iteration τ.

RMSProp. RMSProp (for Root Mean Square Propagation) [61] is another per-parameter adaptive learning rate algorithm. The learning rate of a weight is divided by a running average of its recent gradients. The initial running average is defined as

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2, \tag{2.29}$$

where γ is a factor to control forgetting. The parameters are then updated as

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w). \tag{2.30}$$

Adam Adam [36] is short for adaptive moment estimation and is an update to the RMSProp method. In Adam, the running averages is calculated on both the gradients and the second moments of the gradients. For the t-th iteration, the update procedure is given by

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}, \tag{2.31}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2, \tag{2.32}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t}, \tag{2.33}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^t}, \tag{2.34}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon}, \tag{2.35}$$

where ϵ is a small number used to prevent division by 0, β_1 and β_2 are the forgetting factors. Adam is currently the default recommended optimizer for most neural networks.

2.3.5 Back-propagation

When a neural network is evaluated, it receives an input x and produces an output $\hat{y}$. The information flows forward through the input layer, hidden layers and finally the output layer. This procedure is called *forward propagation*. When the network is being trained, the outputs are fed to the loss function to produce a scalar loss value. The *back-propagation* [70] refers to the algorithm which allows the information flows from the loss value backwards through the network in order to calculate the gradient efficiently.

The back-propagation algorithm is based on the chain rule of calculus. The chain rule aims to compute the derivatives of functions composed of other functions of which derivatives are known. As an example, let f and g both be functions in real number domain and satisfy $y = g(x)$ and $z = f(g(x)) = f(y)$. The chain rule states that

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}. \tag{2.36}$$

Based on the chain rule, the back-propagation algorithm can be clearly demonstrated with the help of computational graphs. Assuming we are trying to calculate the derivatives for a function f, defined as

$$f(x, y, z) = (x + y) \cdot z.$$

The function can be expressed as a computational graph as illustrated in Figure 2.8. Assuming the training data we are using is $x = -2, y = 5, z = -4$, the forward propagation computes the results for each operation node as shown in the figure to the left. The back-propagation algorithm starts from the end and recursively computes the gradients conditioned on the property of each operation node all the way to the inputs, as shown in the figure to the right.

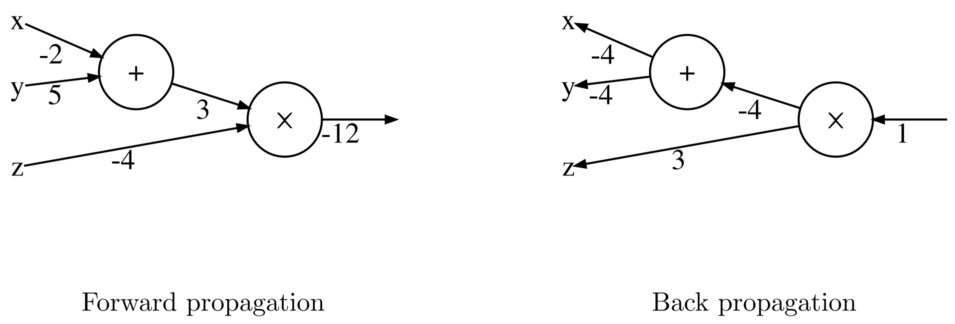


Figure 2.8: Forward and backward propagation.

The back-propagation term is often misunderstood as the optimization of networks. Actually it only refers to the procedure of calculating the gradients. The SGD-family algorithms described in Section 2.3.4 optimize the network according to the gradients.

2.4 Convolutional neural network

Over recent years, the research on computer vision has been drastically altered and pushed forward through the adoption of the convolutional neural networks (CNNs) [37] [39]. In this section, we will give a brief introduction to the concept of CNN and its related terminologies.

2.4.1 Convolutional layer

As we described in Section 2.2.1, the behavior of a neuron can be expressed as a function of vectors as Equ. (2.9). Similarly, a neural network layer can be formulated as

$$\mathbf{y} = f\big(\mathbf{W} \cdot \mathbf{X} + \mathbf{b}\big), \tag{2.37}$$

where $\mathbf{W}$ and $\mathbf{X}$ are matrices representing weights and inputs respectively, $\mathbf{b}$ and $\mathbf{y}$ are vectors for bias and output, the symbol $\cdot$ represents matrix multiplication. The convolutional layer, which is the core component of CNNs, is quite similar to regular neural network layer. The input, weight, bias and activation function remain unchanged. While instead of matrix multiplication, the input and weight are combined with convolution operation in convolutional layers. Its behavior can be expressed as

$$\mathbf{y} = f\big(\mathbf{W} * \mathbf{X} + \mathbf{b}\big), \tag{2.38}$$

where $*$ is convolution operation.

CNN is invented for computer vision task and its inputs are assumed to be images. Regarding the weights are convolved to images, they can be viewed as coefficients of filters as well. Therefore the terminology *weight* in the context of CNN is usually called kernels as well. Processing images by CNNs is equivalent to extracting features from input images by filters. Therefore, CNN is actually a filter-based image processing system with self-learned filter coefficients.

Compared with regular neural network layers, convolutional layers have several unique concepts. We will describe them in the following paragraphs.

Input and output channel. Convolutional layers are commonly used to extract various features from multi-channel images. The channels of the input image/feature are called input channels. Similarly, the channels of output are called output channels. In a CNN, convolutional layers are commonly aligned in cascading style, which means the output channel number of the previous layer should match with the input channel number of the current one.

Kernel size. Because the weights of convolutional layers are just filter coefficients, the size of the filter, also known as kernel size, is a key hyper-parameters for convolutional layers. To simplify the calculation of output shape, the kernel size is usually made of odd number. For computation efficiency, the size is commonly set to 3×3, 5×5 or 7×7.

Padding. For most of convolutional layers, the input is zero-padded before it is fed to the layer. Without this operation, the size of outputs will be shrunk. For example, let the input be a $H \times W$ pixels image, the convolutional layer have a kernel of $kH \times kW$ pixels. The size of the output image will be

$$(H - 2(kH \backslash 2)) \times (W - 2(kW \backslash 2)), \tag{2.39}$$

where $\backslash$ denotes integer division. Zero padding the input enables the kernel size and the output size to be controlled independently, so that the users do not need to balance between the kernel size and the network depth.

Stride. Convolution operation slides the kernel across the image. Stride of a convolutional layer is the step of sliding. For most cases, the stride is 1, which means the kernel is moved one pixel at a time. In some cases, stride 2 is used to reduce the output size. We will discuss more about this configuration in Section 2.4.2. Stride larger than 3 is rarely used in practice.

Let us use a concrete example to demonstrate the behavior of a convolutional layer in detail. Assume that 2 different features, denoted as Y_1 and Y_2, are needed to be extracted from a 3-channel image. The three channels are denoted by I_1, I_2 and I_3. So the convolutional layer will have 3 input channels and 2 output channels. Three sets of weight, W_{11}, W_{12}, W_{13}, are needed to extract the first feature as below

$$Y_1 = W_{11} * I_1 + W_{12} * I_2 + W_{13} * I_3. \tag{2.40}$$

Similarly, another three weights, W_{21}, W_{22}, W_{23}, are needed for the second feature,

$$Y_2 = W_{21} * I_1 + W_{22} * I_2 + W_{23} * I_3. \tag{2.41}$$

If the kernel size is set to 3×3, a 1-pixel padding is probably needed as well. An illustration of such a convolutional layer is provided in Figure 2.9.

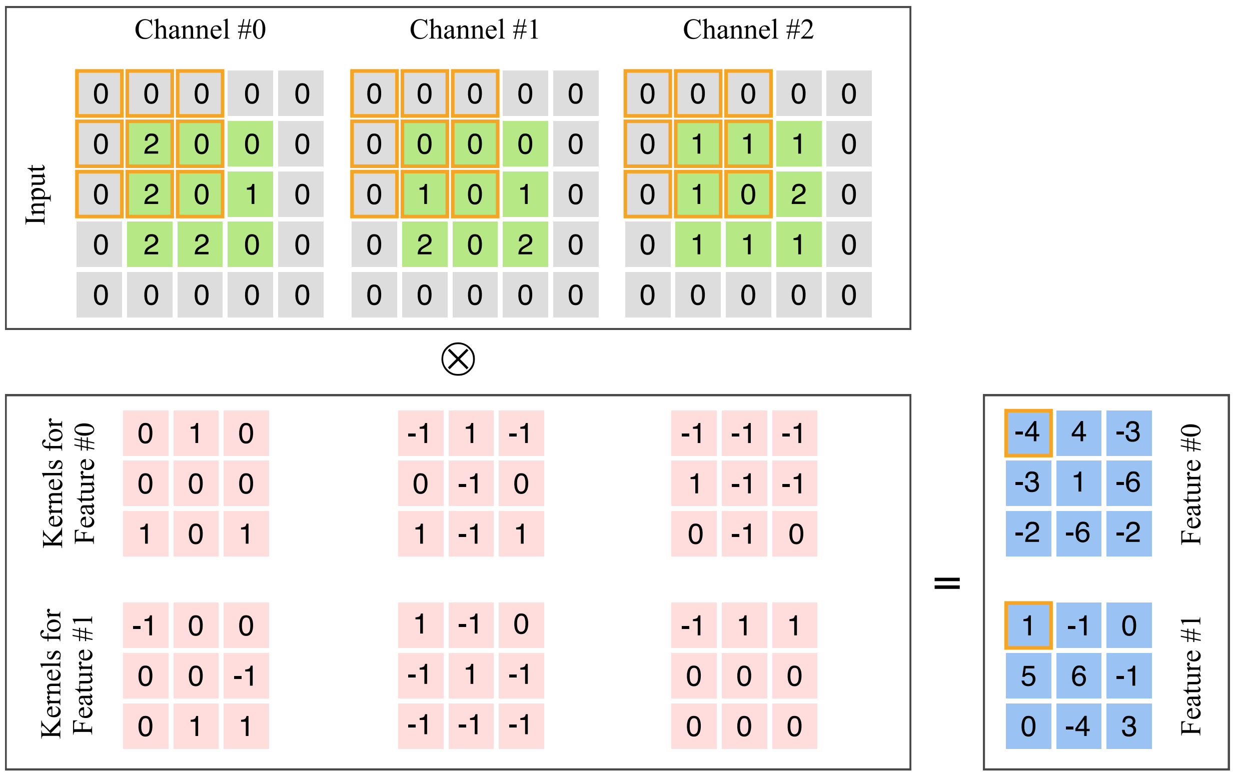


Figure 2.9: A convolutional layer.

As described previously, there might be several sets of weight involved for a multi-channel convolutional layer. To simplify the expression, these weights can be stacked together as a tensor with multiple dimensions. The shape of the weight tensor can be denoted as

$$oC \times iC \times kH \times kW, \tag{2.42}$$

where oC and iC denote the number of output channels and the number of input channels respectively. kH and kW denote kernel height and kernel width respectively. For the previous example, the shape of the weight tensor W will be $2 \times 3 \times 3 \times 3$,

If we reshape the size of an input tensor from $iC \times H \times W$ to $H \times W \times iC \times kH \times kW$ and transpose the weight tensor to $iC \times kH \times kW \times oC$, a multi-channel convolution operation can be expressed as a tensor multiplication,

$$\mathbf{Y}^{H \times W \times oC} = \mathbf{X}^{H \times W \times iC \times kH \times kW} \cdot \mathbf{W}^{iC \times kH \times kW \times oC}. \tag{2.43}$$

In practice, the tensor multiplication can be calculated in parallel and be significantly faster than regular convolution. Therefore it is used by most of recent neural network implementations [3] [11] [51].

2.4.2 Pooling layer

Pooling layers are frequently used with convolutional layers in typical CNNs. A pooling layer receives the output of its previous layer and substitutes the value at a certain position with its nearby summary. For example, the max pooling layer produces the maximum value within a rectangular neighborhood and the average pooling layer reports the average value.

Pooling layers help the representations learned by the network to be more invariant to small translations of the inputs. Being invariant to translation means that if the input is shifted by a small amount, the output is not changed. When existence detection is preferred rather than location detection, the property of translation invariance can be quite useful.

The concepts of kernel size, padding and stride in convolutional layers also apply to pooling layers. It has to be noted that the kernel size in pooling layers refers to the size of the rectangular support area. There is no learnable kernel existing in max and average pooling layers. Illustrations of a max pooling and an average pooling layer are given in Figure 2.10.

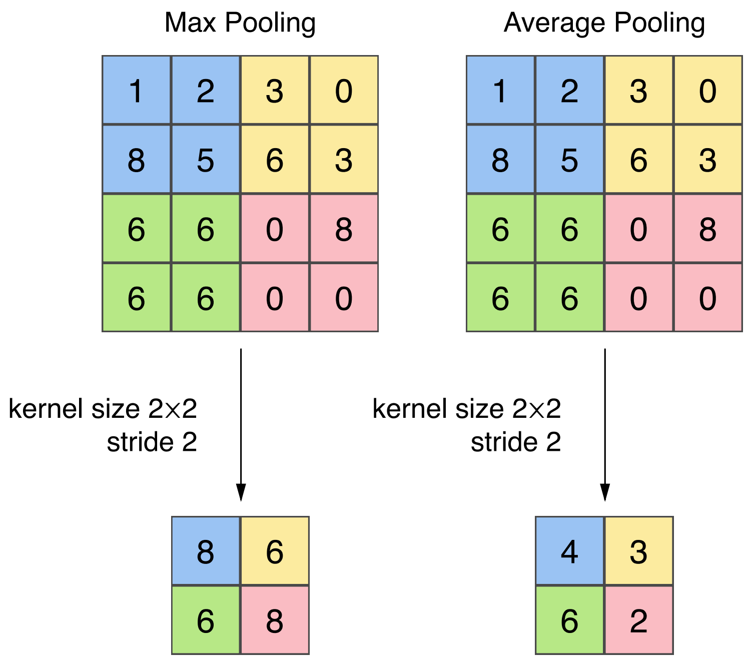


Figure 2.10: Pooling layers.

As shown in Figure 2.10, the output size of pooling layers will shrink when stride is greater than 1. Therefore, pooling layers are also used for down-sampling to reduce the computational cost.

In recent works, convolutional layers with increased stride are often used to replace pooling layers. It is argued that the replacement has the following advantages.

- The learnable kernels in convolutional layers provide more flexibility than regular pooling layers.
- The gradients of convolutional layers are less sparse than pooling layers (especially max pooling). This property might be helpful for the convergence of some networks, such as the generative adversarial networks (GANs).

2.4.3 Common CNN architectures

Most of neural network architectures on computer vision have one or more feature extraction components. Input images are converted to high-level abstracts by these components for classification or synthesis. In this section, we will describe several commonly used feature extraction components.

VGGnet refers to the family of network configurations proposed by Simonyan *et al.* in their work [55]. VGG is short for Visual Geometry Group which is the organization the authors belong to. Its variants, VGG-19 and VGG-16, won the first and second place of ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2014. Because VGG-16 achieves a good balance between performance and efficiency, it is commonly employed by other works as components. As illustrated in Figure 2.11, VGG-16 has 16 layers with learnable weights. The 13 convolutional layers work for feature extraction and the 3 fully connected layers are for classification. VGG-16 achieves fairly good performance while using a limited number of parameters. Therefore, although better performing components exist, VGG-16, especially its 13 convolutional layers, are still frequently employed as parts of larger networks.

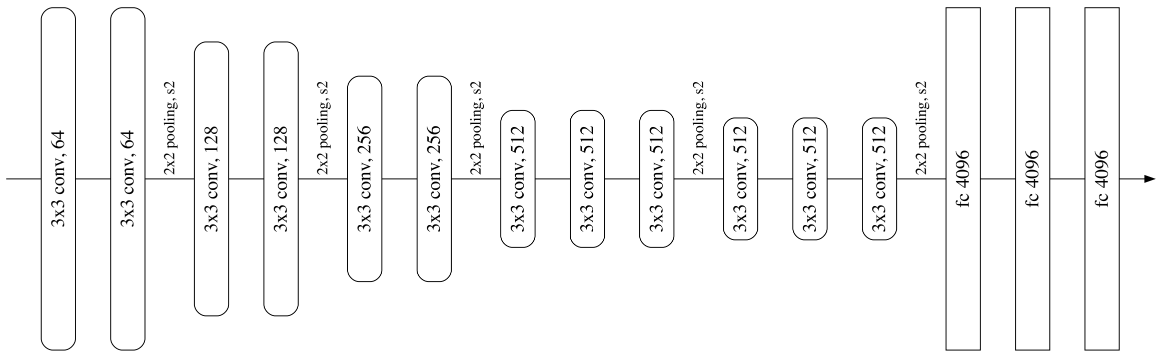


Figure 2.11: A VGG-16 network.

Residual block. As the CNN networks go deeper, problems appear and impede further increase of the performance. One of them is the vanishing/exploding gradients problem [4] [24] which may stop the network from further training. As an example, tanh activation function has gradients in the range (0, 1]. When the gradients of a deep network are computed by back-propagation with chain rule, a large amount of such small numbers are multiplied together to produce gradients for the front layers. This effect makes the error signal decrease exponentially. The front layers will be trained very slowly. The other problem is the degradation of accuracy. When the depth of network increases, accuracy gets saturated and then degrades rapidly, but not due to over-fitting.

To address these issues, He *et al.* proposed the deep residual learning network (ResNet) [28] . A ResNet is formed by stacking multiple residual blocks. Each residual block explicitly guides the layers to learn residual functions with reference to the layer inputs, instead of directly learning target functions. A typical residual block is shown in Figure 2.12. The residual learning architecture makes extreme deep networks easier to optimize. Due to the significantly increased depth, the ResNet achieves outstanding performance.

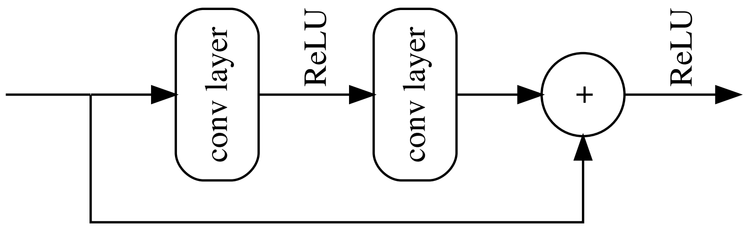


Figure 2.12: Residual block.

Autoencoder is also known as encoder-decoder architecture. In the context of neural network, it refers to a network which is trained to attempt to copy its input to its output.

The network aims to learn data representation via an encoding function $h = f(x)$ and a decoding function $r = g(h)$. The architecture of a typical autoencoder is shown in Figure 2.13. If the network learns to always satisfy $g(f(x)) = x$, it will not be especially useful. Instead, only approximate copy is permitted in autoencoders. The dimensionality of internal layers are smaller than external ones. Therefore, the network is forced to prioritize which parts should be copied. Thus autoencoders often learn useful high level representations from training data.

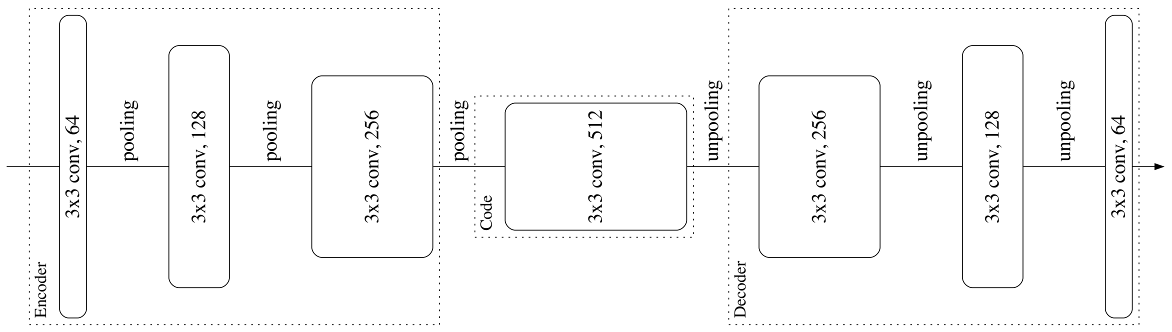


Figure 2.13: Autoencoder.

2.4.4 Spatial invariance components

A desirable computer vision system should be able to disentangle object pose and part deformation from texture and shape in images. To provide more flexible mechanism to deal with variations in the spatial arrangement of data, several architectures have been proposed and will be discussed in this section.

