Deep Autoencoder for Light Field Super-Resolution

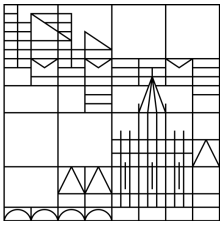
Master Thesis

presented by

Minchen Zhu

at the

Universität Konstanz



Faculty of Sciences

Department of Computer and Information Science

1. **Reviewer:** Prof. Dr. Bastian Goldlücke
2. **Reviewer:** Prof. Dr. Oliver Deussen

Konstanz, 2019

Abstract

When capturing a light field of a scene, one typically faces a trade-off between more spatial or more angular resolution. Fortunately, light fields are also a rich source of information for solving the problem of super-resolution. Contrary to single image approaches, where high-frequency content has to be hallucinated to be the most likely source of the downscaled version, sub-aperture views from the light field can help with an actual reconstruction of those details that have been removed by downsampling. In this work, we propose a three-dimensional generative adversarial autoencoder network to recover the high-resolution light field from its low-resolution version with a sparse set of viewpoints. We require only three views along both horizontal and vertical axis to increase angular resolution by a factor of three while at the same time increasing spatial resolution by a factor of either two or four in each direction, respectively.

Contents

Chapter 1

Introduction

The problem of super-resolution, where one wants to recover a high-resolution image from one or more low-resolution versions, is one of the central challenges in computer vision. Typically, it is a highly ill-posed problem, so the classical approach which solves it as an inverse problem requires carefully constructed image priors [1]. It is well known that having multiple low-resolution images with slightly different viewpoints is both crucial as well as successful in recovering sharp details in the reconstructed image [2–5].

Such a dense collection of viewpoints is described by the light field of a scene, a four-dimensional structure which parametrizes a set of captured rays by their point of origin in the focal plane (representing the viewpoint), and their intersection coordinate with an image plane. Dedicated light field cameras efficiently acquire dozens of views by multiplexing the rays onto a single sensor, or using multiple standard industrial cameras in an array. Compared to stereo imaging, light fields are more redundant and provide richer information about the scene, furthermore, the baseline is typically much smaller. However, there typically is a conflict between either having more views (angular resolution) or more spatial resolution within the views. Light field super-resolution, therefore, aims at increasing both spatial and angular resolution, where the latter amounts to generating novel views of the scene.

Optimization-based approaches [2, 6, 7] can provide results of good quality, but their parameters usually need to be fine-tuned, and they often require much time to converge. Recently, however, inverse problems have been successfully attacked by just brute-force learning a prediction of the desired result given the observed inputs. This is due to the rapid development of deep learning techniques in the last decade, which is made possible by a dramatic increase in the computational power of GPUs and the availability of large amounts of training data. Architectures based on deep encoder-decoder convolutional neural networks (CNN) turned out to be very powerful for super-resolution even of single images [8, 9], where they zoom a single low-resolution image to impressive high-resolution quality. However, fine detail is necessarily hallucinated and reproduced from natural image statistics, as the downsampling removes the high-frequency content. Multiple input images are required to be able to really recover a truthful high-resolution image.

Deep learning methods are nowadays also popular in the light field community. In recent work, the rich information inherent in the light field was successfully used to build deep networks for diverse tasks such as disparity estimation, view synthesis, reflection separation and intrinsic image decomposition [10–12]. The super-resolution problem was also addressed with recent CNN architectures [3, 13]. Although these approaches

give impressive results, we will discuss in our paper that there is still lots of room for improvement.

One of the biggest challenges in light field super-resolution is the high dimensionality of the problem. The light field itself is a 4D data structure, its upscaled version is two or four times larger and it requires enormous amounts of GPU memory to process it with adequate batch size and a sufficient number of feature maps. Thus, the plain architecture where the input image is upscaled to the desired resolution and then refined with a convolutional network is not applicable to light fields. Another challenge is that the light field is 4-dimensional and cannot be directly used in the current deep learning frameworks where convolution operations are performed only in 2-dimensional and 3-dimensional spaces. Finally, capturing a ground truth light field of good quality for training is a difficult task, since for example, the consumer Lytro Illum plenoptic camera produces blurry images that suffer from noise, while the higher quality Raytrix camera is of plenoptic type 2.0, and thus sub-aperture views are not available. A possibility is to use a high-quality camera mounted on a gantry, which is a time-consuming approach that can hardly be performed in the wild, and is only applicable to static scenes, so training data will necessarily be limited.

Content arrangement The following content of this thesis consists of 6 chapters. In chapter 2, we present the basics of the light fields, including the definition, representations, light field acquisition methods and applications. As we solve the light field super-resolution problem by utilizing deep learning, a theoretical introduction with implementation in TensorFlow is given in chapter 3. In the related work, we briefly introduce the literature for super-resolution algorithms and highlight the most helpful CNN-based methods to our approach. Chapter 5 describes our preliminary experiments, in which a simple convolutional autoencoder is built to generate the spatially upscaled center view of the light field. The influence of the color spaces on the image restoration problem by CNN-based approach is analysed in detail as well. Furthermore, we visually identified the checkerboard artifacts caused by the strided 2D convolution transpose in the output of this network, thus avoid using it for upscaling in the proposed approach. We present a detailed introduction of our deep autoencoder for light field super-resolution in chapter 6. There we mathematically describe our model and depict the architecture of the network, followed by the report of the results numerically as well as visually. Finally chapter 7 summarizes the thesis and gives our conclusions.

Contributions This master thesis is based on a conference submission with co-authors Anna Alperovich, Ole Johannsen and Prof. Dr. Bastian Goldlücke [14]. We address the problem of light field super-resolution, i.e. obtaining a light field with larger spatial and angular resolution from only five sub-aperture views of a low-resolution light field. The proposed approach uses information from neighbouring views and benefits from the dense redundant structure of the light field. A fully convolutional asymmetrical encoder-decoder is built as the first network architecture to transform the 4D structure of the light field to its upscaled version. To enhance the sharpness of the reconstructed light field, we propose a novel GAN loss that penalizes the difference between angular and spatial derivatives of the generated light field and its ground truth. Contrary to the original GAN

based architectures for super-resolution [15], where the discriminator takes a generated high-resolution image and the ground truth, we feed the discriminator network also with derivative information, which is much more simple and sparse than the original light field since it contains only edge information. This design choice makes GAN training very stable and avoids unexpected distortions we have otherwise observed. We perform both spatial and angular super-resolution with scale factors two and four, given only a sparse set of sub-aperture views. Our network achieves results with the competitive quality for both artificial and real-world light fields compared to state-of-the-art conventional methods and single image SRGAN.

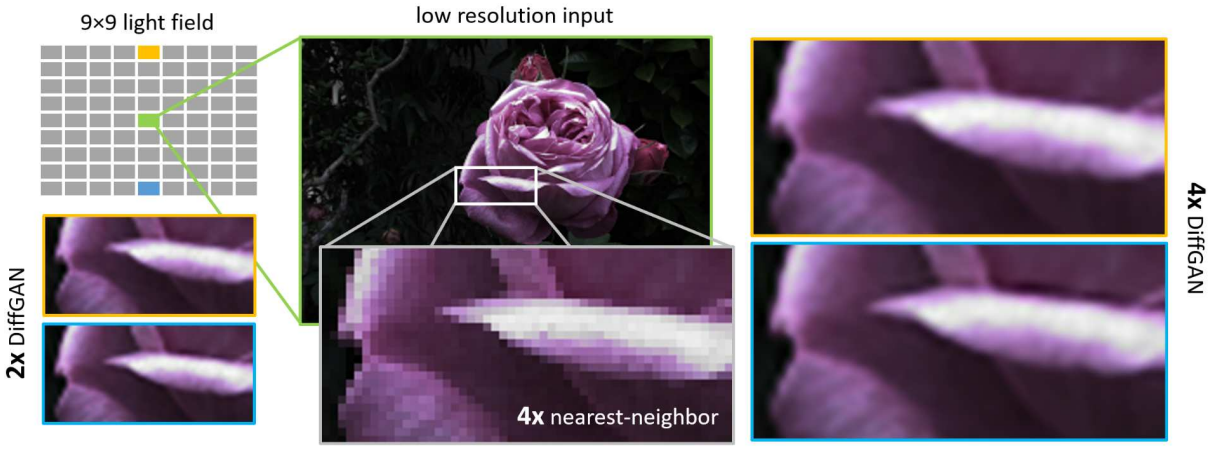


Figure 1.1: ***Deep autoencoder for light field super-resolution.*** *[14] We take 3 views in the horizontal and vertical epipolar volumes from a 9×9 low-resolution light field as the input to our encoder-decoder network. It super-resolves the input to the full size of the original epipolar volumes and upscales each view spatially 2x or 4x. With the proposed DiffGAN, a significant improvement of the image quality can be obtained. Compared to the naive nearest neighbour interpolation, our approach provides more natural and smoother outputs.*

Chapter 2

Light Field

2.1. Theoretical background

Michael Faraday is the first one who proposed that light should be interpreted as a field [16]. Arun Gershun [17] firstly defines the concept of the light field. There he characterized the light ray by calculus and analytic geometry. In geometrical optics, a ray is recognized as the fundamental representation of light. The radiance of all rays in the 3D region under a constant illumination arrangement is referred to the plenoptic function. As Fig. 2.1 shows, the rays in a 3D space can be parametrized by coordinates (x, y, z) and angles (θ, ϕ). The coordinates show the position where the light ray arrives and the angles indicate the horizontal and vertical angles entering this location. Therefore the plenoptic function is a 5D function:

$$L(x, y, z, \theta, \phi) \tag{2.1}$$

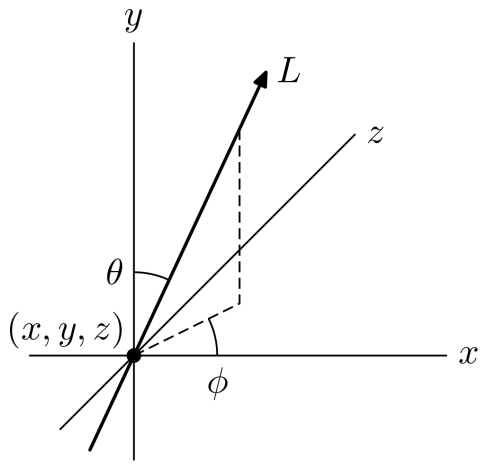


Figure 2.1: *A ray in the 3D space can be parametrized by the arriving position (x,y,z) and the angle (θ,ϕ).*

This function is later perfected by E. H. Adelson and J. R. Bergen in their paper [18] to a 7D function:

$$L(x, y, z, \theta, \phi, \lambda, t) \tag{2.2}$$

describing the light with wavelength λ passing through a point (x, y, z) in the 3D space along the direction (θ, ϕ) at the time t. In practice, the color and time information are usually represented by color channels and different frames. Assume the color of the light is not changing during the propagation and we only observe this ray at the time point t, the function can be reduced from 7 back to 5 dimensions again. Furthermore, in most imaging systems, light is propagating along a limited optical path, so a simpler representation of light field is introduced, shown by Fig. 2.2:

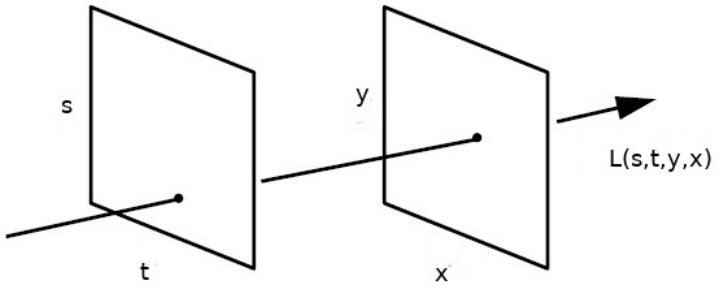


Figure 2.2: *The two-plane parametrization uses the ray and the focal coordinates of two parallel planes to represent a light ray in the light field.*

At this point, a light field can be defined on 4D ray space $\mathcal{R} = \Pi \times \Omega$, where a light ray is identified by four coordinates $\boldsymbol{L} = (s, t, y, x)$, which describe the intersections with two parallel planes. Since most of the imaging systems can be simplified into two parallel planes, such as the focal plane and the image plane in the traditional optical system, the two-plane parametrization has higher rationality and practicability. Here (s, t) are viewpoint coordinates on the focal plane Π, and (y, x) are coordinates on the image plane Ω.

If we define (s', t') as the lens center of a camera in the focal plane, and see the sensor plane as the image plane Ω, then the focal length is the distance between these two planes. A sub-aperture view $L(s', t')$ captured at the viewpoint coordinate (s', t') can be regarded as a sample of all the light rays going through the lens center (s', t') on the sensor plane. The $L(s', t', x, y)$ is thus the value of the $(x, y)_{th}$ pixel on the sensor. Naturally, if we move the position of the lens center within the focal plane, the optical axis of the camera stays orthogonal to the image plane. This way, not only the light in the xy plane but also in the st plane is collected, which is exactly the light field described by two-plane parametrization.

Another popular representation of the light field is the epipolar plane images (EPIs). They can be obtained by restricting 4D ray space to 2D slices, see figure 2.3, while for an epipolar volume, either the s- or t-coordinate is fixed. The latter is called a horizontal, the former a vertical epipolar volume. If the intersection of the horizontal and vertical epipolar volumes is the center view of the light field, these two particular volumes are called the "cross-hair" of the light field.

In this approach, we use the epipolar volumes instead of all images in the light field. On the one hand we believe that the information from the epipolar volumes are sufficient for the super-resolution task; on the other hand, using only the epipolar volumes reduces the dimensionality of the input to the network.

Figure 2.3: *[14] A light field is defined on a 4D volume parametrized by image coordinates (x, y) and view point coordinates (s, t). Epipolar images (EPIs) are the slices in the sx- or yt-planes depicted to the right and below the center view. As the camera moves, projections of scene points trace straight lines on the EPIs, leading to characteristic patterns.*

2.2. Light fields acquisition methods

The light field super-resolution approach proposed in this work is a deep-learning method, i.e. a data-driven method, thus obtaining a large number of ground truth light fields becomes indispensable. As mentioned earlier, there are several ways to capture light fields shown by Fig. 2.4.