Pooling layer. As we have mentioned in Section 2.4.2, conventional CNNs introduce pooling layers to make networks spatially invariant to the position of features. Small translations of the input will not impact the output. However, due to the fact that pooling summarizes a local rectangular area, spatial information is inevitably damaged. Moreover, because the kernel size of pooling layers are usually small (e.g. 2×2 pixels), only slight translations can be tolerated. A deep hierarchy has to be used to realize more general spatial invariance.

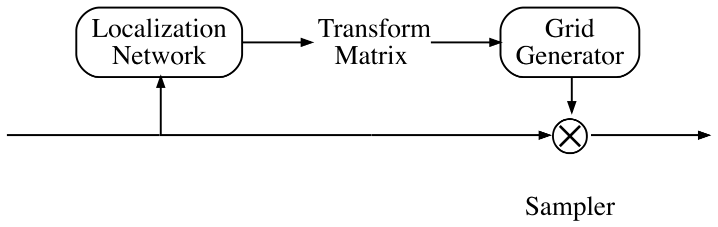


Figure 2.14: Spatial transform network.

Spatial transform network. The spatial transform network (STN) proposed by Jaderberg *et al.* [31] provides an active approach to make networks spatial invariant. STN employs a localization network to estimate a transformation matrix on each input sample, including scaling, cropping, rotations and so on. Then the matrix is used to generate a sampling grid. Finally the input is re-sampled according to the grid. The sampled result will be a transformed version of the input. Unlike pooling layers, whose receptive fields are fixed and local, the STN is dynamic and is performed globally. The *dynamic* here means the behavior of the network will change along with its input.

Dynamic filter network In conventional convolutional layers, the filter kernels are learned via training and fixed for evaluation. Brabandere *et al.* [67] proposed a new framework called dynamic filter network, where the network generates a set of filter kernels which is associated to a specific input. The dynamic filter network has similarities with the STN. Both of them employ a designated network to dynamically report an estimation of the

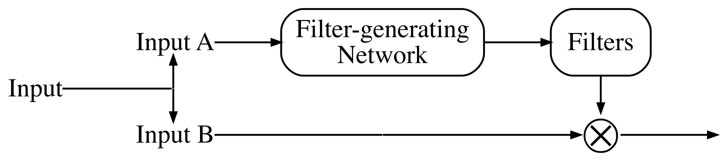


Figure 2.15: Dynamic filter network.

input. Then the estimation is applied somehow to the input. For the STN, the estimation is a transform matrix and for the dynamic filter network, it is filter kernel. Figure 2.15 gives an illustration of the framework. In a dynamic filter network, different variants of the input can be used for estimation and kernel application. Besides conventional convolution, the dynamic filter network can also be extended as locally connected layers, called dynamic local filter layer. The generated kernels are translation variant to the input. Different filters are applied to different positions. As a key component of our proposed network, the dynamic local filter layer will be described in detail in Chapter 4.

Chapter 3

Literature review

3.1 Single image super-resolution

Single image super-resolution (SISR) aims to recover an underlying HR image from a given LR image, assuming the original image is not available. In recent years, neural network techniques have provided numerous new possibilities for SISR and pushed the research much forward. We will review several important neural network based SISR methods in this section.

The super-resolution convolutional neural network (SRCNN) proposed by Dong *et al.* [19] is considered to be one of the first works which introduces CNN to SR tasks. The authors summarized conventional sparse coding based SR methods [69] as a 3-stage framework:

1. Patch extraction: various features are extracted from LR images in the form of high-dimensional vectors which are called feature maps;
2. Non-linear mapping: the feature maps are non-linearly transformed to another set of feature maps;

3. Reconstruction: the feature maps produced by the non-linear mapping are aggregated to the final HR images which are expected to be similar to the ground truths.

The authors found that all of the three stages can be described as convolutional layers. The first convolutional layer performs feature extraction on the bicubic up-sampled input images. The second one applies non-linear transformation to the feature maps with its activation function. The third one averages the transformed feature maps to an HR image, where the average behavior can also be realized as convolution. The architecture of the SRCNN is illustrated in Figure 3.1. With such a lightweight structure, the SRCNN outperformed most of traditional SR algorithms. Moreover, it provided a reasonable theoretical interpretation on its inherent mechanism.

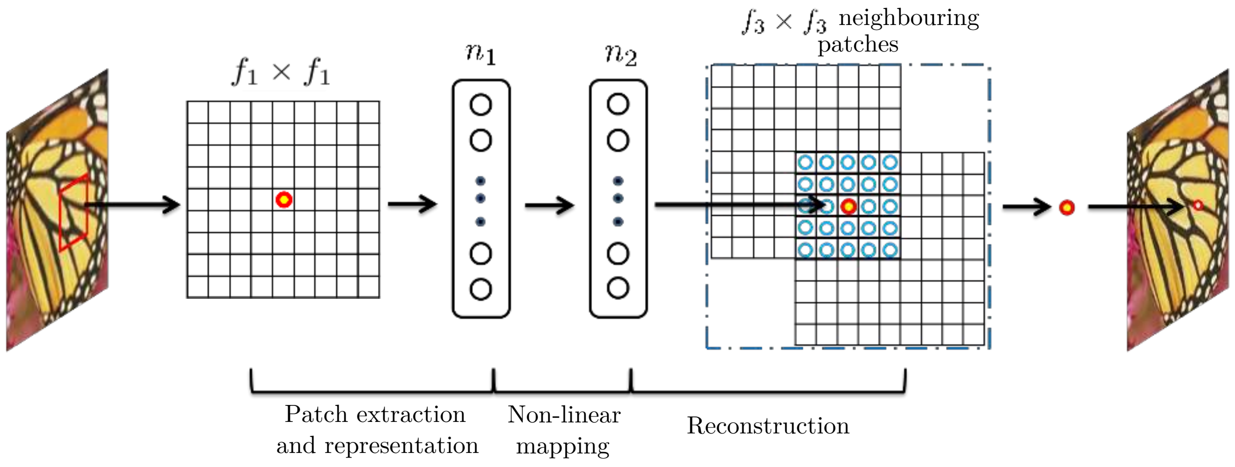

$f_3 \times f_3$ neighbouring patches
$f_1 \times f_1$
n_1
n_2
Patch extraction and representation
Non-linear mapping
Reconstruction

Figure 3.1: The architecture of the SRCNN (adopted from the original paper [19].)

The efficient sub-pixel convolutional neural network (ESPCN) proposed by Shi *et al.* [54] can be considered as an improvement of the SRCNN. In the SRCNN, the input to the network is bicubic up-sampled LR images, hence the SR operation is performed in HR space. The authors of the ESPCN proposed to process the input images by convolutional layers in LR space. Then the extracted feature maps are mapped to the HR space via a novel sub-pixel convolutional layer. The sub-pixel convolutional layer, also called pixel-shuffle layer, arranges multiple LR features as sub-pixels in the HR space as illustrated in Figure 3.2. Convolution in LR spaces can substantially reduce the complexities of computation and memory. In addition, compared with the predefined interpolation, the network may implicitly learn better LR-to-HR mapping. Nowadays, the sub-pixel convolutional layer in the ESPCN has been accepted as a standard up-sampling layer in neural network research.

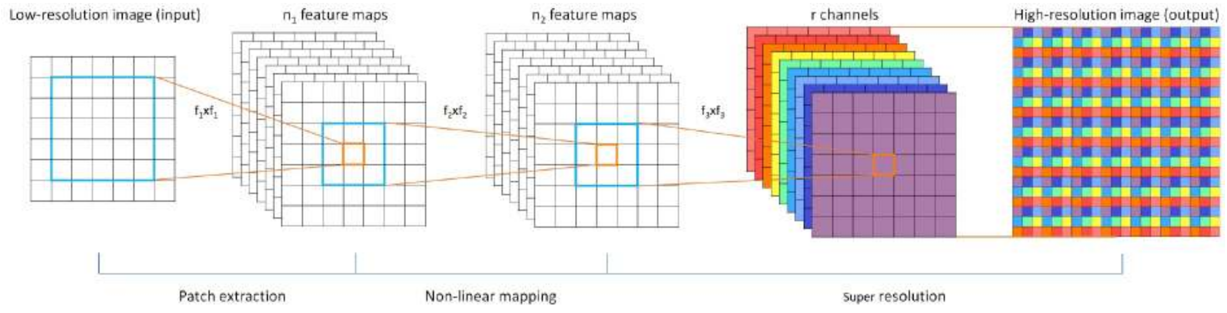


Figure 3.2: The architecture of the ESPCN (adopted from the original paper [54]).

Ledig *et al.* proposed two SISR networks in their work [40], the super-resolution residual network (SRResNet) and the super-resolution generative adversarial network (SRGAN). The SRResNet is a 16-block ResNet [28] based SR network optimized for MSE loss. The ResNet structure makes training very deep network much easier than conventional CNNs. This advantage helps the SRResNet to employ 32 convolutional layers which improve the network performance significantly. The architecture of the SRResNet is exactly the same with the generator network of SRGAN, which can be found in Figure 3.3.

The SRGAN is built on SRResNet and is probably the first generative adversarial network (GAN) [25] based SISR network. In the SRGAN, the MSE loss of the SRResNet is replaced by a perceptual loss which is calculated by a VGGnet [55], as shown in Figure 3.3. The VGG network, as a discriminator network, is trained to differentiate between the original photo-realistic images and the super-resolved images. The perceptual loss motivates the network to recover photo-realistic textures from heavily down-sampled images and generate natural super-resolved images.

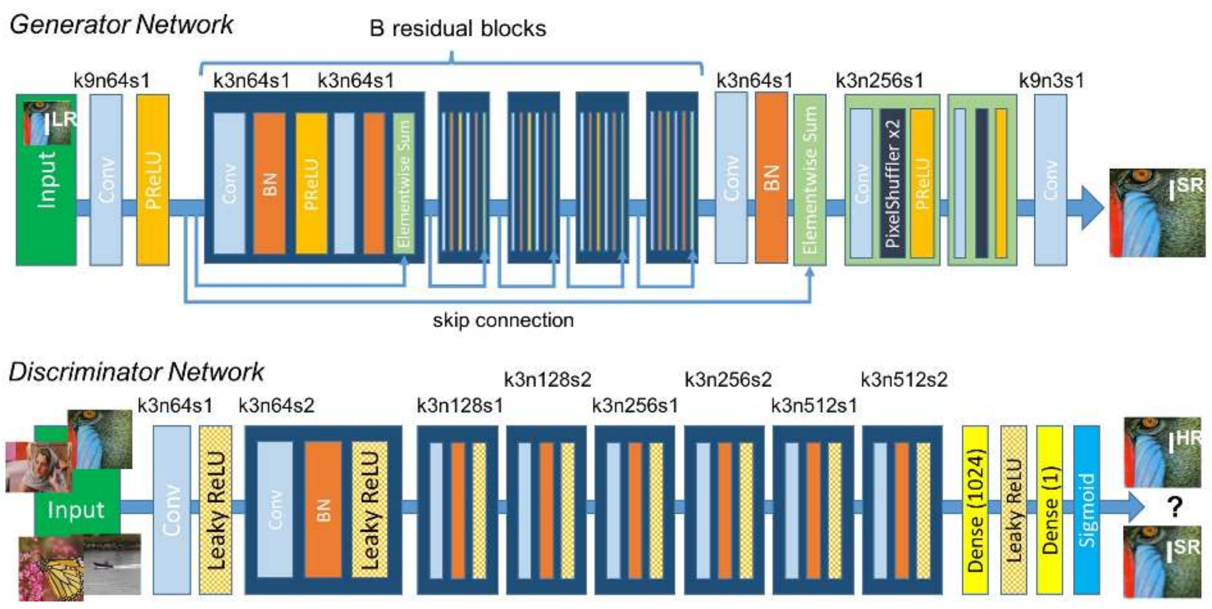


Figure 3.3: The architecture of SRResNet and SRGAN (adopted from the original paper [40]).

The enhanced deep super-resolution network (EDSR) and the multi-scale deep super-resolution (MDSR) proposed by Lim *et al.* [42] improve the SRResNet. The authors optimized the residual block of the ResNet for SR tasks by removing unnecessary batch normalization layers and ReLUs. With the trimmed residual block, the size of the EDSR has been expanded significantly in terms of both depth and filter number, compared with SRResNet. The architecture of the EDSR is shown in Figure 3.4.

When the authors trained the EDSR at scale factor 3× and 4×, they found initializing the network with parameters of pre-trained 2× network will accelerate the training and improve the final performance. This observation indicates that SR tasks at various scale factors are inter-related. The MDSR enhances the EDSR with the ability to take advantage of such inter-scale correlation. In the MDSR, HR images at multiple scales can be reconstructed in a single network. The architecture of the MDSR is shown in Figure 3.5.

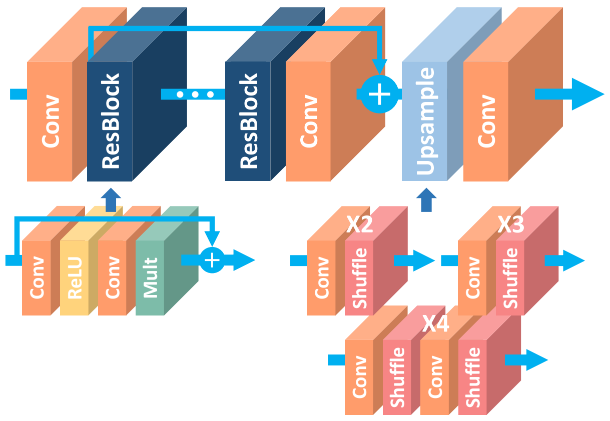


Figure 3.4: The architecture of the EDSR (adopted from the original paper [42]).

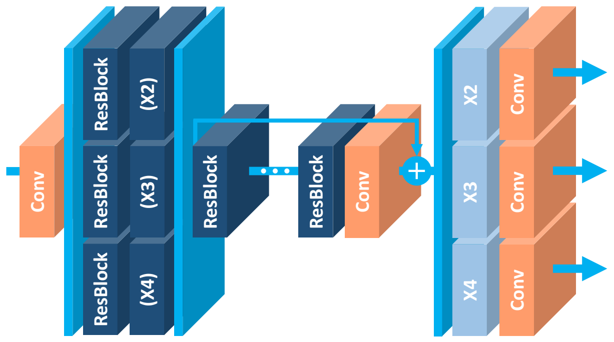


Figure 3.5: The architecture of the MDSR (adopted from the original paper [42]).

3.2 Video Super-Resolution

Different with the SISR where only single LR image is provided, the VSR assumes that various observations of one particular scene are available. These various observations usually appear as the video frames which align along the time dimension. A high degree of correlation exists among the observations and can be exploited to recover lost information. The advent of CNN techniques has pushed the SISR much forward and also benefited the VSR. In this section, we will review three important CNN based VSR methods.

The Video SR network (VSRnet) proposed by Kappeler *et al.* [33] is one of the first attempts to solve the VSR problem by CNNs. The method contains three stages:

1. The motion in the video is estimated by the Drulea's [20] optical-flow algorithm;
2. The input frames are compensated to be similar to the target frame according to the optical-flow;
3. The compensated frames are fed to a post-processing network to perform SR operation.

The authors proposed three variants of SR network where features are concatenated at different stages, as shown in Figure 3.6. The architecture (b) is proved to be the best-performing one.

It is quite creative for the VSRnet to introduce the CNN techniques to the VSR tasks. However problems still exist. The performance of the VSRnet is still sub-optimal to the traditional VSR algorithms. Moreover, the optical-flow computation and motion compensation are done outside the network, hence can not be optimized together with the neural network.

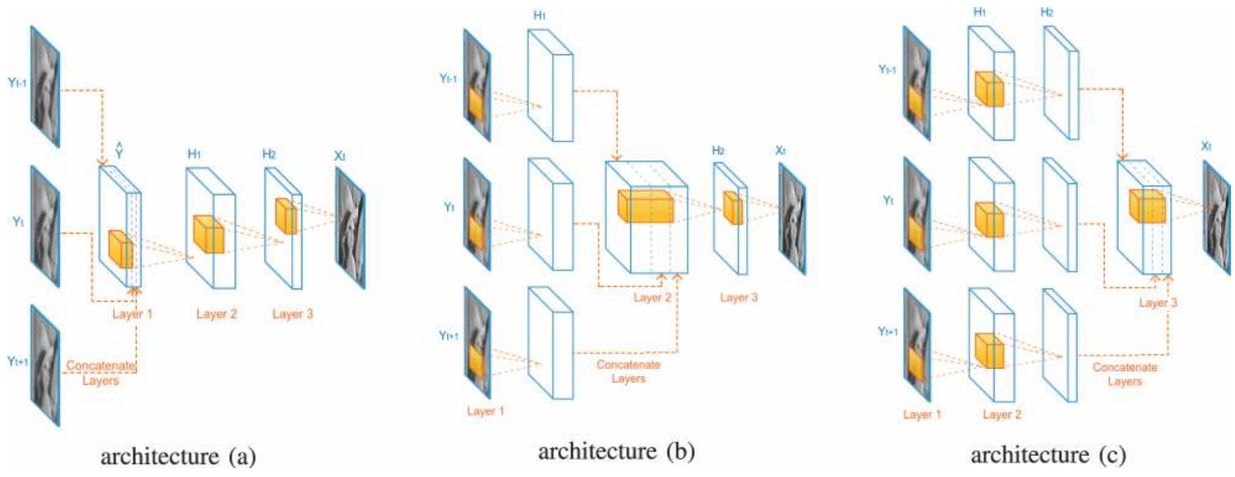


Figure 3.6: Three variants of the VSRnet (adopted from the original paper [33]).

The video efficient sub-pixel convolutional neural network (VESPCN) proposed by Caballero *et al.* [7] is a VSR network which combines the ESPCN with a novel motion compensation module. The motion compensation network employs the spatial transformer network (STN) [31] to encode motions. A multi-scale design as shown in Figure 3.8 is adopted to represent the flow. The compensated frames are then fed to an ESPCN based SR network. A high-level diagram of the VESPCN is illustrated in Figure 3.8.

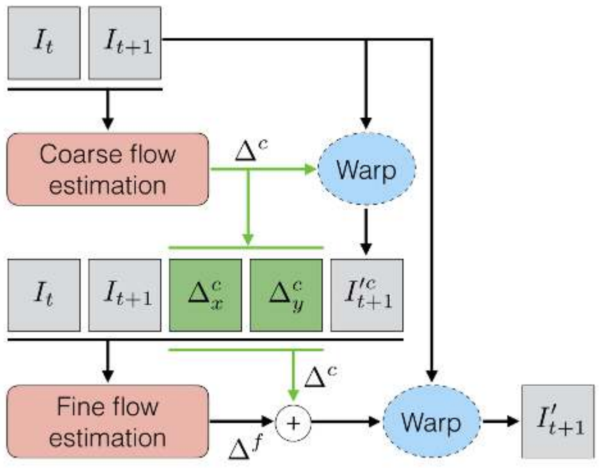


Figure 3.7: The motion compensation module in the VESPCN (adopted from the original paper [7]).

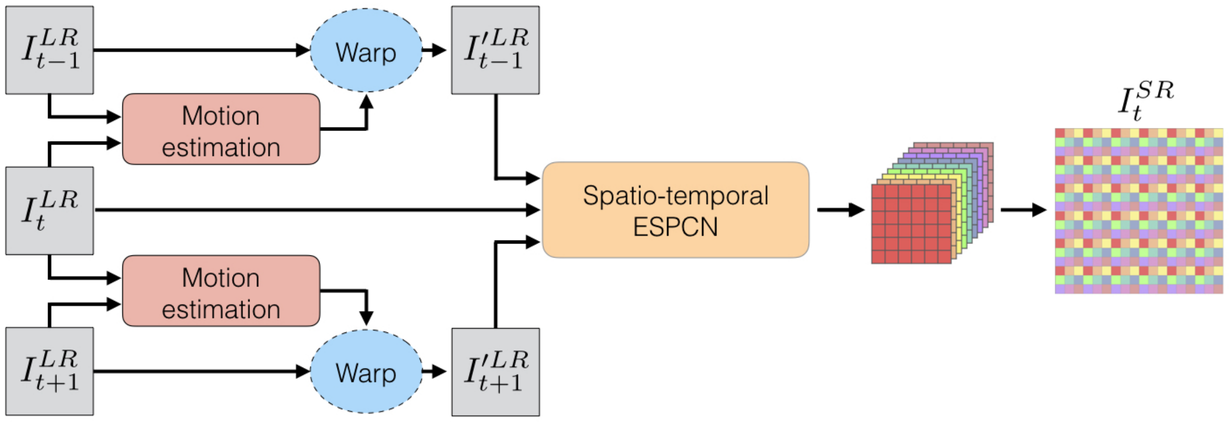


Figure 3.8: The top-level architecture of the VESPCN (adopted from the original paper [7]).