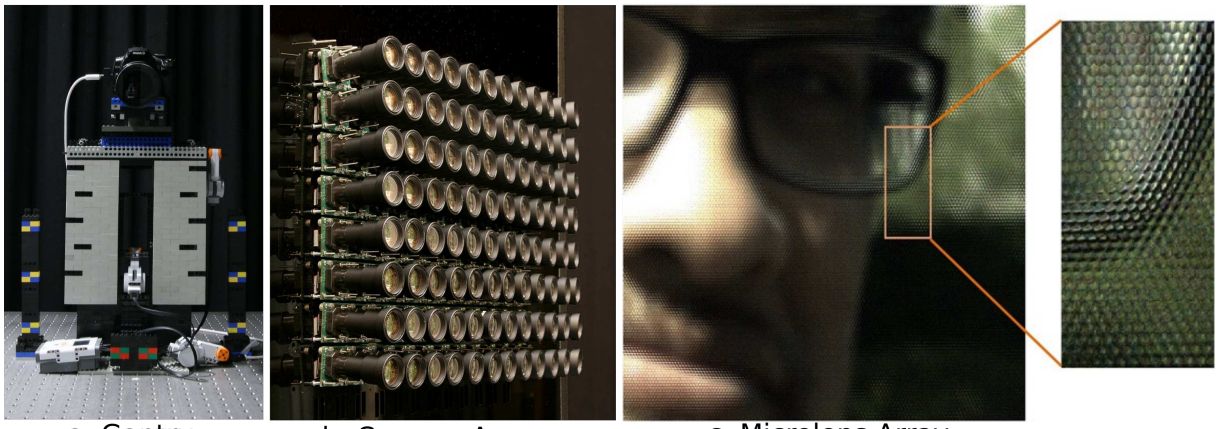


Figure 2.4: *Light fields acquisition methods: **a**. Single camera on the gantry [19]. **b**. The setup of a camera array [19]. **c**. A detailed look of the image captured by a microlens array camera. On the right hand side is the whole image which consists of sub-aperture views that are illustrated on the left.*

A light field of the static scenes can be acquired by moving a single camera over the scene, for example, mounting the camera on a gantry or the robot arm and taking images

of the scene step by step. The quality of the images depends on the specification of the camera. The main drawback of moving a single camera is that it is time-consuming, each exposure takes only one image of the scene.

The second way to capture light field is to use the camera array, i.e. arranging multiple cameras as an array or a matrix with their centers on the same plane. The advantage of the camera array is that, with a single exposure of all the cameras at the same time, one can obtain the light field. However, the angular resolution of the captured light field is limited by the number of cameras used in this setup. Furthermore, it is close to impossible to move the camera array around thus the usage is strongly restricted.

As light field imaging becomes popular, the consumer level plenoptic cameras equipped with microlens array, for example, the Lytro plenoptic cameras, have come onto the stage. The microlens array, as the name suggests, is set up with multiple lenses of small size in front of the sensor in the camera. When capturing an image with the microlens array cameras, the light rays firstly go through the main lens and project onto a specific area of the sensor through the microlens. The acquired image consists thus the sub-aperture views of the scene, i.e. a light field. Although the consumer level microlens array cameras have the advantage that they are easy to carry, the images usually suffer from low spatial resolution, the noise caused by the natural illumination, and sometimes unwanted effects arise due to optical diffraction at the small features.

With the help of computer graphics, it is possible to generate high-resolution light fields through rendering. Compared to the optical equipment based methods mentioned above, rendering provides better ground truth light fields since the information such as resolution, disparity, illumination condition, objects, and background can be customized and stored for later use.

2.3. Light fields applications

2.3.1. Depth estimation

A common usage of the light field is estimating the depth in the captured scene to furthermore reconstruct the 3D structure of it. By fixing the s-coordinate and choosing any two horizontal viewpoint coordinates t_1 and t_2 within the focal plane, we obtain the sub-aperture views $L(s, t_1)$ and $L(s, t_2)$ which form a stereo pair. A real-world point projected on these sub-aperture views will have the same vertical coordinate y on the image plane but a shift in the horizontal direction, so its projections in the sub-aperture image planes are represented by $L(s, t_1, x_1, y)$ and $L(s, t_2, x_2, y)$. Then we can simply compute the depth Z of this point in the real-world coordinate by:

$$b = t_1 - t_2 \tag{2.3}$$

$$d = x_1 - x_2 \tag{2.4}$$

$$Z = \frac{bf}{d} \tag{2.5}$$

where b donates the baseline, i.e. the distance between the camera centers, d serves as the disparity of the corresponding pixels in the sub-aperture views and f is the focal length.

Since the light field has redundant structure, it contains richer information about the scene than a single image, a stereo pair or an image sequence. Therefore the depth estimation using the light field can be more accurate if the corresponding pixels are found in the images.

2.3.2. Super-resolution

Due to the limit of the hardware such as the storage or the built-in processor of the plenoptic cameras, one usually faces the low resolution either in the spatial or the angular domain when capturing the light field. However, the light field itself is a rich source of information which is sufficient for the super-resolution task. A sub-aperture view has multiple neighbouring views about the same scene, which makes it possible to super-resolve a single or several views, even the whole light field.

Having the high-resolution light field L_h we can easily downscale it to a low-resolution version L_l using a degradation model. A single low-resolution view in the light field is spatially downscaled by:

$$L_l(s,t) = D(s,t)B(s,t)L_h(s,t) + N(s,t) \tag{2.6}$$

where the $D(s,t)$ and $B(s,t)$ donate the decimation and blurring matrices , with $N(s,t)$ being the additive noise. For simplicity, we assume that the decimation and blurring process is the same for every view in the light field and call this downscale mapping V, furthermore we ignore the noise term, thus we have:

$$L_l = VL_h \tag{2.7}$$

For the spatial-angular downscaling of a light field, decimation process is not only for the pixel-wise sampling but also view-wise. Without loss of generality, we formulate the degradation model as:

$$L_l = WVL_h \tag{2.8}$$

where W is the decimation matrix or the geometric wrapping matrix for view shrinkage.

In practice, however, the degradation model is often unknown, so the super-resolution, i.e. restoring the high-resolution light field is not simply to apply the inverse matrices but computing the inverse transformation of the degradation process. As a result, the light field super-resolution model becomes:

$$V^{-1}W^{-1}L_l = V^{-1}W^{-1}WVL_h = L_h \tag{2.9}$$

Again, for simplicity we rewrite $V^{-1}W^{-1}$ as A and obtain the model as:

$$AL_l = L_h \tag{2.10}$$

The task turns out to be calculating the inverse mapping A. As [20] points out, this inverse problem is ill-posed as insufficient information given in L_l to reconstruct the L_h

is underdetermined, namely calculating this inverse mapping by

$$A = L_h L_l^{-1} \tag{2.11}$$

does not have a unique solution, thus optimization algorithms must be carried out, where complex mathematical operations, parameter settings and updating strategies need to be taken into account.

In this work, we employ the computational power of GPUs, construct the convolutional neural network C to solve the super-resolution problem:

$$C(L_l, \theta, L_h) = L_h^{'} \tag{2.12}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} E(C(L_l, \theta, L_h), L_h) \tag{2.13}$$

with θ the set of all parameters in the network. $L_h^{'}$ is the output of the trained neural network. The goal of this learning-based approach is, with the help of gradient-descent algorithm, to obtain a set of possibly optimal parameters θ such that the difference E between the output $L_h^{'}$ and the ground truth L_h becomes minimal. E can be any form of a metric that measures the difference between network's output and the ground truth, which is referred to the cost- or loss function in the CNN-based approaches. In the next chapter, we will give a detailed introduction of the convolutional neural network.

Chapter 3

Deep Learning with Convolutional Neural Network

Machine learning helps to solve the problems that are difficult or even close to impossible for deterministic computer programs such as recognizing the animal in an image or understanding a sentence in an audio clip, because they require huge amount of data and an exhaustive mapping between the input and target value. Learning itself is not regarded as the task of machine learning but acquiring the ability to accomplish specific tasks. According to the definition of machine learning by Thomas Mitchell [21], given a task T and the performance metric P, the computer program learns the experience E such that by accumulating E, P increases for the task T. In practice, the definitions of task, experience and performance metric vary, but the core of machine learning remains to get better performance by gaining the experience. Deep learning is a particular branch in machine learning which has achieved great success in the past decade, especially in the computer vision area. The "deep" in the name comes from the structure of the convolutional neural network (CNN) that is widely employed in this area. Usually, the deeper a network is, i.e. the more layers a network has, the better the performance will be.

As this work is dealing with the light field super-resolution problem, we will introduce the deep learning from the perspective of digital image processing and focus on the state-of-the-art techniques employed in our approach in this chapter. Besides, the operations in the convolutional neural network will be explained with the implementations in TensorFlow.

3.1. Supervised and unsupervised learning

Machine learning can be roughly divided into supervised learning and unsupervised learning. The training of supervised learning needs the dataset with each of its sample having a label or a target. By observing the random variable X and its related variable (target) Y in the training data, supervised learning trends to fit the model f_θ to the training samples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. Test data is a set of samples $(x_1^*, y_1^*), (x_2^*, y_2^*), ..., (x_m^*, y_m^*)$ with the same structure and belongs to the same category as the training data but not used for training. The training process can be seen as the learning the mapping to predict the

correct y^* from new instance (x^*, y^*) of the test data:

$$f_\theta(x^*) = y^* \tag{3.1}$$

θ donates the set of all trainable parameters of the model that are learned to minimize the difference between estimated function values and targets, and we call this difference during training the cost or the loss E. A typical cost function for image restoration problems like image super-resolution is the mean squared error (MSE):

$$E_{MSE}(X, Y, \theta) = \frac{1}{N} \sum_{n=1}^{N} \|f_\theta(x_n) - y_n\|_2^2 \tag{3.2}$$

This MSE cost is the so-called L^2 loss. The goal is, therefore, to minimize the loss to train the model:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, E(X, Y, \theta) \tag{3.3}$$

To predict the correct y from new instance of the test data, the model f_θ must have the ability to generalize. However, there are only finite number of samples in the training data and infinite number of test pairs which could be completely different, i.e. one only rarely observes the precise relationship $y = f_\theta(x)$. Instead there exists also measurement error. Under such conditions, we assume that the training and test pairs are independently sampled from the same joint distribution $Pr(x, y)$, and the measurements follow a normal distribution over the correct values:

$$Pr(y|x, \theta) = \textit{Norm}[f_\theta(x), \sigma^2] \tag{3.4}$$

Our goal can be then also formulated as maximizing the probability to see the outputs we sampled from the training data given the inputs from the training data:

$$\begin{aligned}
\hat{\theta} &= \underset{\theta}{\operatorname{argmax}}\, P(y|x, \theta) = \underset{\theta}{\operatorname{argmax}} \prod_{n=1}^{N} P(y_n|x_n, \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \left\{ \prod_{n=1}^{N} Norm_{y_n}[f_\theta(x_n), \sigma^2] \right\} \\
&= \underset{\theta}{\operatorname{argmax}} \left\{ \prod_{n=1}^{N} \frac{1}{2\pi\sigma^2} exp\left(-\frac{(f_\theta(x_n) - y_n)^2}{2\sigma^2} \right) \right\} \\
&= \underset{\theta}{\operatorname{argmin}} \left\{ -log\left[\prod_{n=1}^{N} \frac{1}{2\pi\sigma^2} exp\left(-\frac{(f_\theta(x_n) - y_n)^2}{2\sigma^2} \right) \right] \right\} \\
&= \underset{\theta}{\operatorname{argmin}} \left\{ \prod_{n=1}^{N} \left(-log(\frac{1}{2\pi\sigma^2}) + \frac{(f_\theta(x_n) - y_n)^2}{2\sigma^2} \right) \right\} \\
&= \underset{\theta}{\operatorname{argmin}} \left\{ \frac{1}{2\sigma^2} \prod_{n=1}^{N} (f_\theta(x_n) - y_n)^2 \right\} \sim \underset{\theta}{\operatorname{argmin}}\, L^2(f_\theta(x), y)
\end{aligned} \tag{3.5}$$

This is called the maximum-likelihood estimation, which is proven to be the best asymptotic estimation in terms of convergence rate if the number of training samples goes to infinity. As the deep learning is a data-driven approach, with the increasing number of training samples, maximum-likelihood estimation seems to be the first choice, however, according to equation 3.5, the maximum-likelihood estimation leads to almost the same result as the L^2 loss. So it is reasonable to use the L^2 loss for the image restoration tasks as the cost function.

Unlike the supervised learning, the training data for unsupervised learning usually does not contain the label or target value since unsupervised learning tries to extract useful structure from the training data, for example, principal component analysis and clustering. In a typical image restoration problem, unsupervised learning is often not used, so we are not going to introduce it in detail.

3.2. Feed-forward network and back-propagation

As a classical model in deep learning, feed-forward network defines a mapping $f_\theta(x) = y$ and learns the values of trainable parameters to obtain the best fit to the real mapping f^*. It is known as a network because it consists of many different functions and relates to a directed acyclic graph which describes how these functions are composed. Each function can be seen as a path and its output as the node in the graph. The information of the input flows into these nodes, processed by the operations defined in the functions on the paths and finally becomes the output.

Assume there are four functions g^1, h^1, h^2 and g^2 in the network. The directed graph tells that they are composed as the following:

$$f_\theta(x) = g^2_{w_2} \circ h^2_{v_2} \circ h^1_{v_1} \circ g^1_{w_1}(x) \tag{3.6}$$

with $\theta = (w_1, w_2, v_1, v_2)$. The depth of this network is thus four. g^1 serves as the input layer, while g^2 is the output layer, in which the output of the network is generated. The training sample (x, y) makes it clear that the parameters θ should be learned such that the output is as close as possible to the target value y. By contrast, the layers in between, namely the h^1 and h^2 whose outputs have no target values, are therefore called hidden layers.

At this moment, the network knows what to learn. However, it still has to be wise to how to learn. Simply speaking, the gradient $\nabla_\theta E(x, \theta)$ of the loss function $E(x, \theta)$ with respect to the parameters θ needs to be computed. This way, the network finds the direction in which the loss is going down and updates the parameters to improve the performance. The back-propagation algorithm can do this job. Here we introduce an example to see how the back-propagation algorithm computes the gradient in a feed-forward network by utilizing the chain rule of calculus.