The proposed motion compensation module is a neural network with tunable parameters, hence can be trained together with the SR network as a whole. In addition, the ESPCN is known to have lower computation and memory cost than the SRCNN. Therefore, the VESPCN achieves state-of-the-arts performance while being capable to do real-time processing.

As introduced previously, most recent VSR methods align all other frames to the reference one in LR space to compensate the inter-frame motion. The authors of [58], Tao *et al.*, argued that the compensation in the LR space is suboptimal and proposed a novel subpixel motion compensation (SPMC) layer to perform compensations in the HR space. The SPMC layer utilizes the motion compensation module in the VESPCN to estimate flow information and warps pixels from the LR space directly to the HR space, as illustrated in Figure 3.9.

Due to the fact that the provided LR frames may be insufficient for the SPMC layer to integrate a complete HR image, a Detail Fusion module is proposed to fuse the information in the HR draft and generate the final HR image. A high-level illustration of the architecture is provided in Figure 3.10.

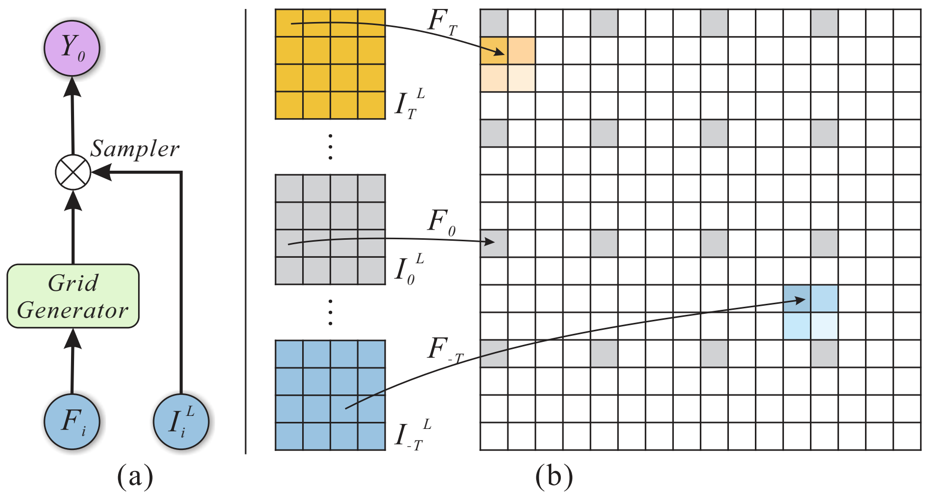


Figure 3.9: The structure of the SPMC layer (adopted from the original paper [58]).

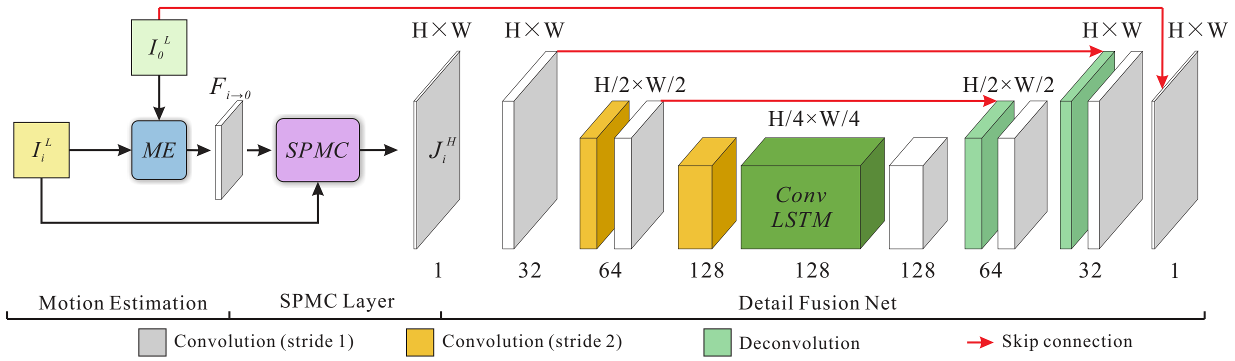


Figure 3.10: The architecture of the detail-revealing deep VSR network (adopted from the original paper [58]).

3.3 Video frame interpolation

Video frame interpolation or motion interpolation is a classic computer vision task which shares common requirements with the VSR. It aims to generate intermediate animation frames between existing ones. This procedure performs similar transformations with the motion compensation in VSR tasks. The difference is that the transformation in frame interpolation is motivated to make animation look more fluid, but the compensation in VSR is used to improve the super-resolved results.

Niklaus *et al.* proposed a novel video frame interpolation method in their recent work [49]. In the method, the interpolation operation is considered as local convolution over neighbor frames. A CNN takes a sequence of frames as input and estimates a set of spatially-adaptive convolution kernels. The kernels are then applied to the input frames to synthesize the interpolated frame. Thus the motion can be captured and compensated in a single operation. An illustration of this procedure is provided in Figure 3.11.

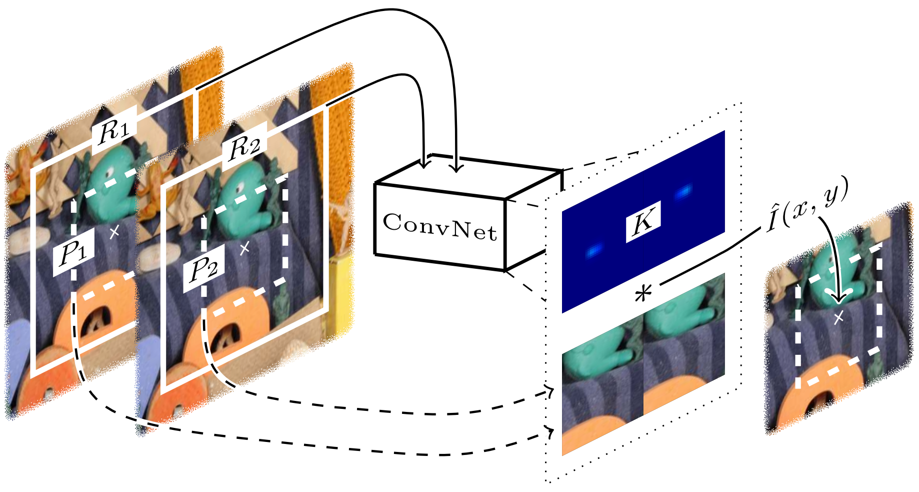


Figure 3.11: Illustration of video frame interpolation via adaptive convolution (adopted from the original paper [49]).

One issue of this method is that the dynamic kernels generally have very limited support region due to the computation cost. Niklaus *et al.* then proposed another method [50] which replaces the regular convolution in [49] by separable convolution. In this work, a regular 2D convolution kernel is represented as the inner product of two 1D kernels. Thus the equivalent kernel size can be expanded to 51 × 51 with no significant increase of computation cost. The network architecture is illustrated in Figure 3.12.

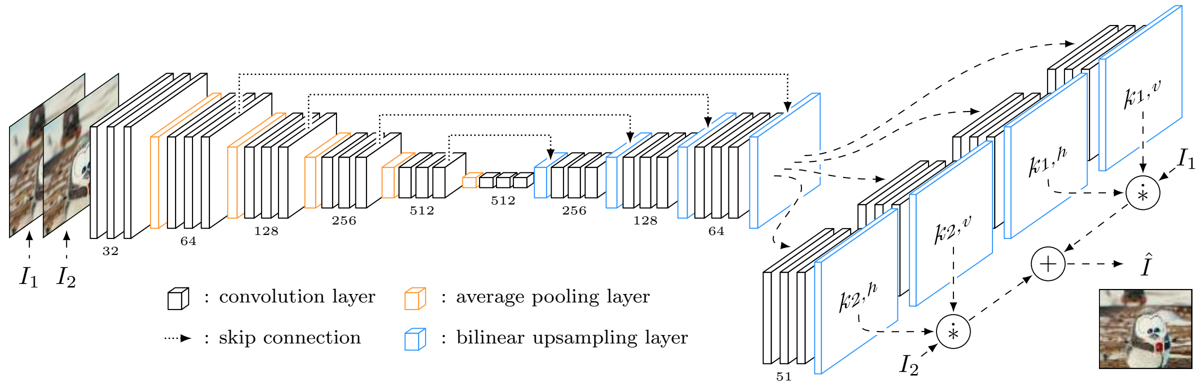


Figure 3.12: Illustration of video frame interpolation via adaptive separable convolution (adopted from the original paper [50]).

Chapter 4

Video super-resolution via dynamic local filter network

The motion of objects among the video frames increases the difficulty of VSR tasks. Recent VSR works usually address this issue by a procedure consisting of the following three stages:

1. A flow-like information is achieved by estimating the motion of the LR input frames with respect to an LR target frame.
2. The motion is compensated according to the flow-like information. After the compensation, all frames have contents similar to the target frame.
3. An HR image is generated by integrating the compensated frames using CNNs. The HR image is expected to be a super-resolved version of the target frame.

In most of recent VSR works, the compensation is performed by manipulating pixels in spatial domain according to the motion estimation. The motion estimation is usually achieved by optical flow or affine-transform based techniques. For optical flow based methods, the final performance largely depends on the accuracy of flow data. However, estimating optical flow is commonly considered computation-expensive and error-prone. Moreover, when the flow data involves non-integer coordinates, the compensated pixels have to be re-sampled. The re-sampling operation usually averages pixels with fixed coefficients and makes edges blurry. Details in the image may not be preserved well. Affine-transform based methods, on the other hand, model the motion in terms of translation, scale, rotation and shearing. All of these transformations can be represented as a transformation matrix as below.

$$\begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix}$$

Obviously, the affine-transform based methods will be more efficient than the optical flow methods because only 6 parameters are involved. However, the diversity of motion patterns which can be expressed will also be significantly limited by the small number of parameters.

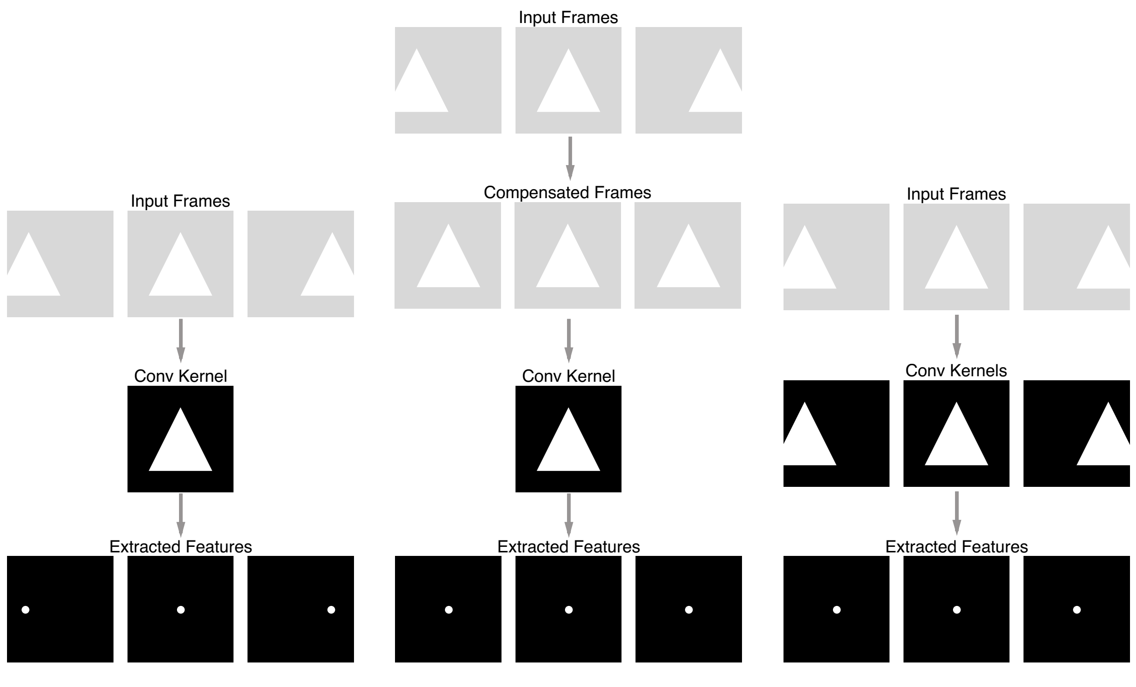
Compensation in spatial domain will inevitably damage the valuable information in the original frames. However, if the compensation happens in convolution kernels, the inter-frame motion can also be canceled. Furthermore, the learned convolution kernels can offer more flexible re-sampling strategies. If the convolution is replaced by locally connected layer, non-rigid transformation may also be handled well.

Based on these ideas, we propose a novel VSR method, Video Super-Resolution via Dynamic Local Filter Network (DLVSR), and its upgraded edition, Video Super-Resolution with Compensation in Feature Extraction (VSR-CFE). Both proposed methods use a dynamic filter network with a locally connected layer for compensation. The DLVSR utilizes a progressive up-sampling work-flow which focuses on 4× SR and an encoder-decoder based refinement module. In the VSR-CFE, a residual block based network is used for feature extraction and fusion and a multi-scale up-sampler is introduced to help the network work for 2×, 3× and 4× SR simultaneously. Both of the proposed methods generate superior results over the current state-of-the-art algorithms, in terms of PSNR and SSIM and visual quality. In the rest of this chapter, we will first introduce the proposed motion compensation via dynamic local filter network. Then we will describe the DLVSR and the VSR-CFE in Section 4.2 and Section 4.3 respectively.

4.1 Motion compensation

Most of conventional convolutions are not spatial invariant, which means that if the input is translated by a small amount, the feature map will be translated as well, as shown in Figure 4.1(a). For the VSR tasks, this property means that the inconsistencies in the input frames will remain in the feature maps, increasing the difficulty of subsequent fusion and synthesis. Therefore extracting the temporal correlation features among the input frames became a key challenge in VSR.

One straightforward solution is to introduce pooling layers as we discussed in Section 2.4.2. But this may not be an optimal solution for VSR because a pooling layer will discard most of the input pixels, possibly making the problem even more challenging than before.



(a) Convolution (b) Compensation (c) Proposed

Figure 4.1: Motion compensation.

Recent VSR methods choose to compensate the motion of the input frames before they are fed into the SR network, as illustrated in Figure 4.1(b). However, as we discussed previously, motion compensation in spatial domain may damage the valuable information in the input frames.

In our work, we propose to make the compensation happen in the convolution kernels. As shown in Figure 4.1(c), if the content in the input frames is translated by a small amount and the corresponding convolution kernel is translated accordingly, then there will be no inconsistency in the extracted feature maps.

To realize this idea, some issues need to be addressed. Most video clips contain non-rigid body transformation, as shown in Figure 4.2. This kind of transformation can not be simply estimated as a single direction movement. In such situations, the processing should be performed in a position-specific manner. A locally connected layer fits such situations well and we will describe it in Section 4.1.1. Another issue is that the objects may move in arbitrary directions in the input frames, as illustrated in Figure 4.3, and a CNN with fixed kernels will not be able to handle all possible situations efficiently. A per-sample feature extraction mechanism is needed. We introduce a dynamic local filter network to fulfill this requirement. To our best knowledge, this is the first time that a dynamic local filter network is employed to solve the VSR problem. Details will be described in the following sections.

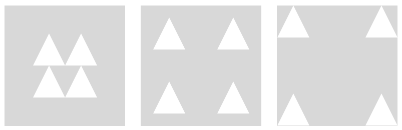


Figure 4.2: Non-rigid body transformation.

4.1.1 Locally connected layer

A locally connected layer can be viewed as a convolutional layer with position-specific

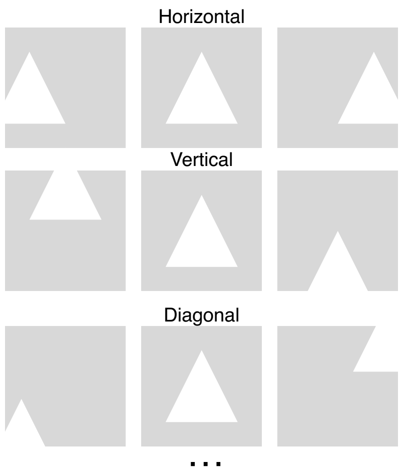


Figure 4.3: Arbitrary direction movement.

kernels. It applies different kernels to patches centered at different positions in an image. Let I be a single channel input image and $P^{(i,j)}$ be a square patch of size $(2K+1)\times(2K+1)$ centered at position (i,j) in I. $P^{(i,j)}$ can be formulated as

$$P^{(i,j)} = \begin{bmatrix} I_{i-K,j-K} & & \cdots & & I_{i+K,j-K} \\ & \ddots & & & \\ \vdots & & I_{i,j} & & \vdots \\ & & & \ddots & \\ I_{i-K,j+K} & & \cdots & & I_{i+K,j+K} \end{bmatrix}. \tag{4.1}$$

For each $P^{(i,j)}$, a corresponding kernel is defined as

$$\theta^{(i,j)} = \begin{bmatrix} w_{1,1}^{(i,j)} & \cdots & w_{2K+1,1}^{(i,j)} \\ \vdots & \ddots & \vdots \\ w_{1,2K+1}^{(i,j)} & \cdots & w_{2K+1,2K+1}^{(i,j)} \end{bmatrix}. \tag{4.2}$$

The locally connected layer f is defined as

$$f(\theta, I) = \begin{bmatrix} \theta^{(1,1)} \odot P^{(1,1)} & \cdots & \theta^{(W,1)} \odot P^{(W,1)} \\ \vdots & \ddots & \vdots \\ \theta^{(1,H)} \odot P^{(1,H)} & \cdots & \theta^{(W,H)} \odot P^{(W,H)} \end{bmatrix}, \tag{4.3}$$

where W, H are width and height of I. The operator $\odot$ is the summation of element-wise product of two matrices, which is defined as

$$A \odot B = \sum_{i=1}^{M} \sum_{j=1}^{N} A_{ij} \cdot B_{ij}, \tag{4.4}$$

where A and B are two $M \times N$ matrices.

4.1.2 Dynamic filter network

The dynamic filter network is a spatial invariant framework proposed by Jia *et al.* [67], where the network generates a set of filter kernels which is associated to a specific input. The dynamic filter network consists of two parts, a kernel generation network (KGN) and an application layer, as illustrated in Figure 4.4. The KGN produces filter kernels based on the input and the generated kernels are applied to the input via an application layer.

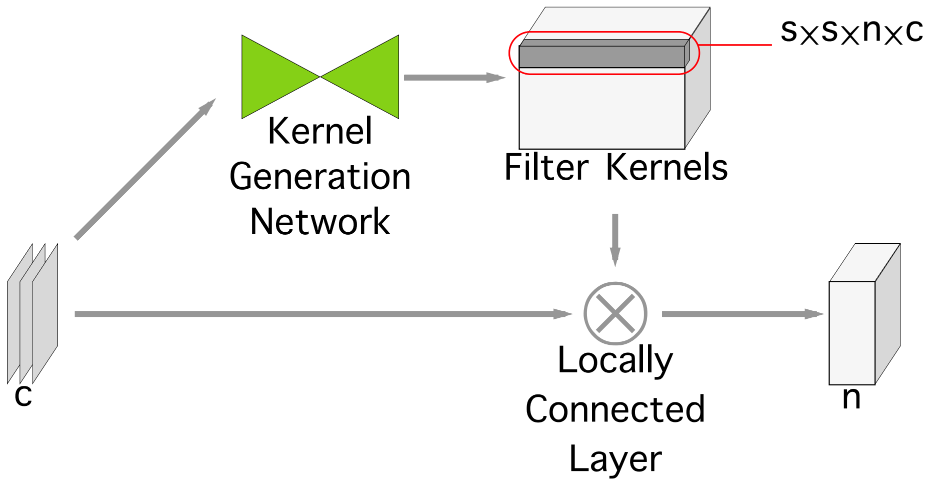


Figure 4.4: The structure of a dynamic (local) filter network.

Unlike conventional CNN, the filter kernels produced by the KGN are sample-specific, which means they are not fixed after training. For the VSR tasks, the KGN takes a sequence of frames as input and generates a set of filters which is associated with this sequence of frames. Then the generated filters will be able to compensate the motion among the input frames.

The shape of the generated kernels depends on the type of the application layer. In our work, the application layer is a locally connected layer which has been described in Section 4.1.1. As illustrated in Figure 4.4, the KGN takes an input $I \in \mathbb{R}^{c\times h\times w}$, where c is the number of input frames, h and w are height and width of the input image I respectively, and generates a group of filter kernels $\theta \in \mathbb{R}^{s\times s\times n\times c\times h\times w}$, where s is the kernel size of the generated filters, and n is the number of output channels. The output of the application layer will be of size $n \times h \times w$.

A dynamic filter network which employs locally connected layer as the operation layer is called a dynamic local filter network. This sample-specific and position-specific structure should be able to perform the motion compensation for the VSR task.

4.2 Video Super-Resolution via Dynamic Local Filter Network

In this section, we will describe the proposed network architecture of DLVSR. The DLVSR consists of two components, an SR module and a refinement module, as illustrated in Figure 4.5. First, the input LR frames are converted to YCbCr color space and only luminance channels are kept for subsequent processing. Then the frames are stacked and input to the network for super-resolving. The SR module compensates the motion and super-resolves the feature maps progressively to synthesize a draft HR image. Then the HR draft is enhanced by the refinement module to obtain the final result.

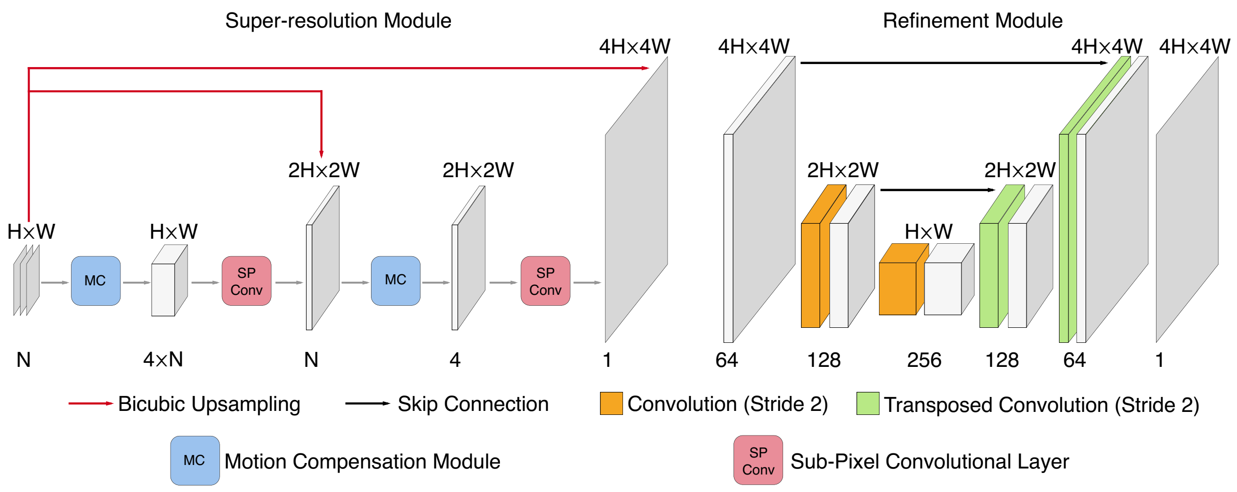


Figure 4.5: The architecture of the DLVSR.

4.2.1 Super-resolution module

In our proposal, the SR module focuses on 4× scale factor. It employs dynamic local filter networks for motion compensation and sub-pixel convolutional layers [54] for upscaling. The motion compensation block is a dynamic local filter network, where an autoencoder [46] is used as the KGN which is shown in Figure 4.6. The number of input channels is c and the number of the output channels is $s \times s \times n \times c$, where $s = 3$, $c = 3$, $n = 4$ for our configuration. All the convolutional layers use a kernel size of 3×3. Larger kernel sizes could provide better performance, but would take longer computation time and occupy more memory than the size of 3×3. Considering the available computation resource at the time of this writing, filters with kernel size larger than 3×3 are impractical. Inspired by the work of [50], the average pooling layers are used in the KGN with kernel size of 2×2 and stride 2. ReLUs are used for all convolutional layers except the last one. Other parameters are noted in Figure 4.6.

With the motion compensation block, common features can be extracted from the input frames without distractions caused by inter-frame differences. The feature maps can then be super-resolved via up-sampling layers, which are sub-pixel convolutional layers in our

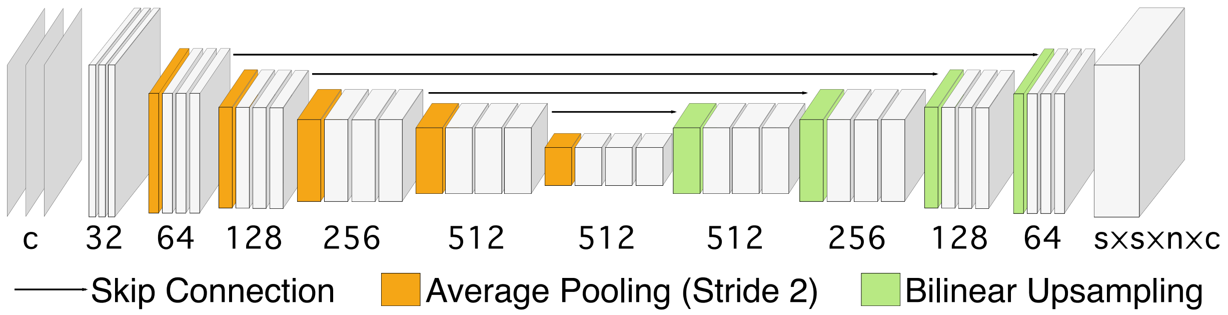


Figure 4.6: The KGN of the motion compensation module in the DLVSR.

proposed method. However, since the KGN-generated filters have fixed kernel size, the maximum motion magnitude they can adapt to is limited. Increasing the kernel size can be one solution, but the network would rapidly become impractical because of the growing computation cost. Separable convolution [50] is another reasonable solution. However, since the separable convolution aims to reduce the number of parameters of a 2D filter by using the inner-product of two 1D filters, it is inevitable to lose representation flexibility and make performance suboptimal.

In our proposed method, we use two cascaded motion compensation blocks and sub-pixel convolutional layers to increase the image resolution progressively as illustrated in Figure 4.5. Each sub-pixel convolutional layer performs a 2× up-sampling. With the cascading structure, the receptive field of the generated kernels is enlarged to an acceptable size. We also equip the module with skip connections [28], which add bicubic up-sampled target frames to the outputs of sub-pixel convolutional layers, to explicitly guide the network to learn residual contents.

4.2.2 Refinement module

Although the SR module has the capability to generate satisfactory results, we observed that blur and noise still exist in the results. An additional refinement stage is supposed to be beneficial. We introduce a refinement module, inspired by the detail fusion network in [58]. The module adapts an encoder-decoder style architecture [46] which is commonly used for image deblurring and denoising. The first and last layer use a kernel size of 5×5. The other convolutional layers have a kernel size of 3×3. The transposed convolutional layers are with a kernel size of 4×4. ReLUs are used for every layer. Other parameters are illustrated in Figure 4.5.

4.3 Video Super-Resolution with Compensation in Feature Extraction

In this section, we will introduce the VSR-CFE, which is an upgrade of the DLVSR. The proposed network consists of two components: a motion compensation module and an SR module. A high-level illustration is provided in Figure 4.7, where colored rectangles represent neural network layers, circles represent input and output images. Note that the input is a sequence of normalized low-resolution images. $2\times, 3\times$ and $4\times$ represent bilinear up-sampled target frames in 2, 3 and 4 scale factors respectively. $I_{2\times}, I_{3\times}, I_{4\times}$ are the corresponding SR results. $\otimes$ denotes locally connected layer and $\oplus$ denotes element-wise summation.

4.3.1 Compensation Module

The compensation module is a dynamic local filter network whose internal architecture is shown in Figure 4.8, where the c in the last two convolutional layers denote the number of input frames, $\otimes$ denotes locally connected layer.

The KGN takes an input $I \in \mathbb{R}^{c \times h \times w}$, where c is the number of input frames, h and w are height and width of the input image I respectively, and generates a group of filter kernels $\theta \in \mathbb{R}^{c \times s \times s \times h \times w}$, where $s \times s$ is the kernel size of the generated filter. Note that the

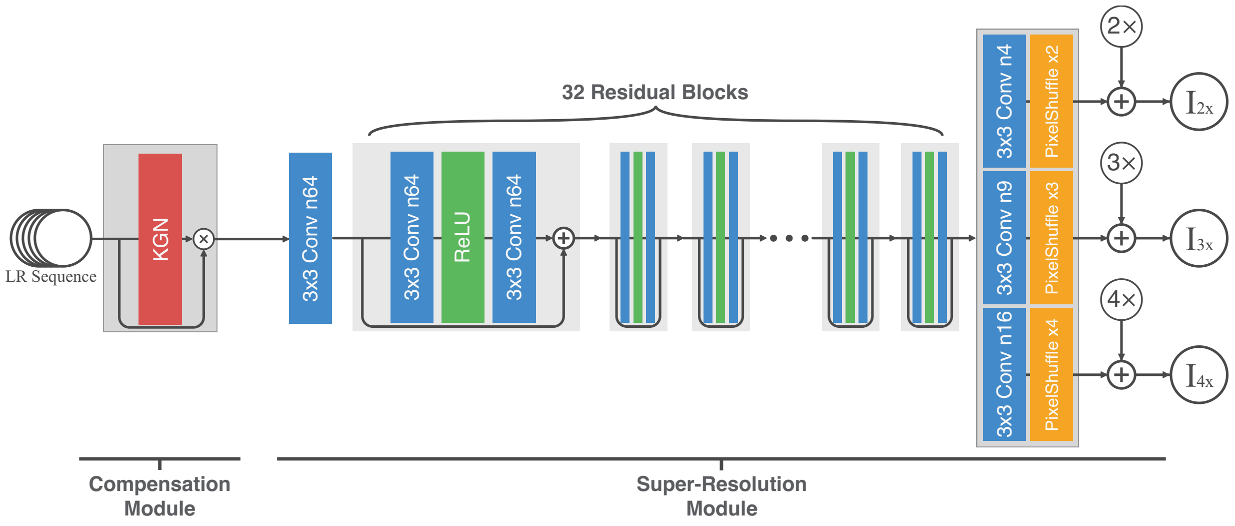


Figure 4.7: The architecture of the VSR-CFE.

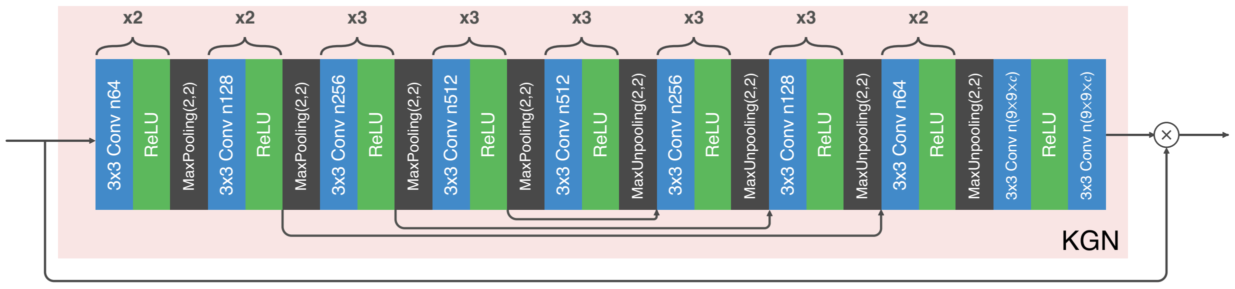


Figure 4.8: The internal architecture of compensation module.

application here is a *depthwise* [12] locally connected layer. The depthwise means that the θ for the i-th channel, denoted as $\theta_i \in \mathbb{R}^{1\times s\times s\times h\times w}$, will only be applied to I_i which is the corresponding channel of I. The output of the application will be of size $c \times h \times w$.

In the VSR-CFE, the KGN is an autoencoder whose encoder part is the first 13 layers of the VGG-16 network which is smaller than the one in the DLVSR. Like the VGG-16 network, the KGN also uses the max pooling layers. Skip connections are used with element-wise summation. Other parameters can be found in Figure 4.8. As we discussed, the generated filters have fixed kernel size, hence the maximum adaptable motion magnitude is limited. The kernel size has to be selected carefully to balance coverage and efficiency. Thanks to the reduced number of parameters in the KGN, the kernel size in the VSR-CFE can be increased to 9×9 with affordable resource consumption.

4.3.2 Super-Resolution Module

While the compensation module can extract features while ignoring inter-frame differences, some extra components are still needed to integrate these features to complete the SR task. Most of recent SISR methods focus on integrating features and serve as inspiring examples for designing our SR module. Recently, ResNets [28] based SISR method SRResNet [40] and EDSR [42] exhibited excellent performance. The MDSR proposed by Lim *et al.* [42] can even perform multi-scale SISR in a single network.

Inspired by the works mentioned above, we propose a residual block based multi-scale SR module which works for $2\times$, $3\times$ and $4\times$ super-resolving simultaneously. The architecture of the proposed SR module is illustrated in Figure 4.7. First, a convolutional layer converts the output of the compensation module to a 64-channel feature map to make the subsequent network adaptive to inputs with various numbers of channels. Then the feature map is processed by 32 cascaded residual blocks. The processed feature map is then passed to a multi-scale up-sampler. For $2\times$, $3\times$ and $4\times$ scales, the up-sampler converts the

64-channel feature map to 4, 9 and 16-channel feature maps respectively. The sub-pixel convolutional layers convert each feature map to an SR residual. The SR residuals are then added to corresponding bilinear up-sampled target frames to generate the final SR results.

4.4 Improvements from the DLVSR to the VSR-CFE

The VSR-CFE introduces a smaller KGN than the DLVSR and uses depthwise locally connected operation instead of regular one. These upgrades significantly reduce the number of parameters used by the VSR-CFE compared with the DLVSR. A detailed comparison will be provided in the following paragraphs. To simplify the calculation, the number of bias parameters will be ignored because bias parameters are much less than weights.

In the DLVSR, a KGN has 25,112,160 kernel parameters. However, due to the fact that two KGNs are used in the SR module, the number of parameters used is 50,224,320. Additionally, the refinement module has 1,660,032 parameters. Therefore the DLVSR contains about 51,884,352 parameters. On the other side, the VSR-CFE employs only one KGN whose number of parameters is 18,441,585. The SR module has 2,377,728 parameters. So the VSR-CFE totally has 20,819,313 parameters which is about 40% of the DLVSR.

Although the VSR-CFE has much less parameters than the DLVSR, it achieves slightly better performance than the DLVSR. Moreover, the training dataset used by the VSR-CFE is only about 30% of the one used by the DLVSR. The detailed experimental setup and performance comparison will be discussed in the next section.

Chapter 5

Experimental results

5.1 Environment setup

The hardware used for the experiments is an Intel-based PC equipped with NVIDIA GPU. The system has an Intel Core i7-7700K CPU, 16 GB system memory and 512 GB Solid State Disk. The graphical adapter is NVIDIA Geforce GTX 1080Ti which is an high-end GPU based on Pascal micro-architecture. There is 11 GB GPU-dedicated memory installed on this adapter to enable training of large neural network.

The software framework we used is PyTorch [51] version 0.3 which is a popular deep learning framework backed by Facebook, Inc. Another popular deep learning framework is Tensorflow [3] backed by Google LLC. Different with Tensorflow which focuses on industrial applications, PyTorch places more efforts on research purposes. It has concise syntax and flexible architecture. Therefore, in recently published works, PyTorch is more frequently used than Tensorflow by researchers. We deployed our implementations on an operating system of Debian GNU/Linux version 9.4. NVIDIA proprietary driver and CUDA toolkit are installed to support GPU accelerated computation.

5.2 Evaluation strategy

5.2.1 Test datasets

We compared our methods with four recent VSR methods for reference.

- BayesSR [44] is a Bayesian approach to adaptive VSR and is considered the best-performing traditional VSR algorithm.
- VSRnet [33] is one of the first works which employs CNN for VSR.
- VESPCN [7] is an CNN based real-time VSR method which focus on the balance between efficiency and performance.
- SPMC [58] is considered the latest and best-performing VSR method at the time of writing.

According to the original papers, all the four VSR methods have been tested on the VID4 dataset [44]. The VID4, also known as Videoset4, consists of four test videos with various features as below.

- *Calendar* has plenty of hand-drawn texture and text content.
- *City* has buildings with rich details at different scales.
- *Foliage* has fast-moving vehicles and thin tree branches.
- *Walk* contains a large number of human faces and clothes.

It should be noted that the VID4 dataset is not used as training data in any form.

Although the VID4 has been tested on the methods [44] [33] [7] [58], the experiment setups of these tests still have minor differences. The first difference is the number of the frames used for reconstruction. BayesSR used 30 frames to reconstruct an HR frame. VSRnet, VESPCN and SPMC receive 3 consecutive frames. Another issue is the number of the frames to be tested. The input frames are selected by a sliding window in which the middle frame is used as the target frame, as illustrated in Figure 5.1. Let the width of the sliding window be N, which means a set of N consecutive frames is needed. When any of the first $N\backslash 2$ frames or the last $N\backslash 2$ frames are chosen as the target frame, where $\backslash$ denotes integer division, there will not be enough consecutive frames available around the target frame. VSRnet and VESPCN avoid to use the first two frames or last two frames as target frame to ensure a set of 3 or 5 consecutive frames is always available for experiments. BayesSR and SPMC did not provide description about how they handled the boundary cases. According to the published experimental results, the numbers of frames super-resolved by various methods are shown in Table 5.1. We followed the setup of VSRnet and VESPCN and skipped the first three frames and the last three frames to guarantee that up to 7 consecutive frames are always available.

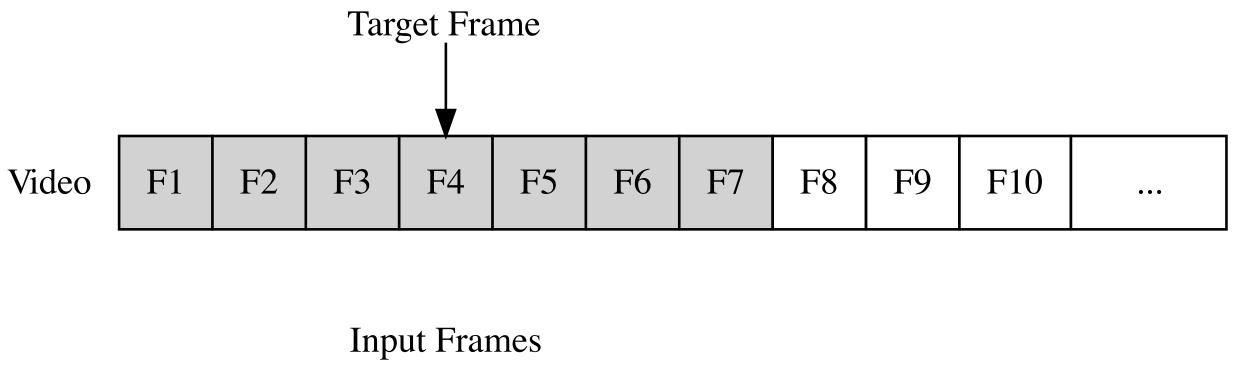


Figure 5.1: The input frames in the video.

The last factor to be clarified is the dimension of the frames. As we discussed in Section 2.4.1, zero-padding is frequently included in convolutional layers to let the kernel size and the output size be controlled independently. However, the introduction of these zeros also inevitably bring distortion to the image boundaries. Therefore, VSRnet and VESPCN chose to discard a 8-pixel border on each side while BayesSR and SPMC just use the

Table 5.1: The numbers of frames super-resolved by various methods

	Calendar	City	Foliage	Walk
Original	41	34	49	47
BayesSR	40	33	48	46
VSRnet	37	30	45	43
VESPCN	37	30	45	43
SPMC	41	34	49	47
Proposed	35	28	43	41

original dimensions, as shown in Table 5.2. We use the original dimensions of each video to test our methods.