Let $x \in \mathbb{R}^N$, $\theta \in \mathbb{R}^K$, $f : \mathbb{R}^N \times \mathbb{R}^K \to \mathbb{R}^M$ and $g : \mathbb{R}^M \to \mathbb{R}$. $y = f(x, \theta) \in \mathbb{R}^M$. Consider an increment $\epsilon \in \mathbb{R}^K$ of the parameter θ, we will obtain:

$$\begin{aligned} f(x, \theta + \epsilon) &= \begin{pmatrix} f_1(x, \theta + \epsilon) \\ \vdots \\ f_M(x, \theta + \epsilon) \end{pmatrix} \\ &= \begin{pmatrix} f_1(x, \theta) + \nabla f_1(x, \theta)^T \epsilon + r_1(\epsilon) \\ \vdots \\ f_M(x, \theta) + \nabla f_M(x, \theta)^T \epsilon + r_M(\epsilon) \end{pmatrix} \\ &= \begin{pmatrix} f_1(x, \theta) \\ \vdots \\ f_M(x, \theta) \end{pmatrix} + \begin{pmatrix} \nabla f_1(x, \theta)^T \\ \vdots \\ \nabla f_M(x, \theta)^T \end{pmatrix} \epsilon + \begin{pmatrix} r_1(\epsilon) \\ \vdots \\ r_M(\epsilon) \end{pmatrix} \\ &= f(x, \theta) + Df(x, \theta)\epsilon + r(\epsilon) \end{aligned} \tag{3.7}$$

If $\lim_{\epsilon \to 0} \frac{r(\epsilon)}{\|\epsilon\|} = 0$, then all components are differentiable. The derivative of $f(x, \theta)$ at θ is the $M \times K$ matrix:

$$\begin{aligned} D_\theta f(x, \theta) &= \begin{pmatrix} \nabla f_1(x, \theta)^T \\ \vdots \\ \nabla f_M(x, \theta)^T \end{pmatrix} \\ &= \begin{pmatrix} \partial_1 f_1(x, \theta) & \cdots & \partial_K f_1(x, \theta) \\ \vdots & \ddots & \vdots \\ \partial_1 f_M(x, \theta) & \cdots & \partial_K f_M(x, \theta) \end{pmatrix} \in \mathbb{R}^{M \times K} \end{aligned} \tag{3.8}$$

This is called the Jacobian matrix of $f(x, \theta)$ at θ. With $\nabla g(y)^T = \nabla_\theta g(f(x, \theta))^T \in \mathbb{R}^M$, the chain rule for calculating the derivative of the concatenation $h = g \circ f$ with respect to the parameter w is

$$D_\theta h(x, \theta) = D_\theta g(f(x, \theta)) D_\theta f(x, \theta) \tag{3.9}$$

Back-propagation can be applied not only to vectors but also to the tensor of any dimensions. With the help of the chain rule, the gradient of any parameter can be written out in a feed-forward network, where the target is a real-valued node later in the graph which depends on this parameter. To do this, a forward pass from the parameter to the node, then a backwards pass to propagate gradients back through the network are computed. This process can be seen as a directed graph for the gradient computation in reverse order to the original graph, where each function generates computational nodes to apply its gradients to a direction vector, which flows back from the next layer.

When back-propagating through the network, we recursively apply the chain rule, the derivative of each layer is multiplied with the previous gradient. As the network grows deeper, multiplication leads to unstable behaviour of the gradients: they can either "vanish" completely or "explode". Both cases will cause the failure of the training. One way to avoid gradient exploding is to set a threshold for the gradient's magnitude. If the magnitude is too large, then simply cut it such that the magnitude is smaller

than the threshold. This method is called gradient clipping. Another choice is weight regularization. The loss function is modified as:

$$E^*(x, \theta) = E(x, \theta) + \alpha\|\theta\|^2 \tag{3.10}$$

α refers to regularizer coefficient. Therefore, if a gradient explosion occurs, the norm of the weight will become very large, and the regularization term $\alpha\|\theta\|^2$ can partially limit the occurrence of the gradient explosion. Some other options to deal with unstable gradient will be explained as they are introduced in the next parts.

3.3. Gradient Descent and Adaptive Moment Estimation (ADAM)

Closely connected to the back-propagation is the gradient-descent algorithm. After the back-propagation computes the gradients in a network, one needs the gradient descent algorithm to update the parameters thus to minimize the cost. Recall that we donate $\nabla_\theta E(x, \theta)$ that is computed via back-propagation as the gradient of the loss function $E(x, \theta)$ with respect to the parameters θ. Let $\tau \in \mathbb{R}$ and θ_0 serve as step size and the initial guess of the parameter, respectively, then the gradient descent proceeds as follows:

$$\theta_{k+1} = \theta_k - \tau\nabla_{\theta_k} E(x, \theta_k) \tag{3.11}$$

where k represents the number of iteration. The gradient $\nabla_{\theta_k} E(x, \theta_k)$ points out the steepest direction in which the value of the cost function goes at the current step k. By negating the gradient, the cost decreases oppositely to that direction. The step size τ, known as the learning rate in the area of machine learning, controls how far it goes to reach the next point θ_{k+1} in each iteration. As the gradient descent algorithm proceeds with the iteration, we get closer to a minimum of the cost function thus obtain a candidate of the best parameter $\hat{\theta}$.

Here we gave the basic principle of the gradient descent algorithm. In practice, however, it is not guaranteed to find the global minimum by using it. The reasons for this are manifold. First of all, the cost function in deep learning, especially for image restoration problems is high-dimensional and often non-convex, i.e. there exists multiple local minima. The gradient algorithm itself cannot recognize which one is the global minimum. Second, the learning rate τ needs to be chosen carefully, otherwise the loss function will not converge. Furthermore, the initial value of the parameter θ_0 decides where the searching starts for the optimal $\hat{\theta}$. Combined with an inappropriate learning rate, it can lead the searching to an awkward situation.

Consider a simple example illustrated in Fig. 3.1. Assume we have a non-convex cost function $E(\theta)$ in which there is a local minimum, and a global minimum. The gradient descent algorithm is applied. With a small learning rate, the descent process lasts long. In addition, if the initial value of the parameters sets the start point near the local minimum, the searching gets stuck in the local minimum, therefore, loses the opportunity to find the global one. By contrast, if the learning rate is too large, with a start point near the

global minimum, the descent step would have the chance to skip the goal and jump to the local one or some points even more inappropriate. Relatively too large step size could also encounter the situation that the searching gets stuck since the descent step goes back and forth on two sides of a minimum but never reaches it.

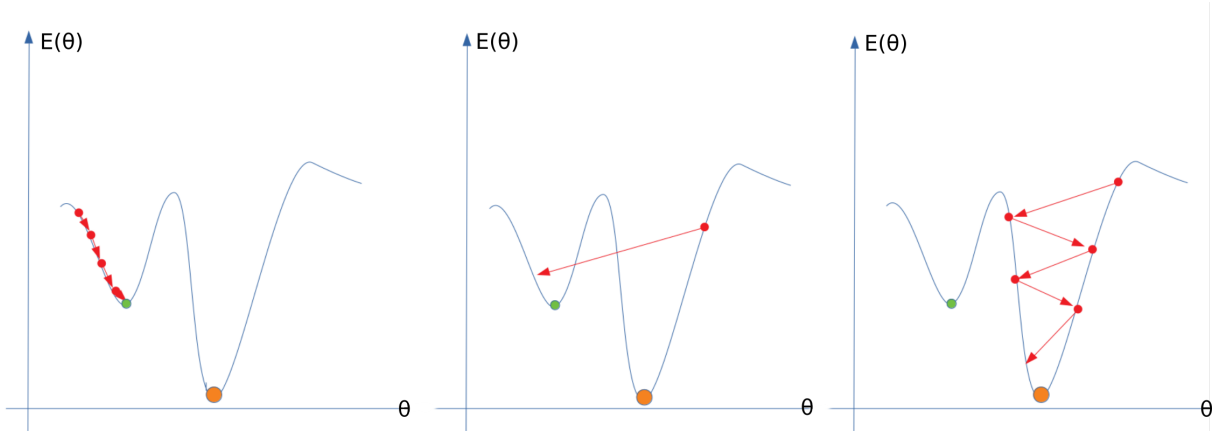


Figure 3.1: *The searching for the global minimum using gradient descent algorithm with inappropriate learning rates τ. Left: Too small step size leads to a very slow searching and getting stuck in the local minimum. Middle: Too large step size causes drastic updates such that the global minimum is skipped. Right: Relatively too large step size let the searching travel back and forth around the minimum or even a divergent behaviour.*

As mentioned earlier in this chapter, the maximum-likelihood estimation, which is equivalent to the L^2 loss, tries to maximize the probability of seeing the target value given the training samples. If we feed the network with the entire training dataset, then it is highly possible that we obtain the real probability distribution of the data set thus have the best performance of the network. In fact, this is close to impossible to fit in all the data for the training due to the limit of the hardware. To proceed the gradient descent, we randomly draw a subset, namely a batch from the training data, and update the parameter according to the gradient within the batch. This technique is the so-called stochastic gradient descent (SDG). However, the gradient from the batch can be noisy since the probability distribution of the batch cannot represent that of the entire dataset. Furthermore, recall that the high-dimensional non-convex cost function has multiple local minima. If the Hessian matrix[1] of the cost function is "badly conditioned", i.e. the ratio of largest to smallest absolute eigenvalue is very large, we can imagine in this case a long, steep valley in the graph of the function, where the SGD struggles as the searching has the "zig-zag" route thus moves very slowly towards the minimum.

Fortunately, there are several extensions of SDG to solve the problems mentioned above basically by accumulating gradients over time, averaging over the recent history and then slowly reducing the contributions of previous gradients again. The Adaptive

[1] Hessian matrix is a square matrix of second-order partial derivatives of a multivariate function, describing the local curvature of the function. It is used to identify the local minimum of the function by observing its eigenvalues. For more detail about the Hessian matrix we refer to chapter 4 of *Deep Learning* by Ian Goodfellow [22].

Moment Estimation (ADAM) [23] is the most popular one among all these extensions. Specifically, the algorithm calculates the exponential moving averages of the gradient and the squared gradient, as well as the hyper-parameters β_1 and β_2 that control the decay rate of these moving averages:

$$
\begin{array}{lll}
1. & g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}) & \text{Compute the gradient at current iteration} \\
2. & m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t & \text{Update biased first moment} \\
3. & v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 & \text{Update biased second moment} \\
4. & \hat{m}_t \leftarrow m_t / (1-\beta_1^t) & \text{Compute bias-corrected first moment} \\
5. & \hat{v}_t \leftarrow v_t / (1-\beta_2^t) & \text{Compute bias-corrected second moment} \\
6. & \theta_t \leftarrow \theta_{t-1} - \tau \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) & \text{Update parameters}
\end{array}
\tag{3.12}
$$

where g_t^2 is the element-wise square $g_t \odot g_t$, τ the learning rate and ϵ a constant(10^{-8}) to prevent dividing by zero. β_1^t and β_2^t donate the β_1 and β_2 to the power of t, respectively. The initial values of the moving averages for moment estimation and β_1, β_2 are close to 1 (recommended as $\beta_1 = 0.9$ and $\beta_2 = 0.999$). The step 4 and 5 of the bias correction are taken into account if they are initialized with zero vectors. The ADAM not only calculates the adaptive learning rate based on the first-order moment m, but also makes full use of the second-order moment v of the gradient. The learning rate τ is rescaled by dividing the term $\sqrt{\hat{v}_t} + \epsilon$ since v is proportional to the magnitude of the gradient, the parameters receiving large updates get smaller step size, while the those having small updates will have larger steps. As the iteration goes on, the term $1 - \beta^t$ grows thus gradually reduces the contribution of the previous gradients.

3.4. Building blocks of a CNN

In deep learning, a convolutional neural network (CNN) is a class of feed-forward networks, most commonly applied to computer vision tasks. CNN is inspired by biological processes because the pattern of connections between neurons resembles the organization of the animal's visual cortex. Individual neurons respond to stimuli only in restricted areas of the receptive field. As a feed-forward network, the CNN composed of multiple layers in which the main operations are convolutions. Besides, there are other operations working together with convolution for different purposes. In this section, we will give an introduction of them.

3.4.1. Convolution

From the perspective of image processing and CNN, the convolution is usually done in the spatial domain of the image. Let K donate a $N \times M$ kernel and I a grayscale image with resolution $H \times W$, we can imagine the convolution as sliding the kernel K onto each pixel of the image I, calculating the element-wise multiplication of its $N \times M$ neighbouring pixels and the corresponding element of the kernel, and finally adding them up as the value of that pixel. In mathematical term, the 2D convolution can be expressed

as follows:

$$(I * K)_{(h,w)} = \sum_{n=1}^{N} \sum_{m=1}^{M} I_{(h-n,w-m)} K_{(n,m)} \tag{3.13}$$

The output of the convolution is called features. The values in the kernel are called weights and they are constant during the operation and only updated after an iteration of gradient descent. This is called weight sharing which enables learning only one set of parameters in a layer instead of a set of parameters for each position in an image.

A fundamental property of convolution is that it is a linear operation. Any 2D convolution can be written out as matrix multiplication. If the 3×3 convolution kernel K and the 4×4 image I are row-first expanded into column vectors, the convolution operation defined above them can be written as follows:

$$I * K = \begin{pmatrix} K_{(0,0)} & 0 & 0 & 0 \\ K_{(0,1)} & K_{(0,0)} & 0 & 0 \\ K_{(0,2)} & K_{(0,1)} & 0 & 0 \\ 0 & K_{(0,2)} & 0 & 0 \\ K_{(1,0)} & 0 & K_{(0,0)} & 0 \\ K_{(1,1)} & K_{(1,0)} & K_{(0,1)} & K_{(0,0)} \\ K_{(1,2)} & K_{(1,1)} & K_{(0,2)} & K_{(0,1)} \\ 0 & K_{(1,2)} & 0 & K_{(0,2)} \\ K_{(2,0)} & 0 & K_{(1,0)} & 0 \\ K_{(2,1)} & K_{(2,0)} & K_{(1,1)} & K_{(1,0)} \\ K_{(2,2)} & K_{(2,1)} & K_{(1,2)} & K_{(1,1)} \\ 0 & K_{(2,2)} & 0 & K_{(1,2)} \\ 0 & 0 & K_{(2,0)} & 0 \\ 0 & 0 & K_{(2,1)} & K_{(2,0)} \\ 0 & 0 & K_{(2,2)} & K_{(2,1)} \\ 0 & 0 & 0 & K_{(2,2)} \end{pmatrix}^T \begin{pmatrix} I_{(0,0)} \\ I_{(0,1)} \\ I_{(0,2)} \\ I_{(0,3)} \\ I_{(1,0)} \\ I_{(1,1)} \\ I_{(1,2)} \\ I_{(1,3)} \\ I_{(2,0)} \\ I_{(2,1)} \\ I_{(2,2)} \\ I_{(2,3)} \\ I_{(3,0)} \\ I_{(3,1)} \\ I_{(3,2)} \\ I_{(3,3)} \end{pmatrix}$$

$$\implies \begin{pmatrix} \sum_{i=0}^{2} \sum_{j=0}^{2} K_{(i,j)} I_{(i,j)} & \sum_{i=0}^{2} \sum_{j=0}^{2} K_{(i,j)} I_{(i,j+1)} \\ \sum_{i=0}^{2} \sum_{j=0}^{2} K_{(i,j)} I_{(i+1,j)} & \sum_{i=0}^{2} \sum_{j=0}^{2} K_{(i,j)} I_{(i+1,j+1)} \end{pmatrix} \tag{3.14}$$

Note that we obtain the final image by reshaping the result of the matrix multiplication which is a 4×1 vector.