Table 5.2: The dimensions of input frames ($pixel \times pixel$) used by various methods

	Calendar	City	Foliage	Walk
Original	720×576	704×576	720×480	720×480
BayesSR	720×576	704×576	720×480	720×480
VSRnet	704×560	688×560	704×464	704×464
VESPCN	704×560	688×560	704×464	704×464
SPMC	720×576	704×576	720×480	720×480
Proposed	720×576	704×576	720×480	720×480

5.2.2 Blur and noise

One of the differences between the traditional algorithms and the neural work based methods is the assumption of the blur and noise. Traditional algorithms, like the BayesSR, will simultaneously estimate the underlying motion, blur kernel and noise level while reconstructing the HR frames. On the other hand, most of the neural network based methods assume that no blur or noise exists in the input frames. One explanation on this approach is that the neural network can learn to reduce the blur and noise without too much effort. For example, random blurs and noises can be applied on-the-fly to the input frames in training to make the network robust to these attacks. The existence of the blur and noise will not alter the design of neural network methods very much. Therefore they are mostly not considered in the test of neural network methods. We will follow this convention in the experiments on our methods.

5.2.3 Color space

Another important factor should be considered is which color space to use for measurements. Recent SISR methods are mostly trained and tested on RGB color space. However, the papers on VSR barely state this setup clearly. We took an experiment on the official experimental results of the reference methods to infer the measurements they used. We found that the PSNR and SSIM in these works are measured on the luminance channel which is obtained via the RGB-YCbCr conversion defined in ITU-R BT.601 standard.

There are two mainstream RGB-YCbCr conversion standards frequently used in computer implementations. The first is the one defined in the ITU-R BT.601 standard. It can be formulated as

$$Y' = 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256}, \tag{5.1}$$

$$C_B = 128 - \frac{37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256}, \tag{5.2}$$

$$C_R = 128 + \frac{112.439 \cdot R'_D}{256} + \frac{94.154 \cdot G'_D}{256} + \frac{18.285 \cdot B'_D}{256}, \tag{5.3}$$

where the R'_D, G'_D, B'_D are 8-bit digital representations, $\mathbb{Z} \in [0, 255]$, of the R, G, B value. The inverse conversion is defined as

$$R'_D = \frac{255}{219} \cdot (Y' - 16) + \frac{255}{112} \cdot 0.701 \cdot (C_R - 128), \tag{5.4}$$

$$G'_D = \frac{255}{219} \cdot (Y' - 16) - \frac{255}{112} \cdot 0.886 \cdot \frac{0.114}{0.587} \cdot (C_B - 128) - \frac{255}{112} \cdot 0.701 \cdot \frac{0.299}{0.587} \cdot (C_R - 128), \tag{5.5}$$

$$B'_D = \frac{255}{219} \cdot (Y' - 16) + \frac{255}{112} \cdot 0.886 \cdot (C_B - 128). \tag{5.6}$$

In this standard, the Y' ranges from 16 to 235 even though it is stored as 8-bit binary which ranges from 0 to 255. The extra room below 16 and over 235 are called footroom and headroom respectively. They are reserved for possible overshooting in analog video equipments. The ITU-R BT.601 standard is used in Matlab and scikit-image library of Python.

The other standard is JPEG conversion standard which is defined as

$$\hat{Y'} = 0 + 0.299 \cdot R'_D + 0.587 \cdot G'_D + 0.114 \cdot B'_D, \tag{5.7}$$

$$\hat{C_B} = 128 - 0.168736 \cdot R'_D - 0.331264 \cdot G'_D + 0.5 \cdot B'_D, \tag{5.8}$$

$$\hat{C_R} = 128 - 0.5 \cdot R'_D - 0.418688 \cdot G'_D - 0.081312 \cdot B'_D. \tag{5.9}$$

In this standard, the Y' covers the full range of $[0, 255]$. This standard is used in the Python Imaging Library (PIL) which is widely utilized by Python based neural network frameworks. Therefore, the images generated by these frameworks have to be converted to the ITU-R BT.601 standard.

Because only the luminance channel is involved in the measurement, the chroma components, C_B and C_R, are constantly assumed to be 128. The Equ. (5.4) (5.5) (5.6) can then be rewritten as

$$R'_D = \frac{255}{219} \cdot (Y' - 16), \tag{5.10}$$

$$G'_D = \frac{255}{219} \cdot (Y' - 16), \tag{5.11}$$

$$B'_D = \frac{255}{219} \cdot (Y' - 16), \tag{5.12}$$

which can be incorporated into Equ. (5.7). We then obtain

$$\begin{aligned} \hat{Y'} &= 0.299 \cdot \frac{255}{219} \cdot (Y' - 16) + 0.587 \cdot \frac{255}{219} \cdot (Y' - 16) + 0.114 \cdot \frac{255}{219} \cdot (Y' - 16) \\ &= \frac{255}{219} \cdot (Y' - 16). \end{aligned} \tag{5.13}$$

Therefore, the luminance conversion from JPEG standard to ITU-R BT.601 standard can be formulated as

$$Y' = \frac{219}{255} \cdot \hat{Y'} + 16. \tag{5.14}$$

5.3 Training strategy

5.3.1 Training datasets

One of the most significant factors which influence the performance of a trained neural network is the selection of its dataset. The test dataset should be able to provide an objective estimation of the network performance. The training dataset should be sufficient in terms of quality and quantity to realize the capability of the network. Moreover, the test and training dataset should be correlated but not too similar to preserve the generality. In this section, we will discuss the dataset selection for the VSR task and introduce our own setup.

For the VSR task, the training dataset is supposed to be video clips of high quality. The frames of the clips should be images of high resolutions with fine details but no obvious blur, noise or other flaws. The contents are supposed to be consecutive without sudden scene change. The amount of the clips should be sufficient with respect to the capacity of the models.

To our best knowledge, there is no public available resource which satisfy all above requirements well. The datasets used in recent methods differ from each other.

- VSRnet [33] uses the Myanmar video [2] as the dataset. The video has 4K resolution (3840×2160 pixels) and is down-sampled to 960×540 pixels as ground truths. There are 59 scenes in the video, of which 53 scenes are used for training.
- VESPCN [7] employs the Consumer Digital Video Library (CDVL) [1] as the training dataset. There are totally 100 full HD (1920×1080 pixels) videos and 30 random clips are sampled from each video. Therefore, 3000 samples are used for training.
- SPMC [58] collects 975 full HD video clips shot with high-end cameras. Each of the clips has 31 frames which are down-sampled to 960×540 pixels for use. 945 of them are used as training data and the rest are for testing. To compare with other methods, the method is also tested on VID4.

According to the training datasets used by recent works, we can infer that a large dataset is preferred for VSR tasks, especially for modern networks which are much deeper than previous ones. Therefore, we setup our training dataset as below.

DLVSR. We collected 17 videos from the Internet to train the DLVSR. 8 of them are 4K commercial videos from [2] and the rest are full HD urban tour videos downloaded from YouTube. All the 17 videos are then analyzed by PySceneDetect [9] and cut to 3022 scenes. We used the first 30 frames of each scenes as the training dataset and down-sampled them to 1280×720 pixels.

VSR-CFE. We used 945 video clips to train the VSR-CFE. These clips are randomly selected from the 3022 clips used in DLVSR. We still only used the first 30 frames of each clip and down-sampled them to 1280×720 pixels. Although the two methods share parts of their training datasets, the designs of them do have a few differences. For example, the two methods use different scale factors, loss functions, etc. Correspondingly, their training strategies will not be the same and will be described separately in the rest parts of this section.

5.3.2 Training the DLVSR

The SR scale factor for DLVSR is fixed to 4. The HR targets are 384×384 pixels patches which are randomly cropped from frame sequences. The LR input frames are 96×96 pixels patches which are down-sampled by bilinear interpolation from corresponding HR frames. The frame sequences are sets of 3, 5 or 7 consecutive frames which are randomly chosen from the training video clips.

As we discussed in Section 2.2.3, most activation functions expect their the inputs to have a zero mean. However, the original pixel values of our training frames are within $[0, 1]$. Therefore, we normalized the pixel values to range $[-1, 1]$ with the formula

$$p_{norm} = p_{origin} \times 2 - 1, \tag{5.15}$$

where p_{origin} and p_{norm} are the original pixel value and normalized pixel value respectively.

We use Adam [36] optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$. Due to the fact that the DLVSR contains a large number of parameters, the mini-batch size for training has to be fixed to 1 due to our hardware limitations.

Because the DLVSR consists of a SR module and a refinement module. To explicitly guide the refinement module to learn the desired behavior, the two modules have to be trained separately. Therefore, the training procedure is divided to 3 stages: the SR training stage, the refinement training stage and the joint training stage.

- Firstly, only the SR module is trained with learning rate 1×10^{-4} for about 200 epochs. The loss function is MAE, which is defined as

$$\mathrm{MAE}(I_{sr}, I_{gt}) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |I_{sr}(i,j) - I_{gt}(i,j)|. \tag{5.16}$$

- Then the parameters of the SR module are fixed. The refinement module is attached and trained with learning rate 1×10^{-4} for about 50 epochs. The loss function remains MAE.

- Finally, the whole network is trained jointly with learning rate 1×10^{-5} for 150 epochs. The loss function is changed to MSE, as defined below.

$$\mathrm{MSE}(I_{sr}, I_{gt}) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I_{sr}(i,j) - I_{gt}(i,j)]^2. \tag{5.17}$$

Figure 5.2 illustrates the PSNR changes during the training of the DLVSR. As shown, the addition of the refinement module significantly expands the network capacity and improves the training PSNR by about 0.5 dB. The F3, F5 and F7 in the figure denotes the results with 3, 5 and 7 input frames respectively. The `stage1` refers to the exclusive training of the SR module. The `stage2&3` refers to the exclusive training stage of the refinement module and the joint training stage. We can see that results with more input frames are constantly better than the ones with less input frames. This observation agrees with our theoretical inferences.

5.3.3 Training the VSR-CFE

The VSR-CFE is trained with a subset of the DLVSR training dataset. However, because there is a multi-scale up-sampler in this proposal, the data has to be preprocessed accordingly. We first randomly select a position in a frame sequence and use it as the top-left corner to crop patches. The patches are of 192×192, 288×288, 384×384 pixels, corresponding to $2\times$, $3\times$, $4\times$ scale factors. Figure 5.3 illustrates of the patch cropping strategy. The frame in the middle of the sequence is used as the HR target. Then all of the patches are down-sampled to 96×96 pixels with bilinear interpolation to serve as the LR input frames.

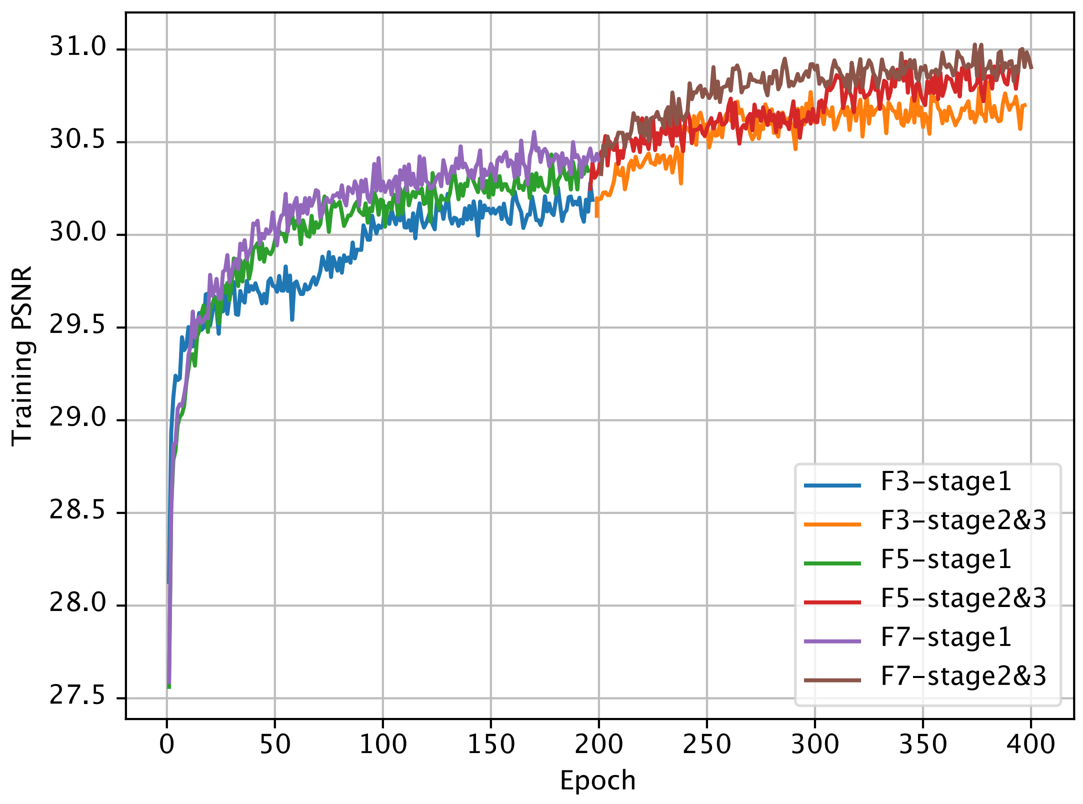


Figure 5.2: The PSNR changes during the training of the DLVSR.

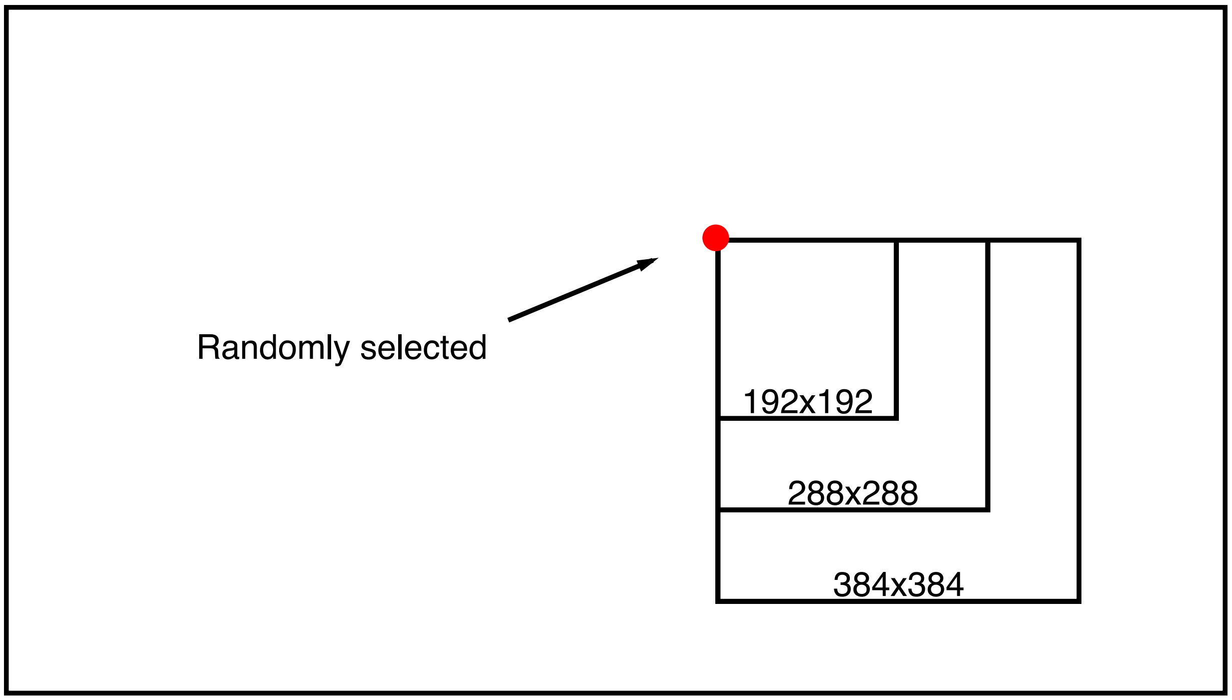


Figure 5.3: Cropping strategy.

To train the sampler for multiple scale factor simultaneously, we used an alternative sequence to input the training data. For every 3 iterations, patches for $2\times$, $3\times$, $4\times$ are fed to the network separately with a random sequence as follows as an example.

$$I^{(1)}_{\times 2}, I^{(1)}_{\times 4}, I^{(1)}_{\times 3}; I^{(2)}_{\times 3}, I^{(2)}_{\times 2}, I^{(2)}_{\times 4}; I^{(3)}_{\times 4}, I^{(3)}_{\times 2}, I^{(3)}_{\times 3}; \ldots$$

Thus three iterations of optimization can be taken on the network parameters per batch and accelerate the convergence.

For the VSR-CFE, we use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ which is the same as the DLVSR. The learning rate is set to 1.0×10^{-4} for all experiments. As we have discussed in Section 4.4, the VSR-CFE employs much less parameters than the DLVSR. Therefore it is possible for us to train the network with mini-batch size larger than 1.

Smith *et al.* discovered a phenomenon in their recent work [56] that the same test accuracies can be obtained by both decaying learning rate and increasing mini-batch size after the same number of training epochs. But increasing mini-batch size can lead to fewer parameter updates, resulting in greater parallelism and shorter training times. Inspired by this discovery, we initialized the mini-batch size as 1 and increased it to 2, 4, 8 and 16 every 50 epochs. The network is therefore trained for about 250 epochs. The loss function is the MAE. We also added another 20 epochs training with the MSE loss function to achieve higher PSNR. However, no significant improvement was observed in this procedure. The training PSNR changes are illustrated in Figure 5.4.

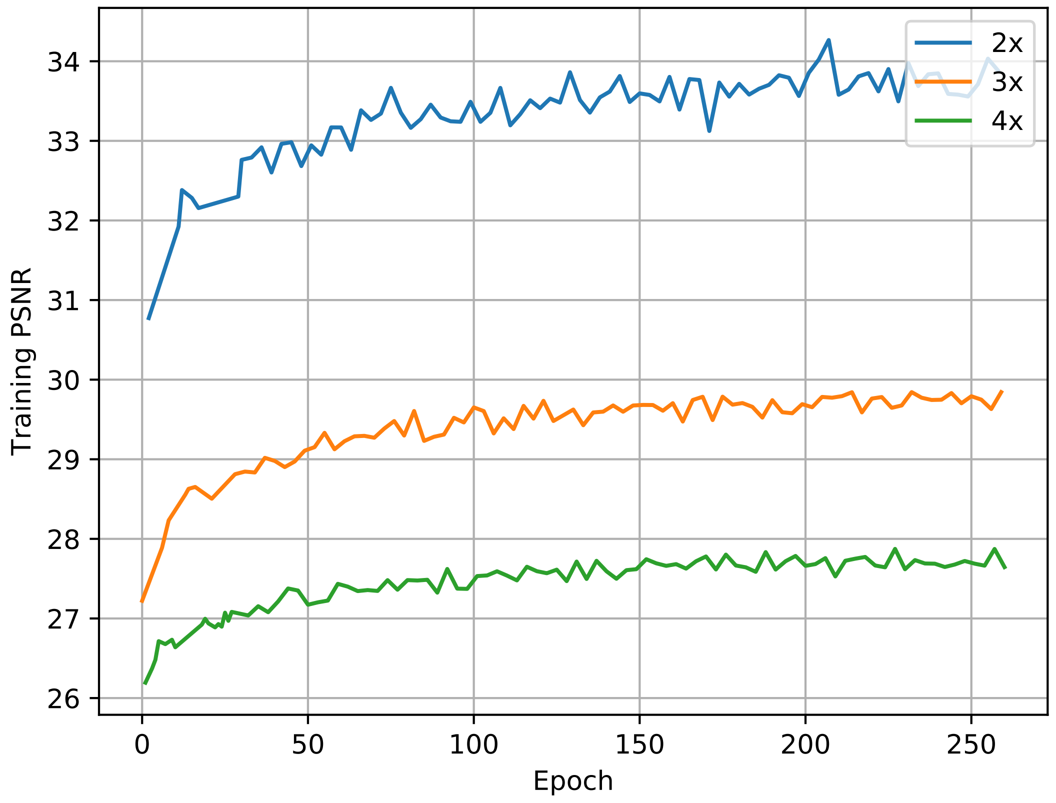


Figure 5.4: The PSNR changes during the training of the VSR-CFE.