In TensorFlow, the 2D convolution *conv2D* accepts a 3D image $I(H, W, P)$ and a 4D kernel $K(N, M, P, Q)$ as inputs, where P and Q define the number of channels in the input and the desired output number of channels in this layer, respectively. The grayscale images have only one channel, but in practice, color images are used more often, while the features usually have multiple channels as well which are going to be fed to the consecutive convolutional layers. The kernel $K(N, M, P, Q)$ actually consists

of Q sub-kernels of size $N \times M \times P$. Each channel of the sub-kernel K_q convolves with the corresponding channel of I as described in equation 3.13 and element-wise sums up to form a single feature map F_q. Finally, all the feature maps are concatenated thus we obtain the output feature volume F with Q channels for this layer. By growing the number of feature maps, the convolution kernel extracts different kinds of structures in the image. As these features going deeper in the network, they become more and more specific thus the trained network preserves the most important structures from the data.

One may notice that if the kernel slides onto the very first pixel of the image, there are no corresponding pixels included in convolution for the first row and column of the kernel. This situation also occurs for all the edge pixels in the image. For this, the *conv2D* needs another input *"Padding"* where we can define it as "VALID" or "SAME". Having the "VALID"-padding, sliding the kernel K of spatial size (N, M) will start from the $(\lceil \frac{N}{2} \rceil, \lceil \frac{M}{2} \rceil)_{th}$ pixel of I, i.e. the "VALID"-padding leaves all the pixels in the image for convolution but finally only $(H - \lceil \frac{N}{2} \rceil) \times (W - \lceil \frac{M}{2} \rceil)$ positions will have output values. The features shrink their size on top and bottom edges by $\lceil \frac{N}{2} \rceil$ pixels and on left and right edges by $\lceil \frac{M}{2} \rceil$, respectively. In case we want to keep the spatial resolution of the output features the same as the input, "SAME"-padding is the option. It adds $\lfloor \frac{N}{2} \rfloor$ rows to the top and bottom edges and $\lfloor \frac{M}{2} \rfloor$ columns on the left and right sides of the image, filling them up with zeros. This way, the convolution starts from the real first pixel of the image and every pixel receives the output value.

Another frequently used input in *conv2D* is *"Stride"*. It controls the step size sliding the kernel in the image. Commonly it is set to 1 since we want the convolution to be applied to every pixel. However, if one wants to downscale the output's size, either for sparing the computational resource or on a specific purpose, the stride can be set other than 1. For example, if the output should have half of the size as the input, the stride is set to 2 as the convolutional kernel skip every second pixel in rows and columns.

The 3D convolution is an extension of the 2D one. The kernel can be imagined as a cube and it moves around in the input volume. The pixel value convolved with the kernel is not only computed with its spatial neighbours but also the ones within the temporal dimension of the kernel. Correspondingly, the *conv3D* in TensorFlow requires one more dimension in the input and kernel as $I(D, H, W, P)$ and $K(D, N, M, P, Q)$, where D represents the temporal dimension of the data and kernel.

3.4.2. Pooling

Instead of strided convolution, pooling also downscales the dimensionality of the features. Essentially, it is a technique that maximizes the retention of spatial information and feature information while streamlining the amount of feature map. The purpose is to compress and concentrate through the feature map, thus the input to the follow-up hidden layer becomes small, and the calculation efficiency can be improved.

Likewise, pooling can also be seen as moving a window over the features. As the pooling window of size $N \times M$ slides to a pixel $I_{(h,w)}$, it extracts value from all pixels currently within this window, which can be the maximum, minimum or the mean. Then the polling window jumps to the next pixel $I_{(h+N,w)}$ in the same row, or $I_{(h,w+M)}$ in the next column.

After pooling, the extracted values form a matrix whose spatial resolution depends on the size of the pooling window. This matrix, as the output features of the pooling layer, is fed to the next layer in the network. Compared to strided convolution, pooling shrinks the size of the features as well. Without training any parameters, it focuses only on compressing the information through feature maps.

3.4.3. Activation functions

Except convolutional layers, the activation functions (Fig. 3.2) are regarded as the most important component in the network. On the one hand, they filter out the non-critical information, only let the key information in the features flow into the next layers. This simulates the working principle of the neurons in a biological network, which is why they are called "activation functions". On the other hand, they provide non-linearity that significantly improves the representative ability of the network, since convolution is a linear operation.

There are several popular activation functions available for deep learning tasks.

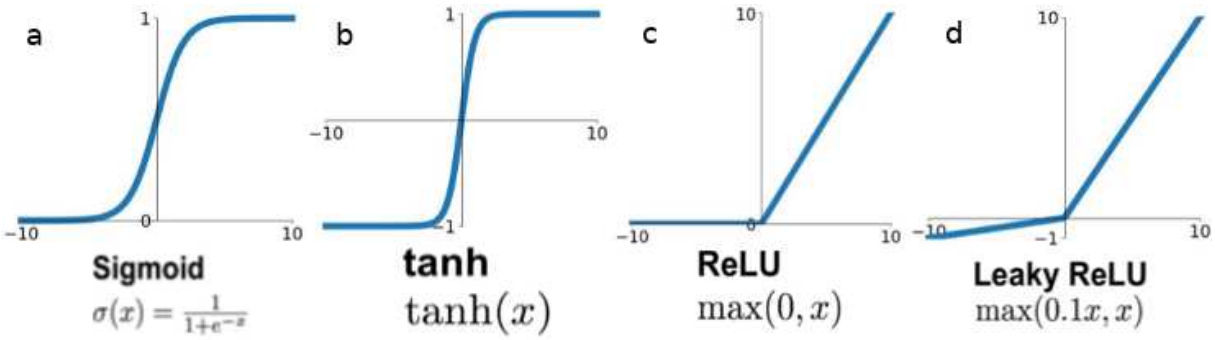


Figure 3.2: *Commonly used activation functions in CNN [24].* ***a.*** *Sigmoid;* ***b.*** *Hyperbolic Tangent (tanh);* ***c.*** *Rectified Linear Unit (ReLU);* ***d.*** *Leaky Rectified Linear Unit (Leaky ReLU)*

Sigmoid squeezes the input into the 0 to 1 interval that is consistent with the range of probabilities, which is why sigmoid is very popular in classification tasks. The shape of tanh is similar to sigmoid, except that tanh "compresses" the inputs into the interval $[-1, 1]$. These two activation functions have a common problem that the activated values are very easy to get saturated due to their value range, because once the input $|x| \geq 10$, they remain constant after the activation, thus the gradients will be close to zero. As a result, no change will happen during gradient descent, i.e. there is no learning for the weight in the network, which is called the gradient vanishing problem.

ReLU (Fig. 3.2c) is the most commonly used activation function because of its simplicity during back-propagation, and it is computationally not expensive. ReLU handles the gradient vanishing problem that frequently occurs in sigmoid and tanh, and it converges faster than some other activation functions. By contrast, ReLU's value range $[0, \infty)$ leads to the so-called "dead neuron" problem, that is, the value of the input that continues to be negative is always zero, thus the weights related to them are never updated.

Leaky ReLU is an extension of ReLU. It has all properties of ReLU, plus it will never

have "dead neuron" problem. We can consider different multiplication factor to form different variations of Leaky ReLU. Parametric leaky ReLU is a further extension of Leaky ReLU in which the multiplication factor is treated as a trainable parameter instead of being pre-defined in usage.

Except the activation functions introduced above, there are some other choices like Max-out, Exponential Linear Unit (ELU), to name but a few. Most of these activation functions overcome the weaknesses of the previous ones but usually require more computational effort. Note that for a deep learning task, one has to take input, output and data changes into account, so the activation functions need to be chosen for different tasks very carefully.

3.4.4. Batch normalization

In deep learning, people often split the training data into batches to proceed the training with stochastic gradient descent (SGD). However, the distribution of the values in a batch might be completely different from that of the entire dataset. As we just mentioned in the previous subsection, the activation function should be chosen carefully, i.e. the activation functions are sensitive to their input. If we do not change the value distribution in the batch and occasionally have an inappropriate initial weight, the unstable gradient or "dead neuron" problems will easily occur.

To deal with this situation, batch-normalization is applied before the activation function in each layer such that the activation function is fed with a batch of proper distribution and one does not have to worry about how to initialize the parameters.

Consider a batch $\mathcal{B} = (x_1, ..., x_n)$ with N sample points.

$$\mu_{\mathcal{B}} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3.15}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_{\mathcal{B}})^2 \tag{3.16}$$

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{3.17}$$

$$y_i = \gamma \hat{x_i} + \beta = BN_{\gamma,\beta}(x_i) \tag{3.18}$$

The batch normalization computes the mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ over values of $x \in \mathcal{B}$ then learns additionally the scale and offset parameters γ and β. This way, each layer is trained on normalized input which makes the training robust to "internal shift". Note that the batch normalization should be disabled during testing since true mean μ_{data} and variance σ_{data}^2 are wanted.

3.5. Deep autoencoder

In the narrow sense, the autoencoder is a type of neural network that reconstructs the input as the output after training. It belongs to the self-supervising algorithm which is an instance of supervised learning whose targets are generated from input data. Requiring the model to reconstruct input at the pixel level accurately is not the interest of machine learning but learning advanced abstract features. In fact, when the main task is for example classification, the best features for such task are basically the worst for reconstructing input.

Autoencoder tries to learn a function $f_\theta(x) \approx x$. In other words, it tries to approximate an identity function, making the output $\hat{x}$ close to the input x. If an autoencoder simply learns to set everything to $g(f(x)) = x$, then this autoencoder is of no special use. Instead, the autoencoder should not be designed to replicate input exactly. This usually requires some constraints to be imposed on the autoencoder so that it can only copy the input approximately. These constraints force the model to consider which parts of the input data need to be copied first, so it can often learn the useful characteristics of the data.

As deep learning develops, people have found more potential of autoencoder. It is not limited to reconstruct the input, but applied to other tasks like image transformation, image restoration, machine translation and so on. The networks of an autoencoder architecture, dealing with these tasks are called encoder-decoder networks. In this work, we still call our approach deep autoencoder for light field super-resolution since our model generates a high-resolution version of the input.

3.5.1. Architecture

An autoencoder consists of three parts: encoder, latent space and decoder (Fig. 3.3). We can imagine the whole architecture as a horizontally placed hourglass.

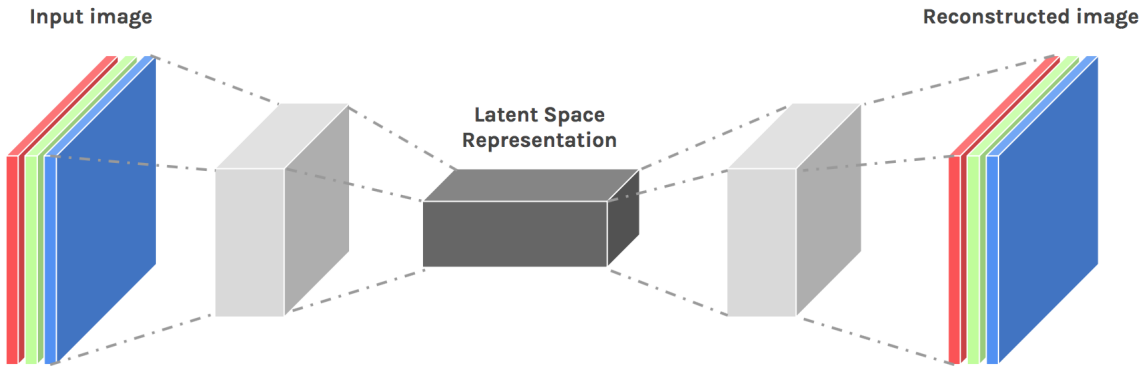


Figure 3.3: *Typical architechture of an autoencoder for input reconstruction [25].*

In order to compress the information until the features reach the bottle-neck of the hourglass, namely the latent space, one usually employs strided convolution or pooling layers to gradually downscale the features in the encoder. The latent space is followed by the decoder, which typically has a symmetrical architecture as the encoder, especially

when an encoder layer performs downscaling, the corresponding decoder layers increase the size of the features as a reverse operation. By consecutively doing these reverse operations in the decoding layers, the features are upscaled such that when the network outputs the result, it has the same shape as the input.

An outstanding advantage of autoencoder in computer vision area is that its architecture compresses the input during the encoding phase and then decompresses the features in the decoder. As we know, the digital images nowadays are of relatively high resolution. Keeping the resolution of the features as same as the input images strongly limits the capacity of the deep network. Using the autoencoder architecture enables constructing deeper network or using a larger batch for the computer vision task, which contributes to a better performance of the model.

3.5.2. Upscaling in the decoder

Previously we introduced pooling and strided convolution as downscaling techniques in the CNN. Employing an autoencoder architecture, one needs to consider how to upscale the features in the decoder.

Unpooling

Unpooling is the reverse operation to pooling. Consider a max-pooling with a 2×2 window over the 4×4 image, three unpooling techniques are shown here:

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 2 \\ 1 & \mathbf{2} & \mathbf{3} & 2 \\ \hline \mathbf{4} & 3 & 2 & 3 \\ 1 & 1 & 2 & \mathbf{5} \end{array}\right] \text{ max pooling} \rightarrow \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{cases} \text{"Bed of nails"} \rightarrow & \left[\begin{array}{cc|cc} 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \\ \text{"Nearest Neighbour"} \rightarrow & \left[\begin{array}{cc|cc} 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \\ \hline 4 & 4 & 5 & 5 \\ 4 & 4 & 5 & 5 \end{array}\right] \\ \text{"Max-unpooling"} \rightarrow & \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & \mathbf{2} & \mathbf{3} & 0 \\ \hline \mathbf{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{5} \end{array}\right] \end{cases}$$

The simplest way is called "bed of nails" in which the max values are kept in the top left corner in the unpooling window at each position and the missing values are padded with zeros. In spite of the fact that this is easy to apply, the result after unpooling has very strong artifacts due to padding zeros. The second unpooling skill "nearest neighbour" fills

the missing values with the value within the unpooling window. The result of it suffers from over-smooth effect. The last technique "max-unpooling" requires remembering the positions of each max value in the original image, but still, fills in the blanks with zeros or some other values uniformly. It shares the same disadvantage with "bed of nails" and additionally needs more memory for the operation.