5.4 Experimental results

5.4.1 Comparisons with other VSR methods

The quantitative comparisons with other VSR methods are shown in Table 5.3. It has to be noted that the BayesSR is tested with 30 adjacent frames and the others, including ours, are tested with 3. The data of the reference methods are from [58]. From the data, we can see that the DLVSR with refinement module outperforms the others by at least 0.86 dB at 4× scale factor. The VSR-CFE outperforms the other methods by at least 1.08 dB and 0.87 dB in terms of PSNR for 3× and 4× SR respectively. The visual comparisons are shown in Figure 5.5 through 5.8. For visual comfort, bicubic up-sampled chrominance channels are used with the super-resolved luminance channel. For *Calendar*, the inner edge of the roof is recovered by BayesSR and our methods, but lost by the others. Our methods achieve more smooth results. For *City*, our proposed method super-resolves the corner of the building with more accurate edges than the others. For *Foliage* and *Walk*, only our methods are able to recover the shape of the car in the back and the string on the outwear respectively.

5.4.2 Comparisons with input frames of various lengths

For the VSR tasks, the multiple input frames can be viewed as various degraded observations of the ground truth. A proper VSR method should be able to generate superior results when extra observations are provided. To prove that our proposed method can exploit information from additional input frames, we evaluate our methods with input frames of various lengths. The DLVSR is tested on 3, 5 and 7 input frames and the VSR-CFE is tested on 3 and 7 frames. The quantitative comparison is shown in Table 5.4, where we can see that the additional frames improve the results effectively for all scenarios. The visual comparison is illustrated in Figure 5.9 and 5.10, where the first row is for video *Calendar*

Table 5.3: Comparison with other VSR methods (PSNR / SSIM)

Method	3x	4x
BayesSR(F30)	25.64 / 0.80	24.42 / 0.72
VSRnet(F3)	26.64 / 0.82	22.81 / 0.65
VESPCN(F3)	27.25 / 0.84	25.35 / 0.76
SPMC(F3)	27.49 / 0.84	25.52 / 0.76
DLVSR(F3)	-	26.04 / 0.80
DLVSR-refine(F3)	-	26.38 / **0.81**
VSR-CFE(F3)	**28.57** / **0.89**	**26.39** / **0.81**

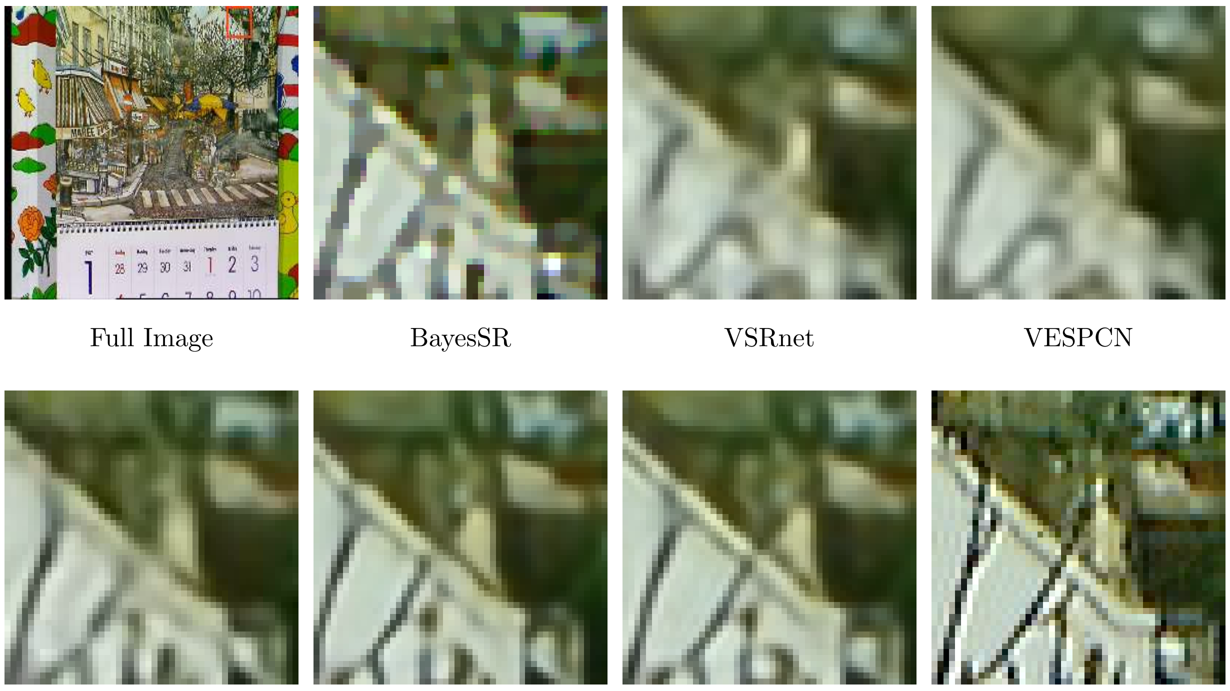


Figure 5.5: Visual comparison on *Calendar* with various VSR methods.

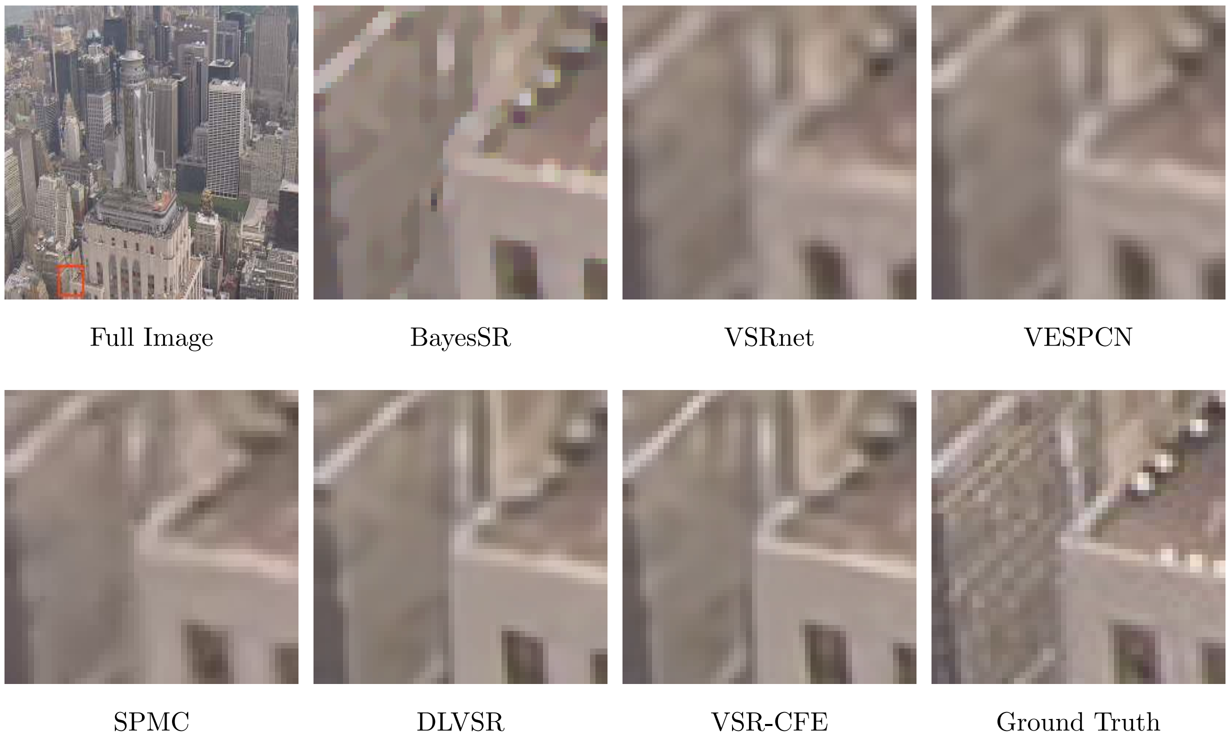


Figure 5.6: Visual comparison on *City* with various VSR methods.

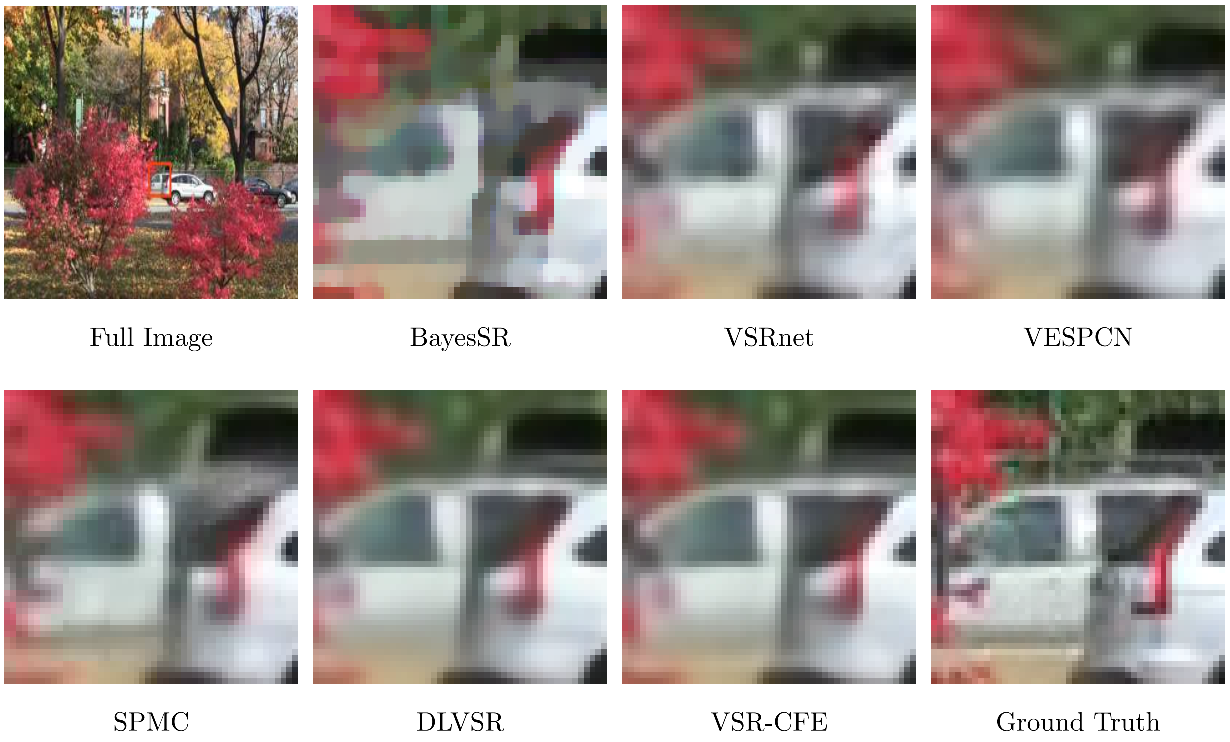


Figure 5.7: Visual comparison on *Foliage* with various VSR methods.

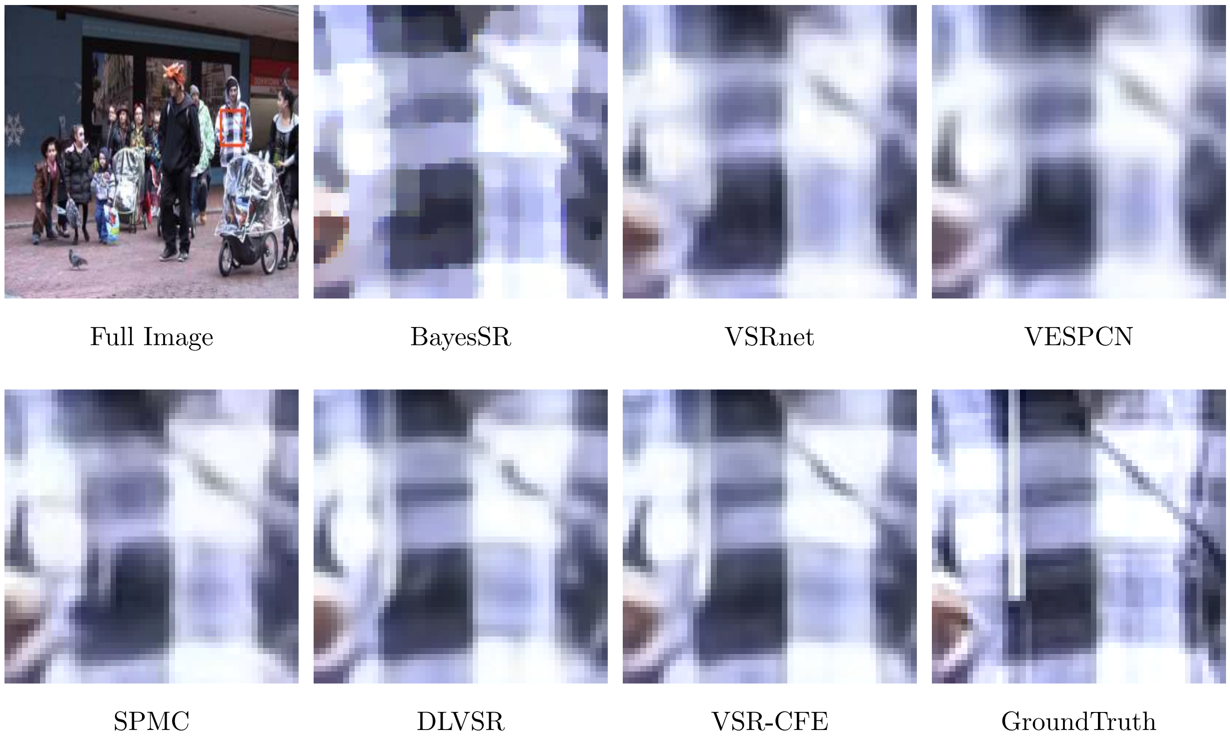


Figure 5.8: Visual comparison on *Walk* with various VSR methods.

and the second row is for video *Foliage*. For *Calendar*, we can see the texture of the wood wall in the result with more frames is clearer than the one with less frames. For *Foliage*, the separator between the two windows of the car is recovered in the 5-frame and 7-frame versions but lost in the 3-frame ones. Experimental results show that more input frames do lead to results with more details and sharper edges as well as higher PSNR/SSIM values.

Table 5.4: Comparison on results with input frames of various lengths (PSNR / SSIM)

Method	F3	F5	F7
DLVSR($\times$4)	26.04 / 0.80	26.17 / 0.81	26.20 / 0.81
DLVSR-refine($\times$4)	26.38 / **0.81**	26.51 / 0.82	26.52 / 0.82
VSR-CFE($\times$4)	**26.39** / **0.81**	-	**26.45** / **0.82**
VSR-CFE($\times$3)	**28.57** / **0.89**	-	**28.64** / **0.89**

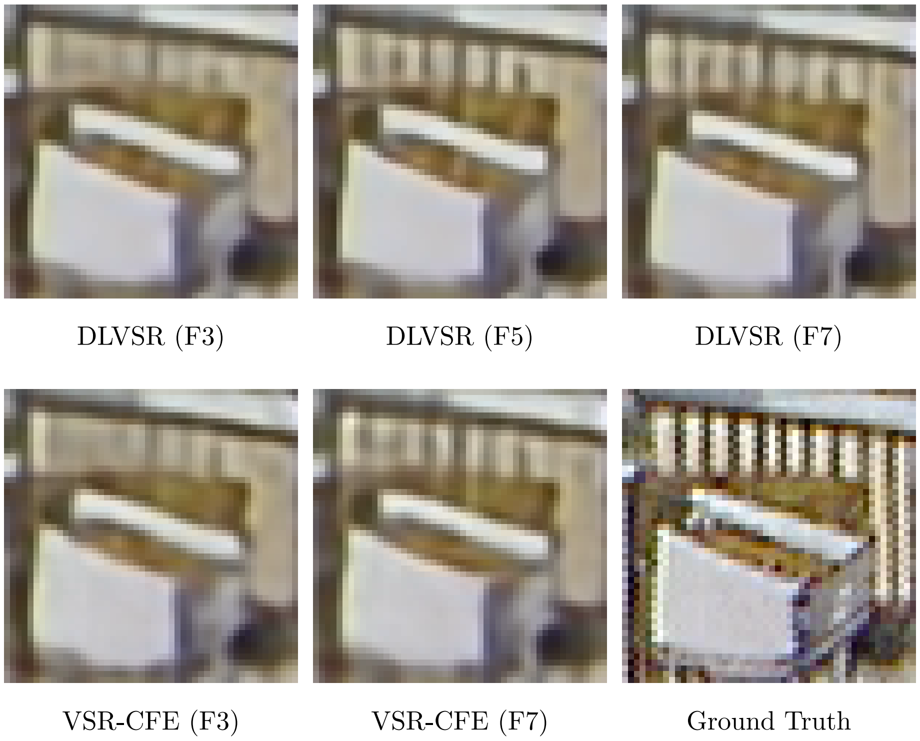


Figure 5.9: Visual comparison on *Calendar* with input frames of various length.

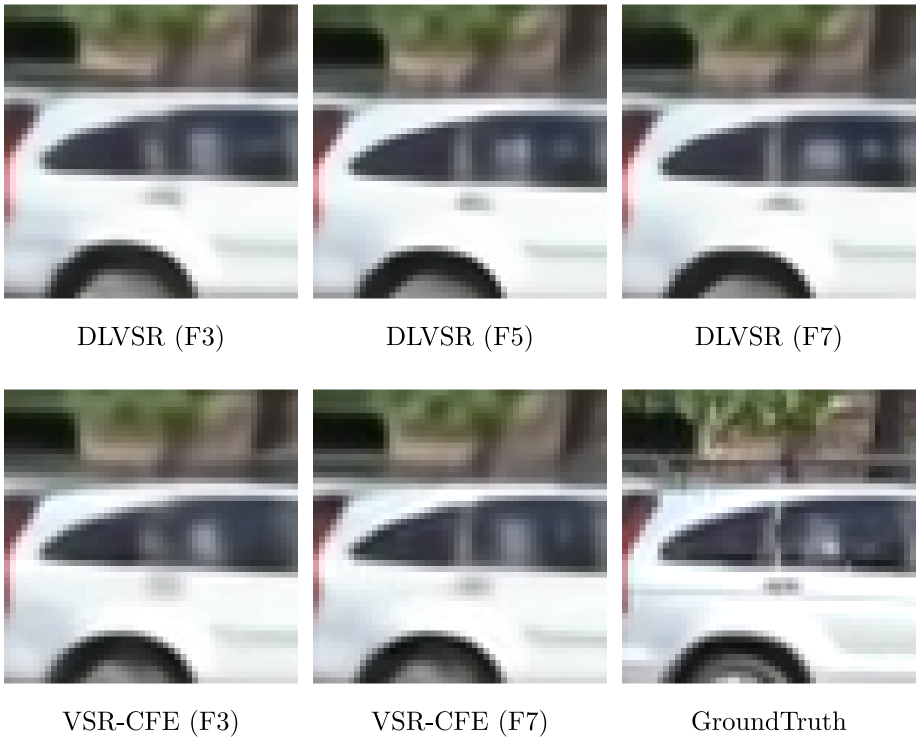


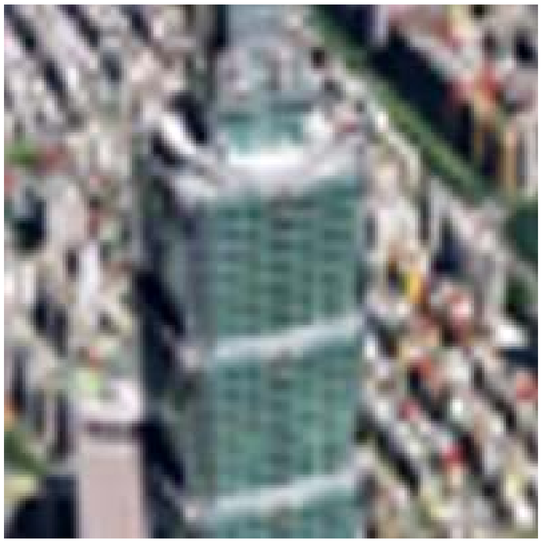
Figure 5.10: Visual comparison on *Foliage* with input frames of various length.

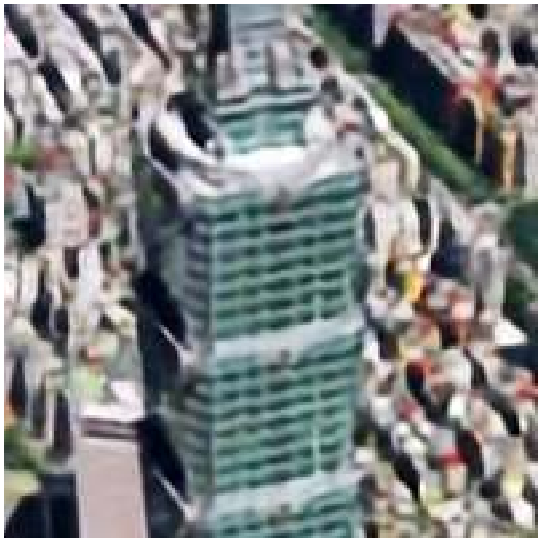
5.4.3 Evaluation with additional test data

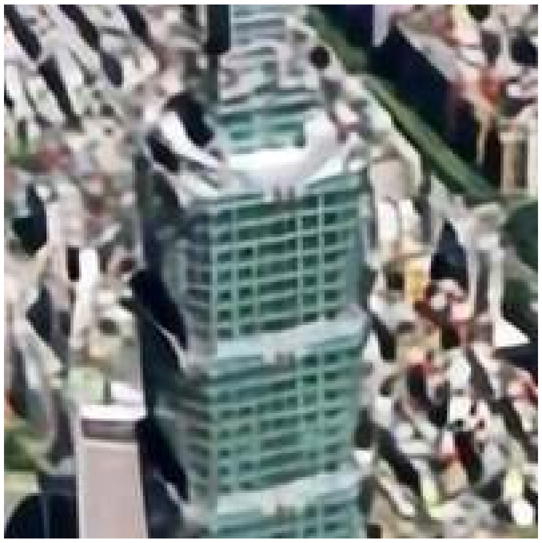
The VID4 video set, which was first introduced to VSR by Liu *et al.* [44] in 2011, is considered to be unable to represent modern videos whose quality and resolution have been improved significantly in recent years. Tao *et al.* [58] proposed several new test videos in their work to provide better estimations of the performance of their method SPMC. The test videos, together with their experimental results have been published on their website. We tested our method VSR-CFE on a part of these test videos and compared our results with the results of SPMC.

The newly proposed test video set consists of 15 video clips. 4 of them are the videos in VID4 and another 4 of them have been used as training data in our proposals. Therefore we tested our method on the rest 7 video clips, which are *hdclub_003_001*, *hitachi_isee5_001*, *HKVTG_004*, *jvc_009_001*, *NYVTG_006*, *PRVTG_012* and *RMVTG_011*. Each of these clips has 31 frames with 960 $\times$ 540 pixels. The results of SPMC are generated with 5 consecutive frames as the input. We followed this configuration and obtained 27 super-resolved frames for each clip. The visual and quantitative comparisons are illustrated in Figure 5.11 through 5.17. The PSNR and SSIM are measured on the total 27 frames of each test video.

As shown in the figures, our method has better results in terms of SSIM on 6 out of 7 clips. In terms of PSNR, our method outperforms the others on 3 out of 7 clips. Moreover, our results consistently have sharper edges and less artifacts than the others in visual comparisons.


Bicubic (PSNR 19.40 dB / SSIM 0.5346)


SPMC (PSNR 21.04 dB / SSIM 0.7185)


VSR-CFE (PSNR **21.10** dB / SSIM **0.7304**)

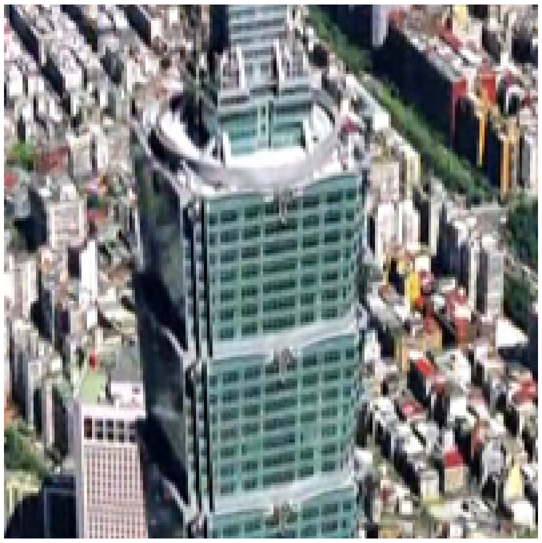
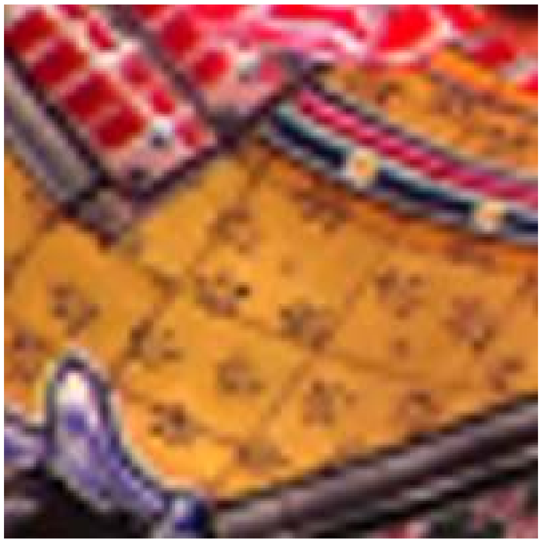
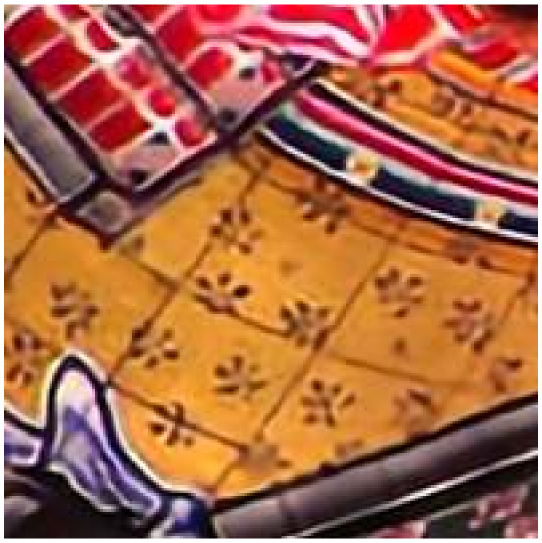

Ground Truth

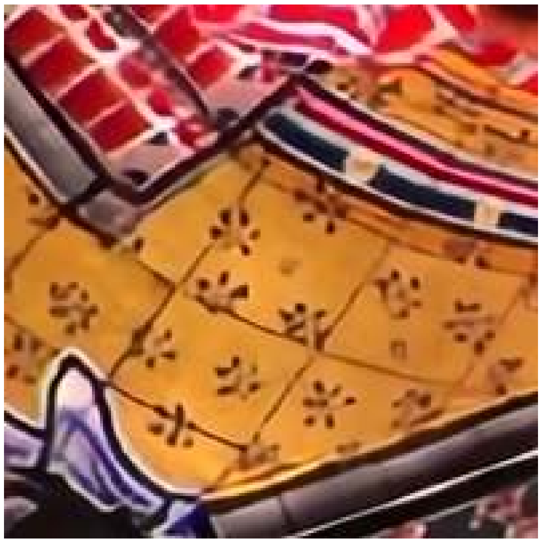
Figure 5.11: Comparison on video *hdclub_003_001*. The vertical textures on the building are better recovered by our method than the others.



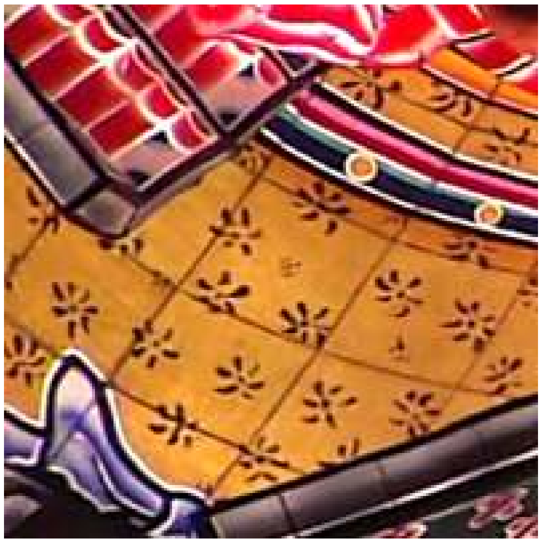
Bicubic (PSNR 19.60 dB / SSIM 0.6192)



SPMC (PSNR **23.75** dB / SSIM **0.8482**)

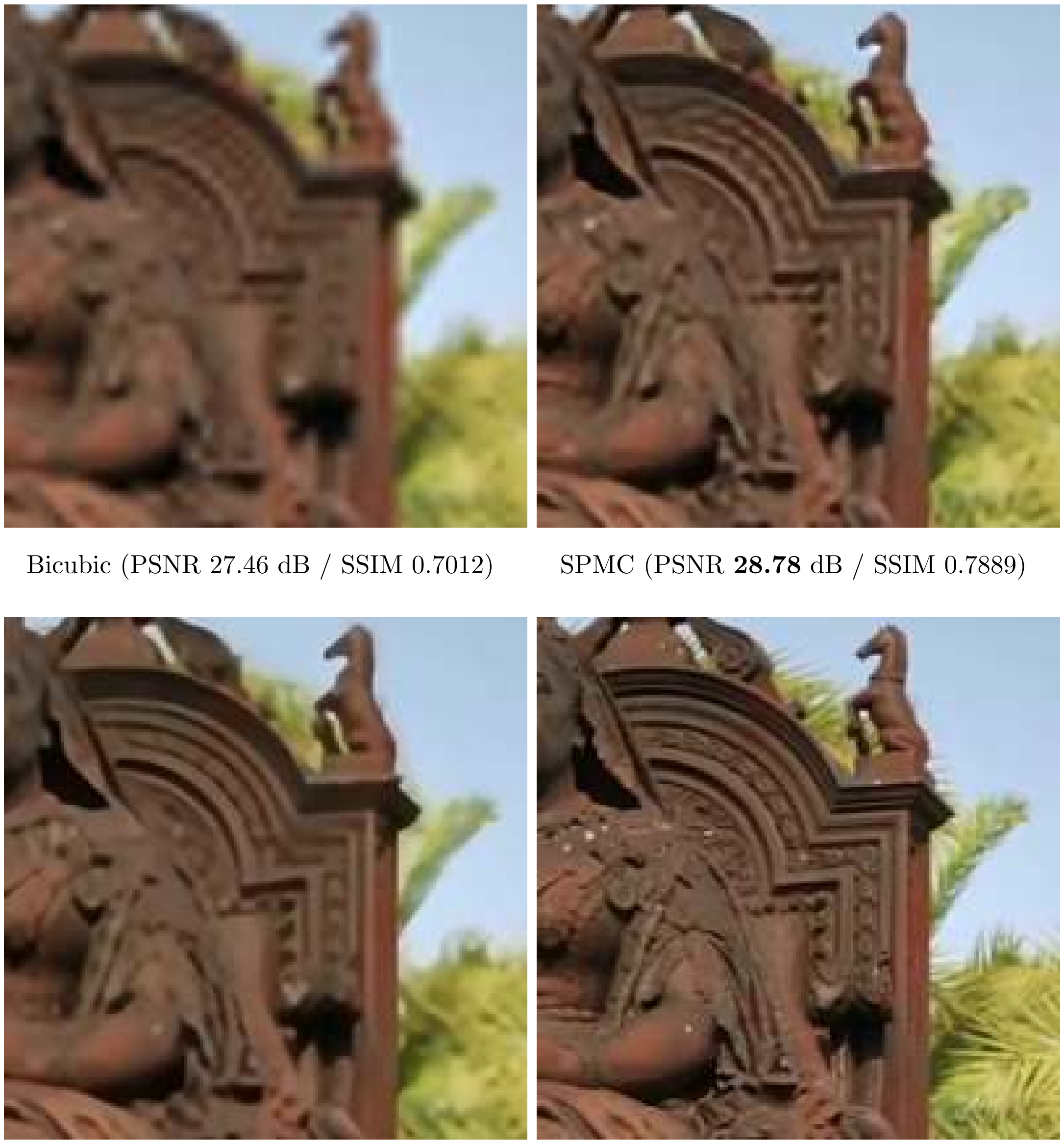


VSR-CFE (PSNR 22.76 dB / SSIM 0.8232)



Ground Truth

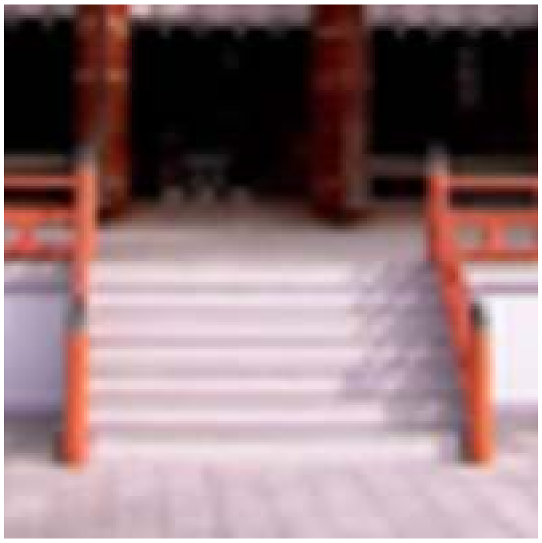
Figure 5.12: Comparison on video *hitachi_isee5_001*. The flower patterns are recovered with less artifacts by our method.



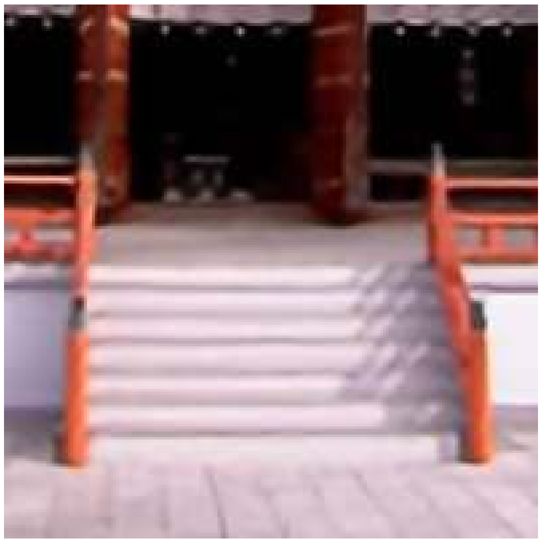
Bicubic (PSNR 27.46 dB / SSIM 0.7012) SPMC (PSNR **28.78** dB / SSIM 0.7889)

VSR-CFE (PSNR 28.67 dB / SSIM **0.7906**) Ground Truth

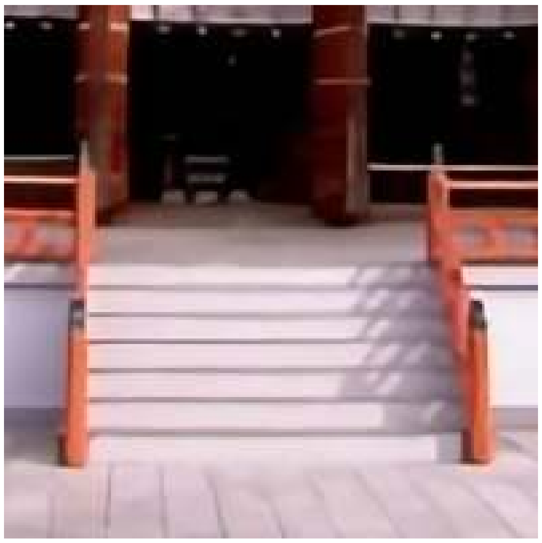
Figure 5.13: Comparison on video *HKVTG_004*. The curve on the back of the chair is more smooth in our result than the others.



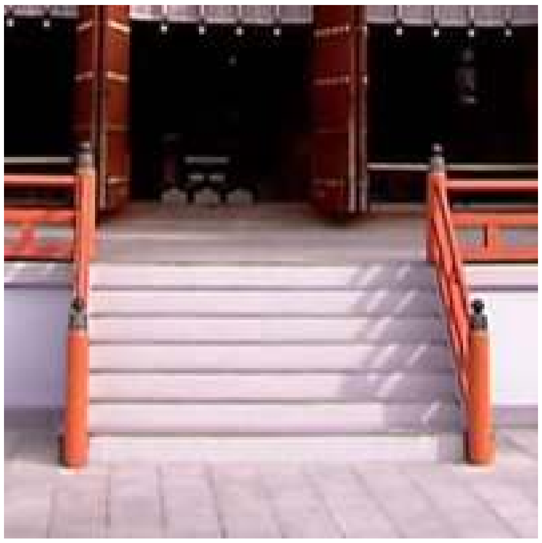
Bicubic (PSNR 25.40 dB / SSIM 0.7645)



SPMC (PSNR 28.21 dB / SSIM 0.8722)

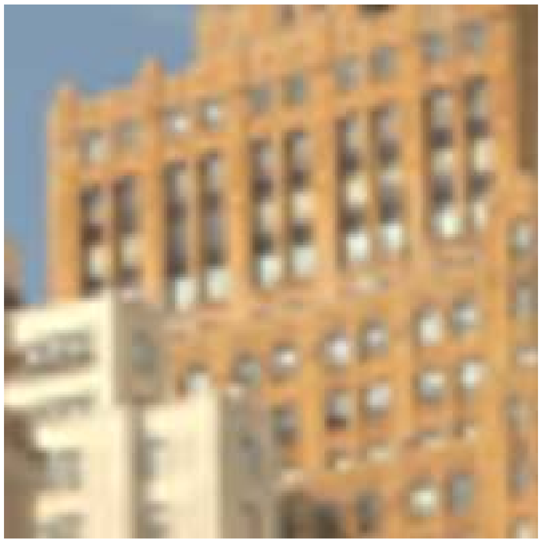


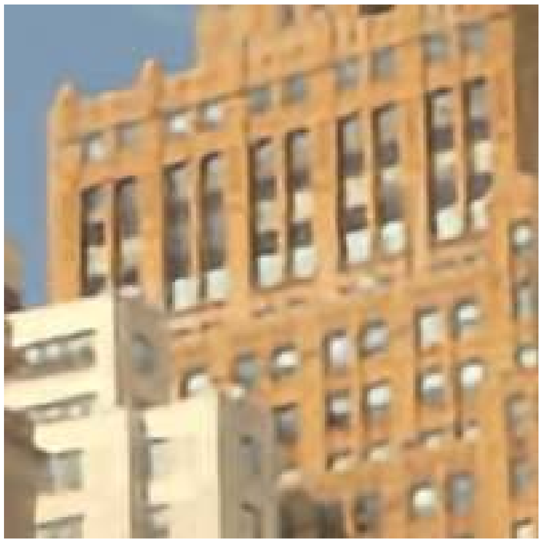
VSR-CFE (PSNR **28.30** dB / SSIM **0.8815**)

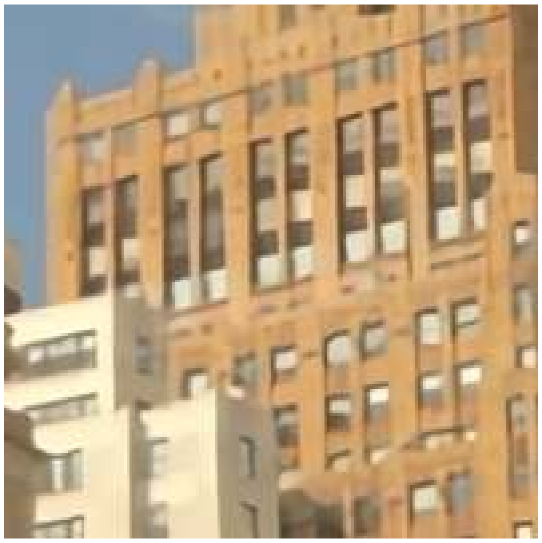


Ground Truth

Figure 5.14: Comparison on video *jvc_009_001*. The fence in the back is clearer in our result than the others.