The pooling and unpooling were once popular in the early years due to the limit of the hardware for deep learning, since both of them are parameter-free. As the hardware develops rapidly this decade, pooling gradually becomes less common in deep learning applications. Moreover, consider the unsatisfying quality of unpooling's results, down- and upscaling are done more often by convolution and deconvolution with strides, respectively.

Interpolation

Speaking of upscaling, one might come up with interpolation as the first choice for the task. There are several commonly used interpolation algorithms: nearest neighbour, bilinear and bicubic interpolation.

The basic principle of the nearest neighbor interpolation algorithm is to make the pixel value of the target position equal to the pixel value closest to it in the neighbourhood. For this, one need to find out where the pixel values comes from in the original image by:

$$\begin{aligned} x_{\text{src}} &= \text{rounding}(x_{\text{dst}} \cdot (W_{\text{src}}/W_{\text{dst}})) \\ y_{\text{src}} &= \text{rounding}(y_{\text{dst}} \cdot (H_{\text{src}}/H_{\text{dst}})) \end{aligned} \tag{3.19}$$

The nearest neighbour algorithm is a basic and simple image scaling algorithm with a small amount of computation, but may cause discontinuities in the pixel values generated by interpolation where sharply jagged artifacts appear.

The bilinear interpolation algorithm is a better image scaling algorithm, which fully utilizes the four real pixel values around a virtual point in the source image to jointly determine a pixel value in the target image, so the result is better than nearest neighbour interpolation while its calculation amount is larger. For a pixel $I_{(i,j)}$ in the target image, let α serve as the scale factor, then a virtual point in the source image I^* is obtained by $I^*_{(i+u,j+v)} = I_{(i/\alpha,j/\alpha)}$, where u and v are float numbers. The value of this pixel is determined by the pixels in the four surrounding positions (i, j), $(i + 1, j)$, $(i, j + 1)$ and $(i + 1, j + 1)$, namely:

$$I^*_{(i+u,j+v)} = (1-u)(1-v)I^*_{(i,j)} + (1-u)vI^*_{(i,j+1)} + u(1-v)I^*_{(i+1,j)} + uvI^*_{(i+1,j+1)} \tag{3.20}$$

It has low-pass filtering properties that damage high-frequency components and may blur the image.

The last interpolation method to be introduced here is the bicubic interpolation. It has the best result compared to nearest neighbour and bilinear interpolation, however demands the largest computational effort as well. Same as the bilinear interpolation, a virtual position in the source image is computed. The bicubic interpolation tries to find a set of coefficients that represents the impact factor of the 16-pixel neighbourhood $\mathcal{N}$ on that virtual position, so that the pixel value of the corresponding point in the target image

is obtained according to the impact factor. Having the bicubic function

$$W(d) = \begin{cases} (\alpha+2)|d|^3 - (\alpha+3|d|^2+1) & \text{for} \quad |d| \leq 1 \\ \alpha|d|^3 - 5\alpha|d|^2 + 8\alpha|d| - 4\alpha & \text{for} \quad 1 < |x| < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

where d stands for the distance between the virtual position and a neighbour either in horizontal or vertical direction, the target value is calculated via

$$I_{(i,j)} = \sum_{n \in \mathcal{N}} I_n^* W_n(d_x) W_n(d_y) \tag{3.22}$$

Although interpolation is parameter-free, because of its high computational requirements and the large batch size used during the training, it is not frequently used in the deep network. Next, we will present convolution transpose which is a preferred upscaling technique in CNN.

Convolution transpose

As convolution is introduced earlier, equation 3.14 shows that it is valid to write the operation as matrix multiplication. To better understand convolution transpose, we give another simple example in 1D: let a 1×4 input convolve with a 1×3 kernel with stride 1 and "SAME"-padding (0-padding)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^T * \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix}^T = \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} = \begin{bmatrix} 0x + ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy + 0z \end{bmatrix} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \mathcal{K}I \tag{3.23}$$

If we transpose $\mathcal{K}$ and multiply it with output, we obtain

$$\mathcal{K}^T \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} = \begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} = \begin{bmatrix} px \\ py + qx \\ pz + qy + rx \\ qz + ry + sx \\ rz + sy \\ sz \end{bmatrix} = \begin{bmatrix} z \\ y \\ x \end{bmatrix}^T * \begin{bmatrix} 0 \\ p \\ q \\ r \\ s \\ 0 \end{bmatrix}^T \tag{3.24}$$

Here the multiplication of the transposed kernel matrix and the output of the original convolution can be written as another convolution, where the new kernel $[z, y, x]$ is the flipped $[x, y, z]$, which is why it is called convolution transpose.

For a 2D case, recall that in equation 3.14 we transform a 3×3 convolutional kernel into a 4×16 matrix for a 4×4 input and finally obtain a 2×2 output after reshaping. Now if we expand the output to a 4×1 vector and right-multiply with the transposed kernel

matrix, we will have a 16×1 vector result. By reshaping it, we upscale the result of the original convolution. Even though it is called the transposed convolution, it does not mean that we take some existing convolution matrix and use the transposed version, i.e. the actual weight values in the transposed matrix does not have to come from the original kernel matrix. Important is that the weight layout is transposed from that of the kernel matrix.

In practice, there is a function *conv2D-transpose* in Tensorflow upscaling the features in the spatial domain by defining its stride larger than 1. Roughly, deconvolution layers allow the model to use every point in the small image to "paint" a square in the larger one. The kernel of convolution transpose traverses the input, weighing the kernel by the value in each position in the input and placing the weighted kernel in the corresponding locations in its output. Overlapping areas are summed up.

Chapter 4

Related Work

4.1. Super-resolution algorithms

Super-resolution can overcome the limit on the resolution of the imaging system. The algorithms for it aim to recover missing details in a high-resolution (HR) image that are not explicitly available in any individual low-resolution (LR) image. The algorithms can be roughly divided into two families, namely the classical multi-image super-resolution, and example-based super-resolution. In the classical algorithms, it is assumed that the high-resolution information is implicitly distributed in multiple low resolution images. Multiple low-resolution images about the same scene are interpolated onto a high-resolution grid after a registration or motion compensation, followed by the deblurring and noise removal. In the example-based ones, people assume that this missing high-resolution information is available in a high-resolution patch database and learn from the low-/high-resolution pairwise examples.

4.1.1. Non-uniform interpolation

Non-uniform interpolation is the most intuitive algorithm. Given the motion information, the HR image on non-uniformly spaced sampling points is obtained. The reconstruction procedure [26–29], either directly or iteratively, is applied to produce the sampling points, followed by the deblurring and noise removal operations. Ur and Gross [30] use the generalized multi-channel sampling theorem [28, 29] to perform non-uniform interpolation on a set of spatially displaced LR images. By applying a block-matching technique to measure relative shifts in multiple images and the Landweber algorithm [31], Komatsu et al. [32] acquire the resolution-improved images. Hardie et al. [33] utilize a gradient-based registration algorithm to estimate the displacement between the acquired frames and introduce a weighted nearest neighbour interpolation approach, in which deblurring and noise removal are achieved by the Wiener filtering. The advantage of the non-uniform interpolation approach is that it takes relatively low computational effort. However, it is limited because it focuses more on registration and sampling methods and ignores the error caused by interpolation, thus suffers from the quality loss of the results.

4.1.2. Frequency domain approaches

Due to the shifting property of the Fourier transform, the frequency domain approach is able to formulate the super-resolution problem as solving the equation relating the aliased discrete Fourier transform (DFT) coefficients of the observed LR images to a sample of the continuous Fourier transform (CFT) of an unknown image:

$$D_{(h,w)} = \Phi C_{(h,w)} \tag{4.1}$$

where Φ is the matrix describing the relation between DFT ($D_{(h,w)}$) and CFT ($C_{(h,w)}$) at the position (h, w). Kim et al. [34] propose a weighted least squares formulation, assuming all LR images have the same blur and noise model. This is later improved by [35] considering different blur and noises in each LR image. Bose et al. [36] introduce the recursive total least squares method for super-resolution to reduce effects of errors in Φ. A discrete cosine transform (DCT)-based method [37] is proposed, decreasing memory requirements and computational costs by using DFT. The frequency domain approaches possess a theoretical simplicity but are also restricted by it because they consider a global translational motion and blur in the LR images.

4.1.3. Regularization-based approaches

As we mention in section 2.3.2, super-resolution is an ill-posed problem. For this, regularization is introduced. The regularization adds some constraints to the loss function of the original problem to narrow down the solution space thus reduce the chance of finding wrong solutions, i.e. makes the model trend to one of the solutions instead of multiple candidates. The regularization is typically formulated as:

$$\hat{X} = \underset{X}{\operatorname{argmin}} \left\{ \|AX - Y\|_p^p + R(X) \right\} \tag{4.2}$$

where X and Y are the true HR and LR images, respectively. A donates the combination matrix of blurring, downscaling and noise. $\| \cdot \|_p^p$ represents the l^p norm. $R(X)$ is called regularization term, or simply regularizer, which has various forms. For example, Tikhonov cost function [38, 39] defines the regularizer

$$R(X) = \alpha \|\Gamma(X)\|_2^2 \tag{4.3}$$

α is the regularization parameter that controls the trade-off between the data term $\|AX - Y\|_p^p$ and the regularizer. Γ is usually a high-pass operator, such as derivatives, Laplacian operators, or even identity matrices. The intuition behind this is to limit the total energy of the image or to force spatial smoothing, because most images are naturally smooth with limited high-frequency activity, and therefore it is appropriate to minimize the amount of high-pass energy in the restored image.

Total variational (TV) [40] methods are regarded as one of the most efficient regularizations for image restoration problems. It penalizes the total amount of change in the image that is measured by the magnitude of the gradient, and tends to preserve edges in

the reconstruction [40, 41]:

$$R(X) = \alpha \|\nabla X\|_p^p \tag{4.4}$$

Farsiu et al. [42] employ the bilateral total variation regularizer (BTV) and arrive at a model which is robust to errors in motion and blur estimation. Wang et al. [43] introduce an end-to-end fast upscaling technique combined with BTV regularization for classical multi-image super-resolution. Li et al. [44] use the locally adaptive bilateral total variation (LABTV) regularization with l^2-norm to balance noise suppression against detail preservation, while Zhang et al. [45] employ the l^1-norm with the BTV and Gaussian Pyramid Optical Flow (GPOF) registration.

4.1.4. Probabilistic approaches

The regularization term is essentially a prior information. The entire optimization problem is a Bayesian maximum a-posteriori estimation (MAP), where the regularization term corresponds to the prior information in the posterior estimation. Given the a-posteriori probability density function (PDF) of the HR image, the super-resolution problem can be expressed in another form:

$$\hat{X} = \underset{X}{\operatorname{argmax}}\, P(X|Y) \tag{4.5}$$

where X is the HR image and Y the low-resolution version of X. By applying Bayesian theorem to the conditional probability, it becomes

$$\hat{X} = \underset{X}{\operatorname{argmax}}\, P(Y|X)P(X) \tag{4.6}$$

The loss function corresponds to the likelihood function $\operatorname{argmax}_X P(Y|X)$ in posterior estimation and the product of it and the prior corresponds to the from of Bayesian MAP. Taking the logarithmic function, it is reformulated as

$$\hat{X} = \underset{X}{\operatorname{argmax}}\, \{\ln P(Y|X) + \ln P(X)\} \tag{4.7}$$

The term $P(X)$ is the prior knowledge of HR images. It forcibly allows the model to have desired characteristics, such as sparseness, low-rank or smoothing. This forces the final solution to tend to the prior knowledge. If we model the HR image with Markov random field, the prior can be conveniently expressed by Gibbs distribution as

$$P(X) = \frac{1}{Z} exp(-\phi(X)) \tag{4.8}$$

where Z is a normalization factor, guaranteeing that the integral over X is 1, and $\phi(\cdot)$ a non-negative function, inversely proportional to the probability of seeing X. This naturally leads to the similar form of regularization:

$$\hat{X} = \underset{X}{\operatorname{argmin}} \left\{ \|XA - Y\|_p^p + \alpha\phi(X) \right\} \tag{4.9}$$

Note that different regularizations have different prior distributions.

There are large amount of works that employ MAP, many of which combine the algorithms introduced above. Shen et al. [46] present a joint formulation which is built upon the maximum a-posteriori framework that merge motion estimation, segmentation and super-resolution. Farsiu et al. [47] combine the bilateral total variation and MAP regularizations to achieve the goal. Chantas et al. [48] use frequency domain representations to accomplish a fast image registration and apply MAP to obtain the HR images, to name but a few.

4.2. CNN-based single image super-resolution

Methods based on convolutional neural networks are widely employed for inverse problems such as deblurring [49, 50], image denoising [9, 51] and image super-resolution [4, 9, 52–54]. Some deep network architectures employ unique units to enhance or refine the HR output, such as the back-projection unit [54] and the information distillation units [52]. Instead of a straightforward deep CNN architecture, Han et al. [53] attempt to use a recurrent neural network for single image super-resolution.

Here, we highlight the works that inspired us.

4.2.1. ResNet

Empirically, the depth of the network is crucial to the performance of the model. When the number of layers is increased, the network can extract more complex feature patterns, so theoretically better results can be obtained when the model is deeper. It has been found experimentally that the deep network has a problem of performance degradation: when the network depth increases, the network accuracy saturates or even decreased. This is not an over-fitting problem because the training error of a deep network is as high as that of a shallow one. We know that deep networks have problems with gradients vanishing or exploding, which makes deep learning models difficult to train. However, now there are some technical means such as batch normalization to alleviate this problem. Therefore, the problem of performance degradation of deep networks is surprising.

He et al. [55] propose residual learning to solve the problem of performance degradation. For a stacked layer structure, when the input is x, the feature that should be learned is denoted as $H(x)$. In the ResNet, the residual $F(x) = H(x) - x$ is learned instead of directly learning $H(x)$, which can be represented as $F(x) + x$. This is because the residual is easier to learn than the features directly. When the residual is 0, the layer only makes an identity mapping at this point, such that the network performance is guaranteed not to decrease. Actually, the residual will never be 0, which makes the layer to learn new features based on the input characteristics and thus leads to better performance.

The ResNet network [55] architecture is inspired by the VGG19 network. The change is mainly reflected in the fact that Res-Net directly uses the convolution with stride 2 for downsampling, and replaces the fully connected layer with the global average pooling layer. An important design principle of ResNet is that when the feature map size is reduced by half, the number of features is doubled, which preserves the complexity of the network layer. Res-Net adds skip connection between every two layers of the ordinary

network, which forms residual learning. As it goes deeper in the network, the residual learning is performed between three layers. This way, ResNet solves the problem of performance degradation very well.

The residual learning unit proposed by He et al. is employed in our approach which we call a residual block. For the implementing details we refer to section 5.3.2 and section 6.2.1.

4.2.2. Skip connection in autoencoder

Mao et al. [9] introduce a fully convolutional autoencoder architecture with skip connections to deal with single image restoration including denoising and single image super-resolution. Skip connections are applied between encoder and decoder to form the residual learning paths. The network architecture is shown in the following figure:

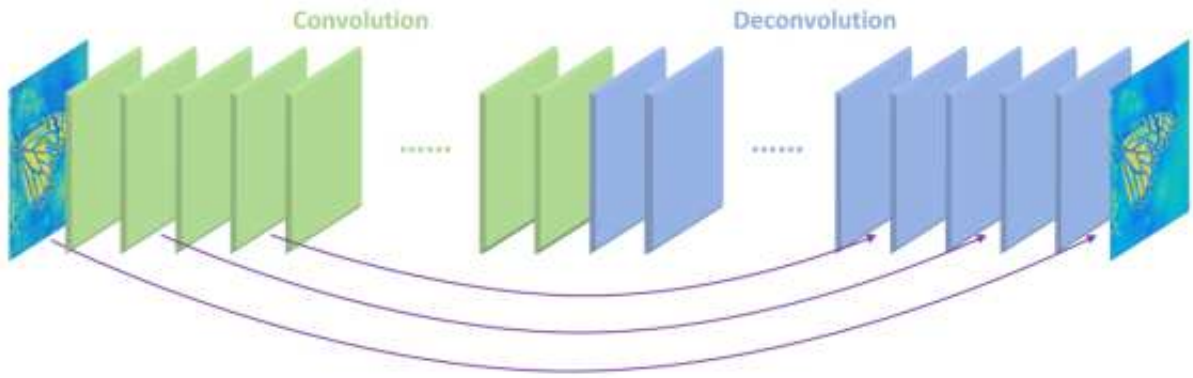


Figure 4.1: *Network architecture in Image Restoration Using Very Deep Convolutional Encoder-Decoder Network with Symmetric Skip Connection [9].*

First of all, convolution is used to extract features. After several convolution layers, the image features are obtained with the noise being filtered out, thereby achieving the purpose of noise reduction. Subsequently, the features extracted are used by deconvolution to perform image reconstruction. In this network, convolution extracts features, and deconvolution upsamples them, thus completing the transition from image to feature and then from feature to image. Secondly, skip connection is employed in an encoder-decoder network, i.e. linking the convolutional layer in the encoder with the corresponding decoder layer via skip connection. This has been mentioned in [55] that as the network deepens, the problem of unstable gradient will occur. Skip connection can be used to form a path for residual learning, thus effectively solving the gradient vanishing or exploding problem.

The authors also compare the performance between the models with and without the skip connection. It is shown that the unstable gradient problem rarely appears and less influences the loss when the number of layers is small. Once the network grows deeper, the performance without skip connection is poor. Therefore, skip connections are proven to be very useful to guide the decoding process and increase detail in image restoration problems by encoder-decoder networks. In our approach, we consequently employ 3D skip connections between our encoder and decoder.

4.2.3. Sub-pixel convolution for upscaling

For a super-resolution task, the upscaling operation is essential and usually done by the deconvolution in a deep network. The deconvolution kernel of size $N \times M$ is centered on the position of each pixel, the surrounding $N \times M$ neighbourhood is initialized with 0, and then deconvolution operation is performed with that kernel. However, the drawbacks of the standard deconvolution operation are apparent. First, the padded zero value does not contain any image-related information, and the computational complexity increases. As [56] and [57] point out, deconvolution operation with strides causes the checkerboard artifact in the reconstructed images. Deconvolution with strides is prone to "non-uniform overlap" therefore produces more "wrongly emphasized" values in some places than somewhere else. In a 2D domain, the uneven overlaps on the two axes form a unique checkerboard pattern.

To prevent such checkerboard artifact caused by deconvolution for upscaling images, Shi et al. [4] propose a sub-pixel convolutional layer that maps low-resolution (LR) image to high-resolution (HR) output.

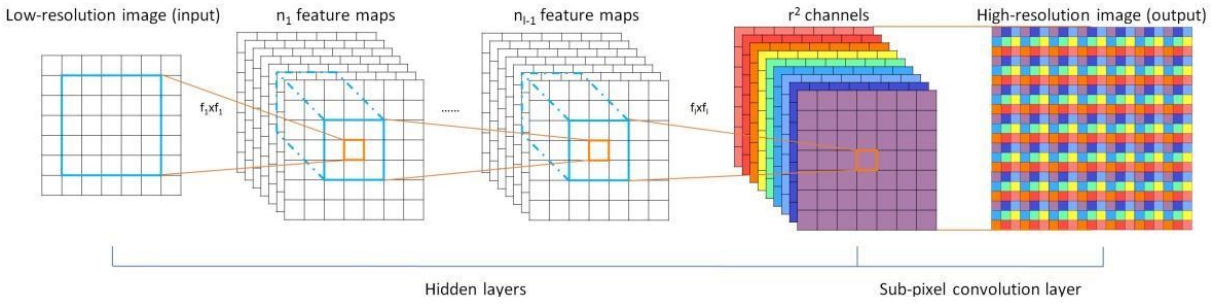


Figure 4.2: *Network architecture in Efficient Sub-Pixel Convolutional Neural Network (ESPCN) [4].*

The central idea of sub-pixel convolution is that for an image of size $H \times W \times C$, the output's dimension of the standard deconvolution operation is $rH \times rW \times C$, where r is the scale factor. The output feature map of sub-pixel convolution is of size $H \times W \times (C \times r^2)$, which makes the feature map consistent with the spatial size of the input image, but increases the number of channels of the convolutional kernel. This strategy utilizes the information of the neighbouring pixels in the input image effectively and avoids the computational complexity introduced by padding 0 at the same time.

Inspired by [4], [56] and [57], we get rid of the checkerboard artifacts by first upscaling the features spatially through bicubic interpolation, thus removing the need to apply strides.

4.2.4. GAN: Generative adversarial network

Ledig et al. [15] employ a generative adversarial network SRGAN to force the network to produce more natural high-resolution images. Their work mentions that although a high PSNR can be obtained using the MSE as the loss function, the recovered image usually loses high-frequency details so that people cannot have an excellent visual experience.

SRGAN uses perceptual loss [58] and adversarial loss [59] to enhance the realism of the recovered picture. Perceptual loss [58] is computed by comparing the features extracted from the generated image and the target image in the layers of a trained network such as VGG19. By minimizing the perceptual loss, the generated image and the target image are more similar in a semantic and style fashion. The task of GAN is: the generator network generates (G) high-resolution images. The images obtained by discriminator network (D) are generated by G or the real high-resolution images in the ground truth data. When the generator can successfully "fool" the discriminator, then the super-resolution is completed through this GAN.

The generator consists of multiple layers of the residual blocks [55], each of which contains two 3×3 convolutional layers, followed by a batch normalization layer. Parametric ReLU is selected as the activation function. Two sub-pixel convolutional layers [4] with scale factor $r = 2$ are used to increase the feature size. In the discriminator, there are 8 convolutional layers. The number of features increases while the feature size decreases as they go deeper in the discriminator network. The activation function is Leaky ReLU. Finally the probability of being predicted as a natural image if obtained by passing the features to two fully connected layers and a sigmoid activation function. The loss function of SRGAN is formulated as

$$L^{SR} = L_X^{SR} + 10^{-3} L_{GAN}^{SR} \tag{4.10}$$

The L_X^{SR} is the content loss which can be either the L^2 loss (MSE)

$$L_{MSE}^{SR} = \frac{1}{r^2 HW} \sum_{x=1}^{rH} \sum_{y=1}^{rW} (I_{(x,y)}^{HR} - G_\theta(I^{LR})_{(x,y)})^2 \tag{4.11}$$

or the perceptual loss

$$L_{VGG(i,j)}^{SR} = \frac{1}{H_{(i,j)} W_{(i,j)}} \sum_{x=1}^{H_{(i,j)}} \sum_{y=1}^{W_{(i,j)}} (\phi_{(i,j)}(I^{HR})_{(x,y)} - \phi_{(i,j)}(G_\theta(I^{LR}))_{(x,y)})^2 \tag{4.12}$$

where $\phi_{(i,j)}$ is the feature map obtained by the j_{th} convolution after activation before the i_{th} max-pooling layer in the VGG19 network. $H_{(i,j)}$, $W_{(i,j)}$ here are the height and width of the features in corresponding layers. The adversarial loss L_{GAN}^{SR} is

$$L_{GAN}^{SR} = \sum_{n=1}^{N} -logD_\omega(G_\theta(I_n^{LR})) \qquad n \in [1, \text{total views}] \tag{4.13}$$

where ω and θ are the parameter sets for generator and discriminator networks, respectively. In our approach we make use of the perceptual loss, utilizing the pre-trained Inception-v3 network [60] instead of the VGG19. As a modification, we introduce our DiffGAN, which not only takes the pixel value of the generated images and the ground truth, but also the derivatives of them. This way, DiffGAN helps to enhance the details of the generated images.

4.3. Light field super-resolution

Due to the need of sacrificing resolution to sample angular coordinates, the light fields usually suffer from the lower spatial resolution compared to standard images. Bishop et al. [61] introduce a variational Bayesian framework for light field super-resolution, closely related to classical approaches [1]. Among recent studies, Wanner and Goldluecke [2] and Pujades et al. [62] propose a variational super-resolution framework using estimated high-accuracy depth maps from epipolar plane images, and solve novel view synthesis as an inverse problem. Likewise, Mitra and Veeraraghavan [63] make use of depth information of the scene to construct an inference model with a Gaussian mixture model prior. Rossi and Frossard [7] adopt a multi-frame approach with a graph-based regularizer.

With the continuing success of deep learning, CNN-based light field super-resolution methods have become common.

4.3.1. Deep convolutional network for light field super-resolution

Yoon et al. [13, 64] propose a new deep learning structure, where the spatial as well as the angular resolution of the light field are increased.

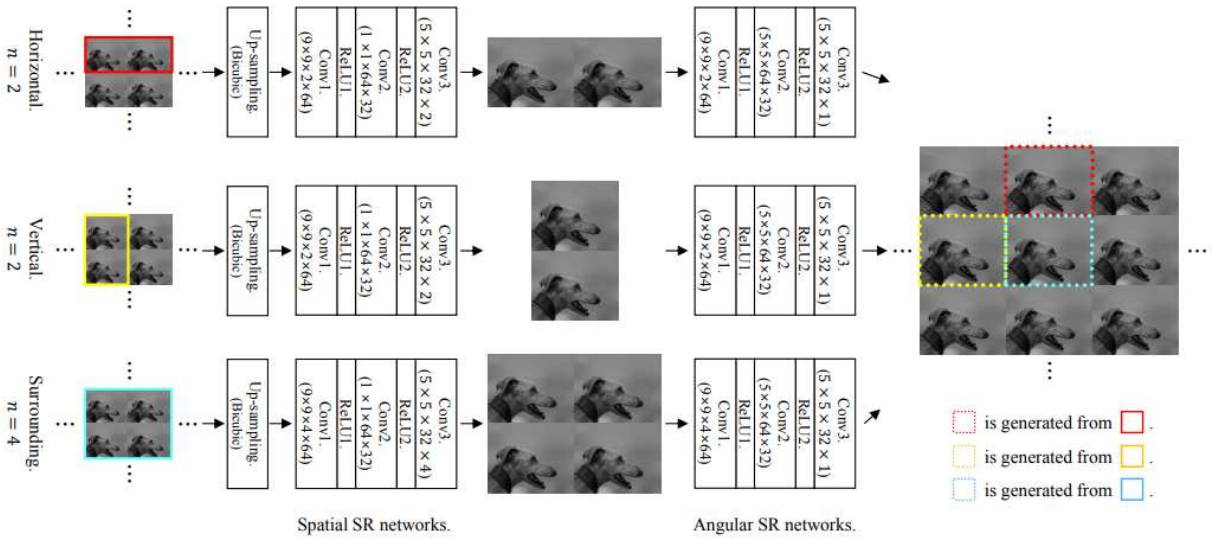


Figure 4.3: *Network architecture in Learning a Deep Convolutional Network for Light-Field Image Super-Resolution (LFCNN) [13].*

The input and output of their spatial SR network are dual-images ($n = 2$) or quad-images ($n = 4$), since generating a novel view requires multiple images. The spatial SR network consists of three convolution layers, in which the kernels are $9 \times 9 \times n$ with 64 channels, $1 \times 1 \times 64$ with 32 channels, and $5 \times 5 \times 32$ with n channels, respectively. The first two layers are followed by ReLU layer. Input images are interpolated by a desired upscale factor before fed into the network. The angular SR network accepts n images from the output of the spatial SR network and produces one novel view. The kernels in

the first, second and third convolution layers are of size $9 \times 9 \times n \times 64$, $5 \times 5 \times 64 \times 32$ and $5 \times 5 \times 32 \times 1$, respectively. Three pairs of spatial- and angular SR networks sharing the same architecture takes three different types of input: a horizontal pair $(n = 2)$, a vertical pair $(n = 2)$ and surroundings $(n = 4)$ and are trained jointly to achieve spatial-angular super-resolution of a light field.

In addition, they train the model with the Y-channel of the images in the *YCbCr* colorspace, interpolate the other two color channels to the desired resolution and finally concatenate them to a super-resolved output. This strategy saves computational resources and also leads to competitive results. We compare the network performance in *RGB* and *YCbCr* color spaces in the preliminary experiments and finally trained the proposed network with *YCbCr* images for better results.