Bicubic (PSNR 28.40 dB / SSIM 0.8119)


SPMC (PSNR **31.40** dB / SSIM 0.9019)


VSR-CFE (PSNR 31.28 dB / SSIM **0.9121**)

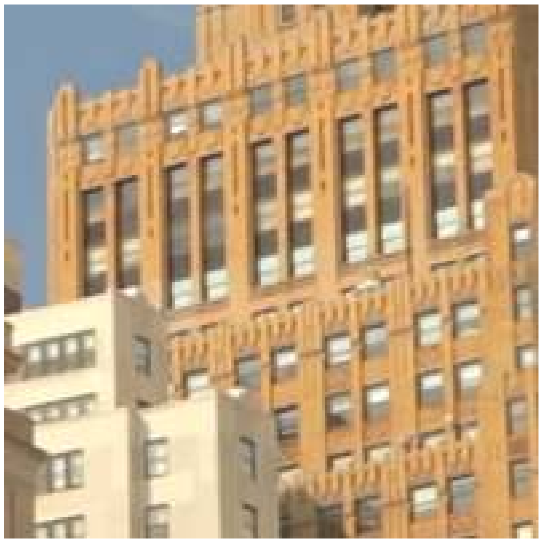
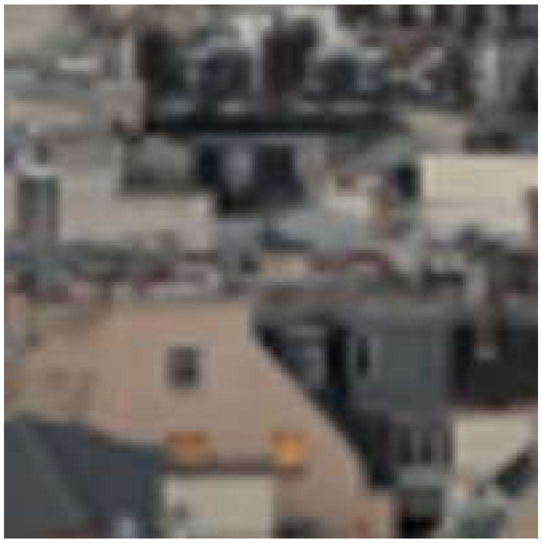
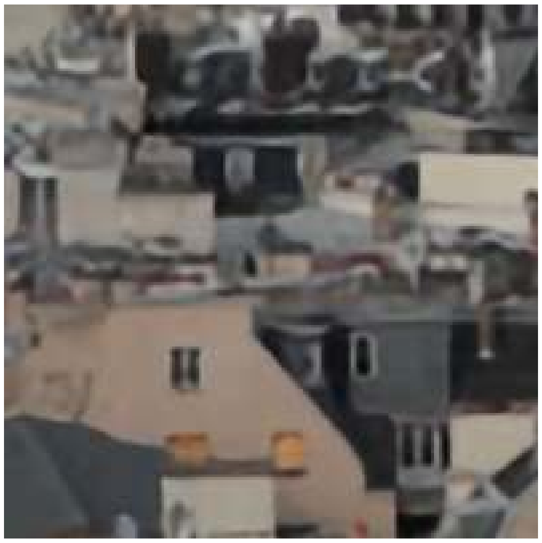

Ground Truth

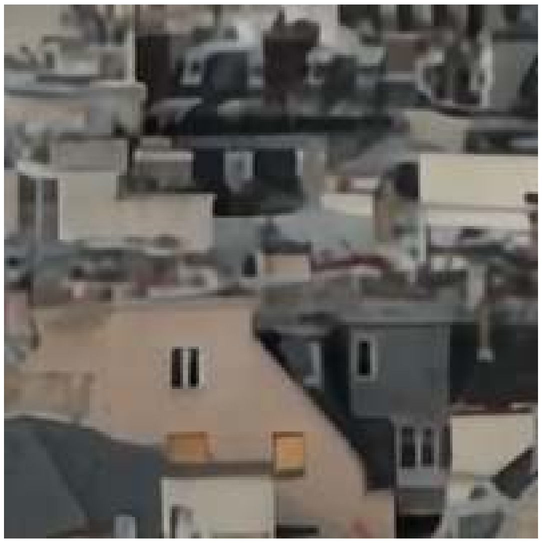
Figure 5.15: Comparison on video *NYVTG_006*. Windows have sharper edges in our result than the others.



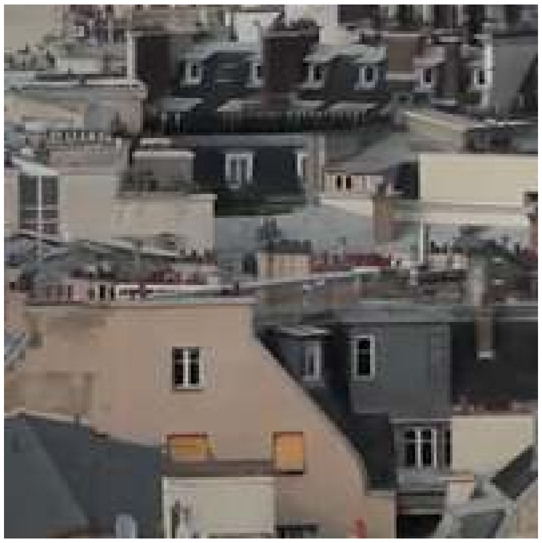
Bicubic (PSNR 25.54 dB / SSIM 0.7275)



SPMC (PSNR 27.00 dB / SSIM 0.8171)

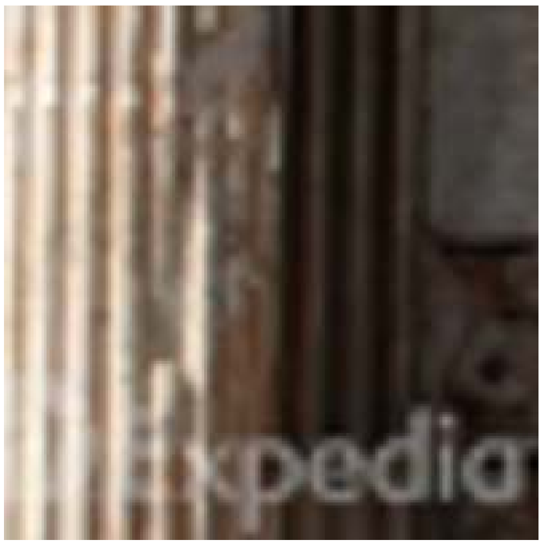


VSR-CFE (PSNR **27.04** dB / SSIM **0.8222**)

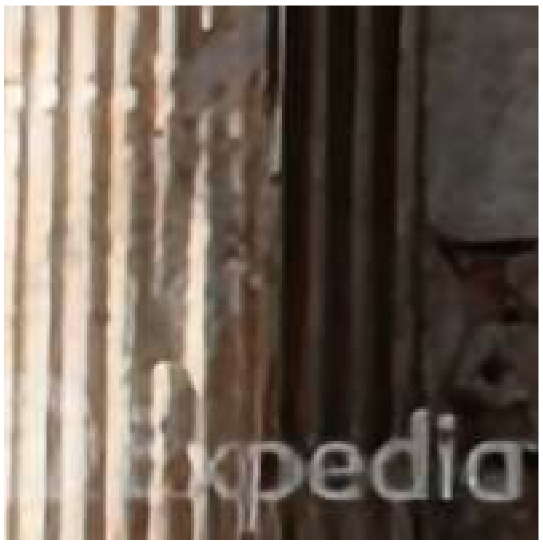


Ground Truth

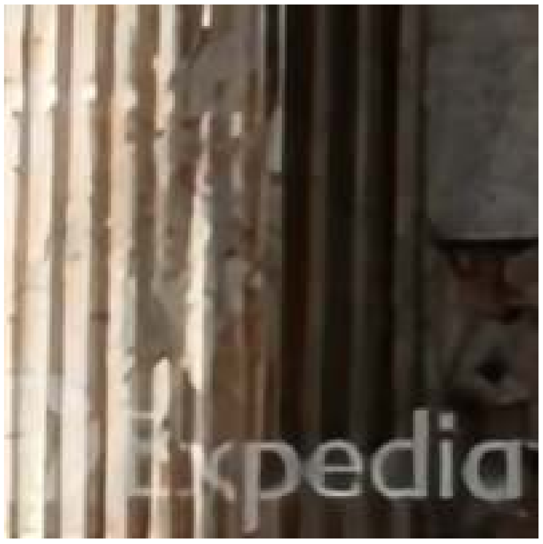
Figure 5.16: Comparison on video *PRVTG_012*. The windows in our result have clear square shapes.



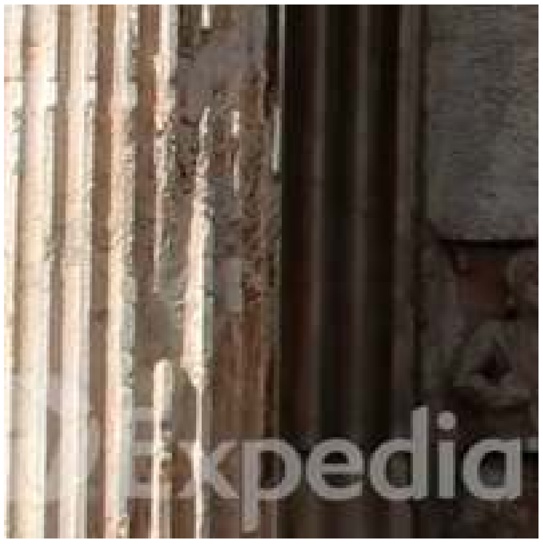
Bicubic (PSNR 23.99 dB / SSIM 0.6835)



SPMC (PSNR **26.43** dB / SSIM 0.8018)



VSR-CFE (PSNR 26.39 dB / SSIM **0.8037**)



Ground Truth

Figure 5.17: Comparison on video *RMVTG_011*. The text 'Expedia' is better recovered by our method than the others.

Chapter 6

Conclusion

Neural network techniques open numerous new possibilities for solving the video super-resolution (VSR) problem. In this thesis, we first introduced basic concepts of the neural network and reviewed several recent methods on single image super-resolution (SISR) and VSR. Then we proposed our own methods.

Motion compensation is commonly used in recent works to cancel the inconsistencies among the input frames. Different with other methods which manipulate pixels to compensate the motion, we proposed to cancel the motion during feature extraction procedure. We employed a dynamic local filter network as the motion compensation module. The module generates sample-specific filter kernels and applies the kernels via a position-specific operation. Based on this module, we proposed two VSR networks. The first one is DLVSR which compensates and super-resolves the input frames in a progressive manner and then refines the result with an autoencoder based module. The second one is VSR-CFE which is an upgrade of the DLVSR. It compensates the motion during feature extraction in LR space and generate super-resolved result with a residual block based module. Multi-scale SR structure is also introduced to the VSR-CFE to make the network work for multiple scale factors simultaneously. In our experimental results, both proposed methods outperform the state-of-the-art methods in terms of PSNR and SSIM and generate superior results over the others in visual comparison. As an upgrade edition, the VSR-CFE achieves slightly better results than the DLVSR while employs about 40% parameters and 30% training samples.

Traditional image processing algorithms commonly estimate specific parameters based on the samples provided. However the parameters of convolutional neural network (CNN) are mostly fixed after being trained. This difference potentially results in the fact that CNNs usually introduce much more parameters than traditional algorithm to handle a large number of various situations. The dynamic local filter network reduces the number of required parameters and keeps the superior modeling capability of CNN simultaneously. It has the sample-specific and position-specific properties together with the power of CNN. In this thesis, we proved that the dynamic local filter network works well for VSR tasks. More applications of it are still left to be explored in the future work.

References

[1] Consumer Digital Video Library, Oct 2016. `https://www.nist.gov/ctl/pscr/consumer-digital-video-library`.

[2] Free 4K Demo Footage - Ultra HD Demo Footage, May 2017. `https://www.harmonicinc.com/free-4k-demo-footage/`.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, 2016.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[5] C. Bishop, A. Blake, and B. Marthi. Super-Resolution Enhancement of Video. In *The International Conference on Artificial Intelligence and Statistics*, 2003.

[6] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings in Computational Statistics*, pages 177–186. Springer, 2010.

[7] J. Caballero, C. Ledig, A. Aitken, A. Acosta, J. Totz, Z. Wang, and W. Shi. Real-time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2848–2857, July 2017.

[8] D. Capel and A. Zisserman. Super-Resolution Enhancement of Text Image Sequences. In *Proceedings 15th International Conference on Pattern Recognition (ICPR)*, volume 1, pages 600–605 vol.1, 2000.

[9] B. Castellano. Breakthrough/PySceneDetect, Sep 2017. `https://github.com/Breakthrough/PySceneDetect`.

[10] A. Chakrabarti, A. N. Rajagopalan, and R. Chellappa. Super-Resolution of Face Images Using Kernel PCA-Based Prior. *IEEE Transactions on Multimedia*, 9(4):888–892, June 2007.

[11] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*, 2014.

[12] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, July 2017.

[13] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units. In *The International Conference on Learning Representations (ICLR)*, 2016.

[14] Z. Cui, H. Chang, S. Shan, B. Zhong, and X. Chen. Deep Network Cascade for Image Super-Resolution. In *European Conference on Computer Vision (ECCV)*, pages 49–64. Springer, 2014.

[15] H. Demirel and G. Anbarjafari. Discrete Wavelet Transform-Based Satellite Image Resolution Enhancement. *IEEE Transactions on Geoscience and Remote Sensing*, 49(6):1997–2004, June 2011.

[16] H. Demirel and G. Anbarjafari. Image Resolution Enhancement by Using Discrete and Stationary Wavelet Decomposition. *IEEE Transactions on Image Processing*, 20(5):1458–1460, May 2011.

[17] H. Demirel, S. Izadpanahi, and G. Anbarjafari. Improved Motion-Based Localized Super Resolution Technique Using Discrete Wavelet Transform for Low Resolution Video Enhancement. In *2009 17th European Signal Processing Conference*, pages 1097–1101, Aug 2009.

[18] C. Dong, C. Loy, K. He, and X. Tang. Learning a Deep Convolutional Network for Image Super-Resolution. In *European Conference on Computer Vision (ECCV)*, pages 184–199. Springer, 2014.

[19] C. Dong, C. C. Loy, K. He, and X. Tang. Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, Feb 2016.

[20] M. Drulea and S. Nedevschi. Total Variation Regularization of Local-Global Optical Flow. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 318–323, Oct 2011.

[21] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[22] G. Freedman and R. Fattal. Image and Video Upscaling from Local Self-Examples. *ACM Transactions on Graphics (TOG)*, 30(2):12, 2011.

[23] D. Glasner, S. Bagon, and M. Irani. Super-Resolution from a Single Image. In *2009 IEEE 12th International Conference on Computer Vision*, pages 349–356, Sept 2009.

[24] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[25] I. Goodfellow, J. Pouget-Abadien, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[26] H. Greenspan. Super-Resolution in Medical Imaging. *The Computer Journal*, 52(1):43–63, 2008.

[27] B. K. Gunturk, A. U. Batur, Y. Altunbasak, M. H. Hayes, and R. M. Mersereau. Eigenface-Domain Super-Resolution for Face Recognition. *IEEE Transactions on Image Processing*, 12(5):597–606, May 2003.

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

[29] P. H. Hennings-Yeomans, S. Baker, and B. V. K. V. Kumar. Simultaneous Super-Resolution and Feature Extraction for Recognition of Low-Resolution Faces. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[30] J. B. Huang, A. Singh, and N. Ahuja. Single Image Super-Resolution from Transformed Self-Exemplars. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5197–5206, June 2015.

[31] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial Transformer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[32] K. Jia and S. Gong. Generalized Face Super-Resolution. *IEEE Transactions on Image Processing*, 17(6):873–886, June 2008.

[33] A. Kappeler, S. Yoo, Q. Dai, and A. K. Katsaggelos. Video Super-Resolution with Convolutional Neural Networks. *IEEE Transactions on Computational Imaging*, 2(2):109–122, June 2016.

[34] T. Kasetkasem, M. Arora, and P. Varshney. Super-Resolution Land Cover Mapping Using a Markov Random Field Based Approach. *Remote Sensing of Environment*, 96(3):302 – 314, 2005.

[35] J. Kim, J. K. Lee, and K. M. Lee. Accurate Image Super-Resolution Using Very Deep Convolutional Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1646–1654, June 2016.

[36] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *The International Conference on Learning Representations (ICLR)*, 2015.

[37] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[38] C. Lanczos. Trigonometric Interpolation of Empirical and Analytical Functions. *Studies in Applied Mathematics*, 17(1-4):123–199, 1938.

[39] Y. LeCun and Y. Bengio. Convolutional Networks for Images, Speech, and Time Series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[40] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, July 2017.

[41] F. Li, X. Jia, D. Fraser, and A. Lambert. Super Resolution for Remote Sensing Images Based on a Universal Hidden Markov Tree Model. *IEEE Transactions on Geoscience and Remote Sensing*, 48(3):1270–1278, March 2010.

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

[43] F. Lin, C. Fookes, V. Chandran, and S. Sridharan. Investigation into Optical Flow Super-Resolution for Surveillance Applications. 2005.

[44] C. Liu and D. Sun. A Bayesian Approach to Adaptive Video Super Resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 209–216, June 2011.

[45] C. Mancas-Thillou and M. Mirmehdi. Super-Resolution Text Using the Teager Filter. pages 10 – 16, 8 2005. First International Workshop on Camera-Based Document Analysis and Recognition.

[46] X. Mao, C. Shen, and Y. Yang. Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections. In D. D. Lee,

M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2802–2810. Curran Associates, Inc., 2016.

[47] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[48] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, Haifa, Israel, June 2010. Omnipress.

[49] S. Niklaus, L. Mai, and F. Liu. Video Frame Interpolation via Adaptive Convolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2270–2279, July 2017.

[50] S. Niklaus, L. Mai, and F. Liu. Video Frame Interpolation via Adaptive Separable Convolution. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 261–270, Oct 2017.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in PyTorch. In *Advances in Neural Information Processing Systems 31*, 2017.

[52] M. Robinson, S. Chiu, J. Lo, C. Toth, J. Izatt, and S. Farsiu. New Applications of Super-Resolution in Medical Imaging. *Super-Resolution Imaging*, 2010:384–412, 2010.

[53] R. R. Schultz. Super-Resolution Enhancement of Native Digital Video versus Digitized NTSC Sequences. In *Proceedings Fifth IEEE Southwest Symposium on Image Analysis and Interpretation*, pages 193–197, 2002.

[54] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-

Pixel Convolutional Neural Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, June 2016.

[55] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *The International Conference on Learning Representations (ICLR)*, 2015.

[56] S. Smith, P. Kindermans, and Q. Le. Don't Decay the Learning Rate, Increase the Batch Size. In *International Conference on Learning Representations (ICLR)*, 2018.

[57] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, pages III–1139–III–1147. JMLR.org, 2013.

[58] X. Tao, H. Gao, R. Liao, J. Wang, and J. Jia. Detail-Revealing Deep Video Super-Resolution. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4482–4490, Oct 2017.

[59] A. J. Tatem, H. G. Lewis, P. M. Atkinson, and M. S. Nixon. Super-Resolution Target Identification from Remotely Sensed Images Using a Hopfield Neural Network. *IEEE Transactions on Geoscience and Remote Sensing*, 39(4):781–796, Apr 2001.

[60] M. Thornton, P. Atkinson, and DA Holland. Sub-Pixel Mapping of Rural Land Cover Objects from Fine Spatial Resolution Satellite Sensor Imagery Using Super-Resolution Pixel-Swapping. *International Journal of Remote Sensing*, 27(3):473–491, 2006.

[61] T. Tijmen and H. Geoffrey. Lecture 6.5-RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[62] Y. Toki, Y. Fujisawa, and M. Kato. Super-Resolution Processor and Medical Diagnostic Imaging Apparatus, 2009. US Patent 7,492,967.

[63] D. H. Trinh, M. Luong, F. Dibos, J. M. Rocchisani, C. D. Pham, and T. Q. Nguyen. Novel Example-Based Method for Super-Resolution and Denoising of Medical Images. *IEEE Transactions on Image Processing*, 23(4):1882–1895, April 2014.

[64] Y. H. Wang, J. B. Li, and P. Fu. Medical Image Super-Resolution Analysis with Sparse Representation. In *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 106–109, July 2012.

[65] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[66] Z. Wang, D. Liu, J. Yang, W. Han, and T. Huang. Deep Networks for Image Super-Resolution with Sparse Prior. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 370–378, Dec 2015.

[67] J. Xu, D. B. Bert, T. Tinne, and G. Luc V. Dynamic Filter Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 667–675. Curran Associates, Inc., 2016.

[68] J. Yang, Z. Lin, and S. Cohen. Fast Image Super-Resolution Based on In-Place Example Regression. In *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1059–1066, June 2013.

[69] J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image Super-Resolution via Sparse Representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, Nov 2010.

[70] L. Yann, B. Léon, O. Genevieve, and M. Klaus-Robert. Efficient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[71] L. Zhang, H. Zhang, H. Shen, and P. Li. A Super-Resolution Reconstruction Algorithm for Surveillance Images. *Signal Processing*, 90(3):848 – 859, 2010.