4.3.2. Deep CNN with low-rank prior for light field super-resolution

Farrugia and Guillemot [65] upscale the whole light field spatially by utilizing a low-rank approximation restored by a CNN.

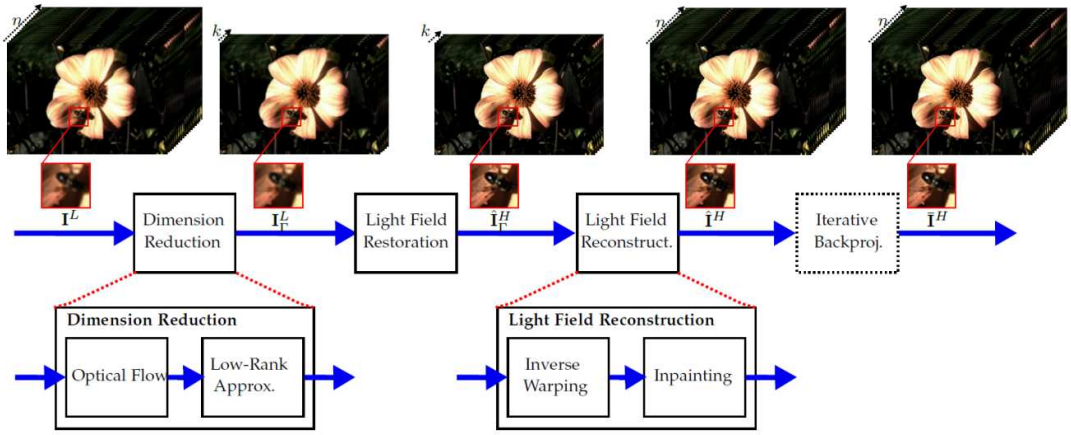


Figure 4.4: *Network architecture in Light Field Super-Resolution using a Low-Rank Prior and Deep Convolutional Neural Networks [65].*

Their approach consists of three parts: dimensionality reduction, reconstruction and refinement. Since light field has a redundant structure, they use the optical flow to align sub-aperture views I_i to the center view I_c by

$$I_c(x, y) = I_i(x + u_i, y + v_i) \quad i \in [1, \text{total views}], \quad i \neq c \tag{4.14}$$

Given the flow vectors, the dimensionality reduction can be formulated as

$$\underset{A^L}{\operatorname{argmin}} \|\Gamma_{u,v}(I^L) - A^L\|_2^2 \quad \text{with} \quad rank(A^L) = k \tag{4.15}$$

$\Gamma_{u,v}$ and A^L serve as the forward-mapping operator and the combination weight matrix, which is the low-rank representation of the aligned low-resolution light field. Similarly,

they denote A^H as the low-rank representation for the aligned HR light field. These two terms are used in their network as training data with the target function

$$E_\theta(\theta, A^L, A^H) = \frac{1}{2}\|A^H - \text{CNN}_\theta(A^L)\|_2^2 \tag{4.16}$$

The first convolutional layer has the kernel of size $3 \times 3 \times k \times 64$ while the last kernel is of size $3 \times 3 \times 64 \times k$ to reconstruct the high-resolution light field. All the other layers use 64 filters with size $3 \times 3 \times 6$.

In the reconstruction phase, the optical flow is used for computing the inverse wrapping such that all the sub-aperture views are brought back to normal from the low-rank representation of the aligned super-resolved light field. During the inverse wrapping, the reconstructed images suffer from cracks and holes due to the occlusions, so the authors fill in the missing values by employing the epipolar images, in which information available in other views is diffused. The refinement is done by iterative back-projection, which is similar to that of [54].

4.3.3. Light field intrinsic decomposition

The recent works of Alperovich et al. [10, 12] successfully deal with the vast scope of light fields by using a encoder-decoder architecture and patches instead of a entire image. Although they are not for light field super-resolution, we are enlightened by their network architecture.

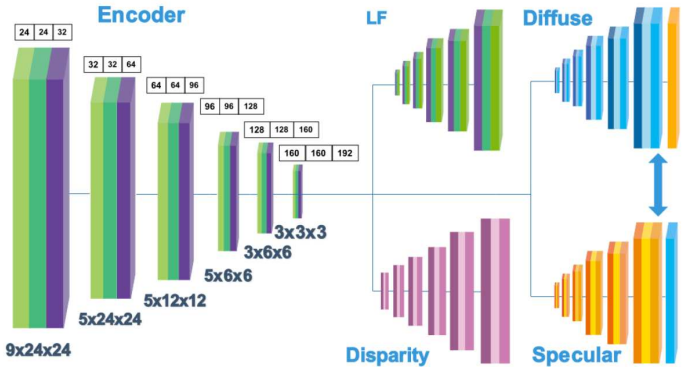


Figure 4.5: *An asymmetrical encoder-decoder architecture for multiple tasks in Light field intrinsics with a deep encoder-decoder network [10].*

A fully convolutional encoder decreases the resolution of the patch-wise 48×48 horizontal and vertical epipolar volumes by strided convolution in the residual block [55], instead of max-pooling in each layer. There are 18 layers in total in the encoder, where every third one reduces the patch resolution via strided convolution, increasing at the same time the number of features. The encoded volumes have shape $3 \times 3 \times 3 \times 192$ when they reach the latent space. Horizontal and vertical epipolar volumes are encoded separately while they share the same weights in the encoder because they have the exact same structure after the horizontal one is spatially transposed.

There are four decoders linked to the latent space. They are trained jointly for tasks

light field reconstruction, disparity estimation and intrinsic decomposition to diffuse and specular. Each decoder has the symmetrical structure as the encoder while the strided convolution in the residual blocks is substituted by the deconvolutions with strides.

Their works greatly inspire our approach. First, the autoencoder, unlike the straightforward neural network architectures, shrinks the size of the input data in the encoding process, which is of significant use to process huge data such as the light fields. Secondly, larger light fields are segmented into small patches. While in this way the network can deal with light fields of any shape, the features are not learned at object level thus are more representative. At last, because there are sufficient information for super-resolution in the two epipolar volumes, the complexity brought by the redundant structure of light field is reduced. All these three aspects leave more space for the number of features and batches, especially for high-dimensional training data like light fields.

Chapter 5

Data Preparation and Preliminary Experiments

5.1. Datasets for training

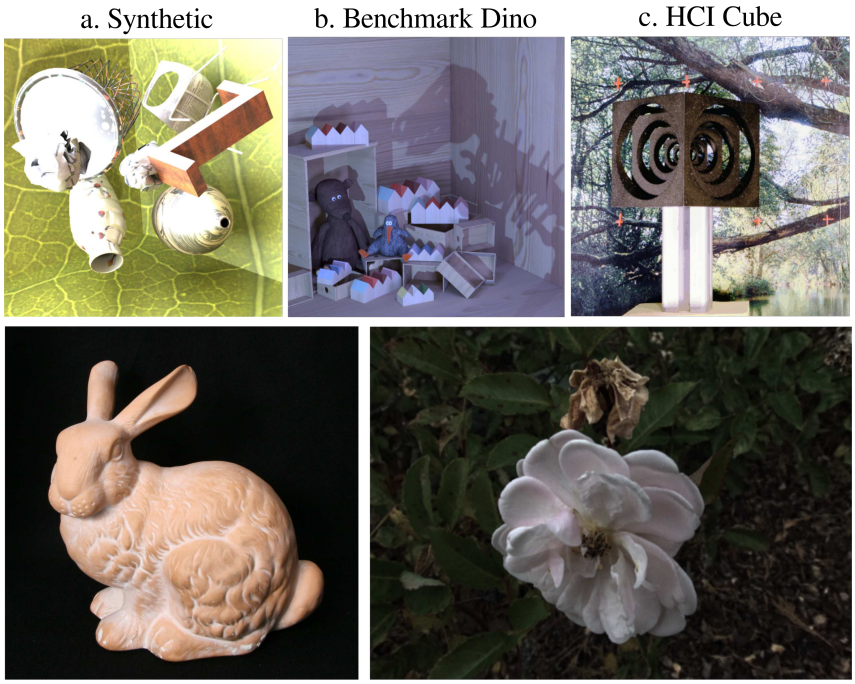


Figure 5.1: *Examples of the light fields used for training.* ***a.*** *Synthetic scenes and* ***b.*** *Benchmark light fields [66] are generated using the Blender add-on. There are multiple objects in random positions and a background with complex texture under environmental illumination in the scenes.* ***c.*** *HCI gantry [67] has one object per light field with a picture of nature as the background. Likewise,* ***d.*** *Stanford light fields [19] have only one object in the scene, but the background is uniformly black or grey.* ***e.*** *Lytro light fields are real-world data captured by the Lytro Illum camera. Comparing to the other light fields, the real-world data has different illumination conditions and noises, thus is a more challenging dataset.*

The training data is generated using the Blender add-on provided with [66] and follows the same procedure as in [12] for generating random light fields from a number of template scenes, objects and textures. We generated a total of 750 of these random light fields for training. Although placing random textures on objects leads to unrealistic objects usually not encountered in the real-world, we believe that for small patches (Fig. 5.3) and for the task of super-resolution this is not a drawback, as the network will probably not learn at object level. Resolution and disparity range were chosen such that they allow us to downscale the spatial component of the light fields by a factor of 2 and 4, respectively, while still keeping disparities readily distinguishable. In addition, we use publicly available data from the HCI database [67], the 4D light field Benchmark [66], the Stanford multi-camera array [19], and light fields captured with the Lytro plenoptic camera.

5.2. Pre-processing

Since we are using the light fields in their original resolution as the ground truth, the LR input data must be generated by downscaling the original ones. In order to avoid the aliasing and maintain the image quality as much as possible in the low-resolution version, we applied the bicubic downscaling, instead of simply selecting the every second (or fourth if the scale factor is 4) row and column from the HR images.

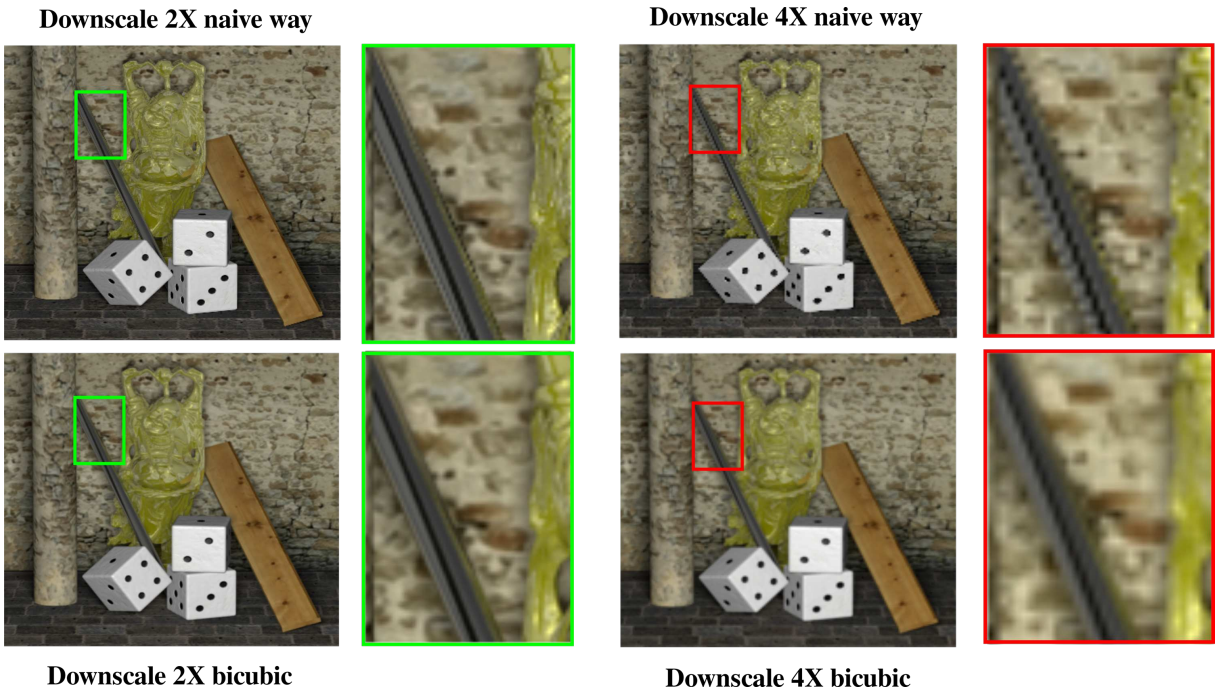


Figure 5.2: *The top row shows the low-resolution images of HCI-blender Buddha [67] downscaled by selecting every second row/column and every fourth row/column from the original image (naive way), while the bottom row illustrates the results of the bicubic 2x and 4x downscaling. An aliasing is obvious on the edge of the object in the zoomed-in areas of the images downscaled in a naive way.*

Like Alperovich et al. [10] did, we furthermore split the low-resolution images into 48×48 patches. This size ensures that in the downscaled images each patch contains at least some parts of the main object of the scene. Otherwise, many patches would possess the background only, such as the uniformly black one of the Stanford dataset (Fig. 5.1d), or too complicated structure in the scene, like the synthetic light fields (Fig. 5.1a). The patch extraction procedure is illustrated in Fig. 5.3

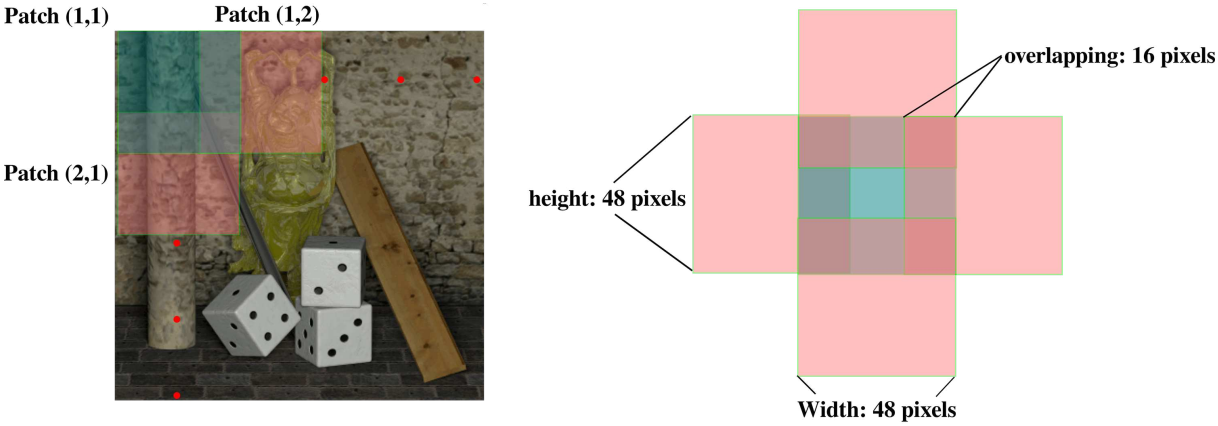


Figure 5.3: *The low-resolution input to the network consists of the patches of size* 48×48*. They are extracted from each image in the cross-hair with an overlapping area of 16 pixels with each of its neighbouring patches. The high-resolution center view is split into patches, with size* 96×96 *for scale factor 2 or* 192×192 *for scale factor 4, as input ground truth.*

5.3. Preliminary experiments

In the preliminary experiments, we build a simple convolutional network that generates only the super-resolved center view from the light fields. This network has an autoencoder structure followed by an upscaling net. All the layers in the encoder are downscaling the features either spatially or angularly. The upscaling operations in the decoder and upscaling net are performed by the stride-2 convolution transpose (deconvolution). Except the skip connection [9] and residual block [55], no other enhancing units are employed in this network, see Fig. 5.5, thus allowing us to observe the unique advantage brought by the light field comparing to other single image super-resolution approaches.

Figure 5.4: *Preliminary experiment: Each light field used for training has* $9 \times 9 = 81$ *views of the scene [67]. Here we aim at using the low-resolution epipolar volumes in vertical and horizontal directions (in red and green frames, respectively) to generate 2x super-resolved center view. These two epipolar volumes should be sufficient supplying the neighbouring information around the center view.*

Another intent of the preliminary experiment is to analyse the influence of the color spaces on the image restoration problems by CNN-based approaches. Namely to compare the results of this network trained with the data in different color spaces and determine a suitable one for our next work.

5.3.1. Color spaces

The *RGB* images and gray-scale images are frequently used for training CNN in computer vision tasks. In practice, it turned out that the *RGB* space is not sufficient due to its simplicity and extensiveness. In addition, *RGB* is an additive color space, one specific color must be represented by the sum of the values of all three channels. Each channel does not have its own meaning but only the proportion of the red, green or blue color. Since the neural network is sensitive to the input data, once the training data contains images of certain color tone and the test data has another, the results would have a notable bias towards the training data. From this perspective, converting the *RGB* images into the color spaces in which every channel has its own meaning should be considered.

HSV is a well-known color space. The *H* is for hue, *S* indicates the saturation and *V* stands for brightness and they could be easily converted from and to *RGB*. The *H* value converted from the *RGB* color space is not absolute for certain hue, for example the color hue red can have the value $(0°, 1, 1)$ or $(360°, 1, 1)$. This leads to the problem that some values in the training data do not have unique ground truth and the losses of training oscillates or even does not converge. The *HSV* color space is proven to be not viable for the purpose of this project.

Apart from the *HSV*, there are other popular color spaces whose channels have different meanings while the values are unique as well, to name but a few, *CIELab*, *YCbCr* and *YUV*. In *CIELab*, the *L* value is the lightness, *a* and *b* values correspond to green–red and blue–yellow color components, respectively. *YCbCr* and *YUV* color spaces are very similar. The *Y* channel is the luminance component and can be seen as a gray-scale copy of the original color image, in which the details of the image are preserved. *Cb* and *Cr* are the blue-difference and red-difference chroma components in *YCbCr* color space.

Table 5.1: *Value range of different color spaces*

Color Space	*RGB*			*CIELab*			*YCbCr*		
Channels	*R*	*G*	*B*	*L*	*a*	*b*	*Y*	*Cb*	*Cr*
Min Value	0	0	0	0	-128	-128	16	16	16
Max Value	255	255	255	100	+127	+127	235	240	240

The training data for the preliminary experiment are the 4D light field Benchmark [66] and the synthetic scenes in *RGB* format. If other color spaces are used in the experiments, the *RGB* images are converted to the desired color space and presented to the network. The final output of the network is then in the same color space. For the evaluation, the *RGB* test data are converted to the corresponding color space. The PSNR and SSIM are calculated between the output and ground truth there. Additionally, the output is converted back to *RGB* and evaluated again with the original *RGB* ground truth of test data.

5.3.2. 3D-2D super-resolution autoencoder

In order to take the full advantage of the rich information in the data generated from the cross-hair, a 3D-2D convolutional autoencoder followed by a deconvolutional spatial upscaling network is constructed (Fig. 5.5).

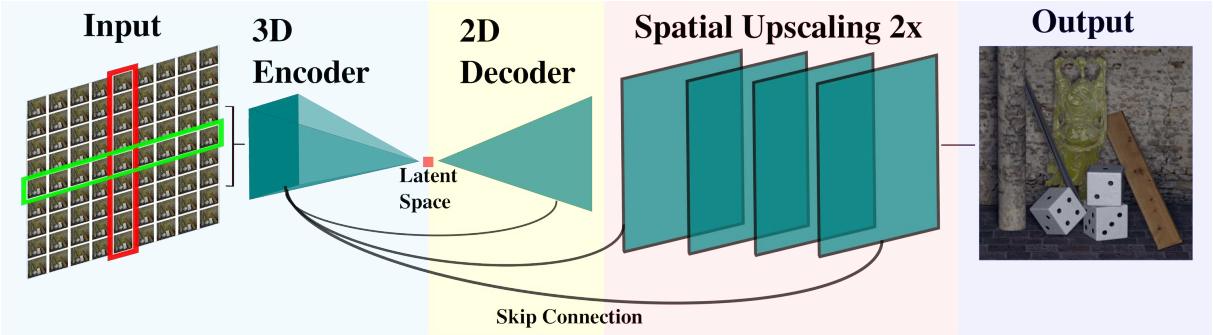


Figure 5.5: *The preliminary 3D-2D autoencoder: The inputs are the patch-wise vertical and horizontal epipolar volumes from the low-resolution light field cross-hair. The high-resolution patches from the center views are fed as ground truth. The 3D encoder downscales the input in angular and spatial coordinates (Fig. 5.6). At each layer, the encoded vertical and horizontal volumes are concatenated to 2D to form the volume for skip connection, see Fig. 5.7. The latent-space (bottleneck) of the autoencoder is flattened and decoded in 2D till the features have the same spatial resolution as the input patches. The upscaling network consists of four layers of deconvolution. Here the features are spatially 2x upscaled and refined. Finally we obtain the 2x super-resolved center views.*

The epipolar volumes consist of 9 patches in vertical and horizontal directions from the corresponding views respectively. The horizontal patches are transposed such that they have the same viewpoint movement as the vertical ones. This way, the vertical and the horizontal patches are sharing the variables of convolutions in the encoder. The 3D encoder has seven layers, whose building block is the residual block (Fig. 5.6). The residual block is known to be helpful to deal with vanishing or exploding gradient during the training and to learn identity mapping that provides reasonable preconditioning [55]. The odd layers of the encoder downscale the features spatially by applying the 3D convolution with stride (1, 1, 2, 2, 1) [1]. As the result, the spatial resolution of the features is reduced to half when they go through these layers. The even layers are responsible for the angular downscaling from 9 views to 3 views by the means of 3D convolution with stride (1, 2, 1, 1, 1). The number of the features grows gradually from 3 to 192.

In the encoding phase, the vertical and horizontal volumes are not only downscaled by strided convolution, but also transformed to a volume for the skip connection. In each layer of the encoder, the views in the vertical or horizontal volumes are firstly superimposed, then these two volumes are concatenated to a single one. Finally we apply

[1] The 3D convolution or deconvolution in TensorFlow accepts the stride in format (**batch, angular dimension, height, width, channel**).

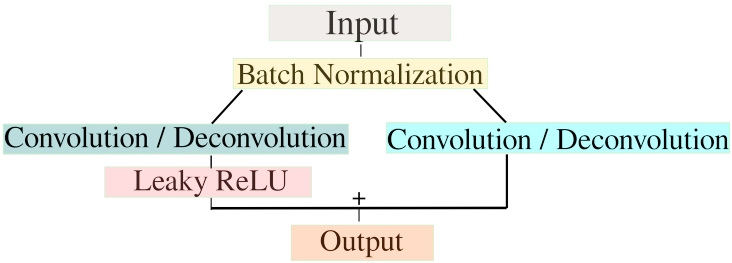


Figure 5.6: *Illustration of the residual block. The input to the layer goes though batch-normalization. In the encoder, the left block performs the 3D convolution of kernel size* $3 \times 3 \times 3$ *and followed by leaky ReLU. The right block is applied to the input only if the number of features or the resolution of the input is changed by the previous layer. The outputs of both sides are element-wisely summed up to form the final output of the residual block. The convolution is changed to deconvolution in the decoder while the rest of the procedure remains the same.*

a 1×1 convolution to reduce the features such that the skip connection volume has the same number of features as the epipolar volumes in this layer (Fig. 5.7).

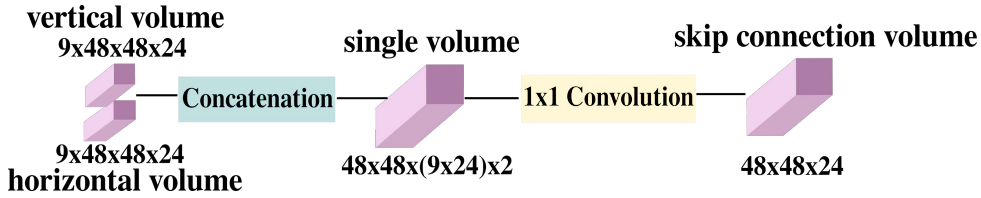


Figure 5.7: *Example creating skip connection patch at the second layer of the encoder. In this layer, the epipolar volumes have 24 feature maps of size* 48×48*. The views of both volumes are superimposed so each of them is transformed to* $48 \times 48 \times (9 \times 24)$*. The transformed vertical and horizontal volumes are then concatenated to a single one with* $9 \times 24 \times 2$ *features. Now this volume does not have the angular dimension therefore it convolves with an* 1×1 *kernel which brings its number of features to 24 again, thus the skip connection volume for the second layer between the encoder and decoder.*

In the latent-space of the autoencoder, the flattened vertical and horizontal features are concatenated and reshaped to 2D. In the decoder, they are upscaled spatially only when the corresponding layer in the encoder applied spatial strided convolution. At the layers whose corresponding encoding-layers angularly downscaled the input, the decoded features deconvolve with a 3×3 kernel of stride 1 to get the correct number of features, which should be 3 times as many as the skip connection patches. The reason for such design is that otherwise, the skip connection would dominate the decoding.

At the last layer of the 2D decoder, the features have the same spatial resolution as the input. They are sent to the deconvolutional network and upscaled to the desired spatial resolution, which is 96×96 with scale factor 2. The input vertical and horizontal volumes are upscaled by the bicubic interpolation to high-resolution and treated in the same way as Fig. 5.7 shows. This upscaled input is concatenated afterwards with the

output features from the first layer of the upscaling network and together go through the next two deconvolutional layers. There the number of features is gradually reduced, without their spatial resolution being changed. The low-resolution center view in the input volumes is bicubic interpolated to high-resolution and concatenated with the features in the 4th layer of the upscaling network. As the last step, a 1×1 convolution is applied to the concatenated volume which reduces the number of features to 3 to obtain the super-resolved center view.

The cost function is simply the L^2 loss between the ground truth center view and the output of this network.

5.3.3. Configuration of training

Training data The training dataset contains twenty light fields from the 4D light fields Benchmark [66] and 200 synthetic light fields. As Sec. 5.2- Fig.5.3 described, all the images are split into 48×48 patches, so there are 43120 training samples in total.

Training process The optimization is performed with the ADAM optimizer where the initial learning rate is $1e-4$. The GPU capacity is still limited therefore only the batch size 20 was used.

The trained model is saved every 100 iterations with the initial learning rate. After the training loss converges, the learning rate is dropped to $1e-5$ and then the model is saved every 5 iterations so that the training can be stopped in time.

Choice of color spaces The *RGB* and *YCbCr* color spaces are used in training. The *RGB* color space plays an important role because the most CNN-based image super-resolution methods are using it. The *YCbCr* is selected to represent the color spaces in which the channels have their own meaning. An important reason for using the *YCbCr* is revealed in the Tab. 5.1. Namely, the channels have the similar value ranges as the *RGB* channels which make the comparison between these two color spaces easier. The recent works of Yoon et al. [13] and Timofte et al. [8] also support this choice.

5.3.4. Experiment results

Two models were trained with the network, one for the *RGB* color space and another for *YCbCr*. In the evaluation, the output patches are reassembled to a complete high-resolution image. The constructing process of a high-resolution center view takes approximately 3 seconds per light field.

The PSNR and SSIM are used as the metrics. For the *YCbCr* color space, the images are not only evaluated in their own color space but also converted back to *RGB* color space and evaluated again[2]. The results of the preliminary network and other light field or single image super-resolution methods are shown in Tab. 5.2 and Tab. 5.3 [3], using the HCI-blender [67] and the Stanford [19] datasets for testing.

Here we choose the bilinear interpolation and other four state-of-the-art image super-resolution approaches to compare with our preliminary network. The VarSR is the variational approach [62] based on [2]. GMM represents the Gaussian-mixture model method [63] by Mitra and Veeraraghavan. DCSR, as the name implies, is the deep convolutional super-resolution for single image [68]. GB-SQ is the recent graph-based approach [7] by Rossi and Frossard.

Table 5.2: *HCI Dataset - PSNR for the Super-Resolution Scale Factor* $\alpha = 2$

Lightfield	Bilinear	VarSR [62]	GMM [63]	DCSR [68]	GB-SQ [7]	RGB	RGB*	YCbCr
buddha	35.22	38.22	39.12	37.73	39.00	**40.22**	38.84	***44.93***
buddha2	30.97	33.01	33.63	33.67	34.41	**36.06**	35.26	***40.60***
horses	26.37	29.14	**33.13**	27.80	32.62	30.75	30.87	***36.90***
medieval	30.07	30.53	33.34	31.23	33.45	28.14	**34.31**	***40.10***
mona	35.11	37.54	38.32	39.07	39.37	38.49	**39.60**	***45.40***
papillon	36.19	39.91	40.59	39.88	**40.70**	38.28	33.07	39.52
stillLife	25.28	25.58	28.84	27.27	**30.98**	28.37	28.51	***34.49***

Table 5.3: *Stanford Dataset - PSNR for the Super-Resolution Scale Factor* $\alpha = 2$

Lightfield	Bilinear	VarSR [62]	GMM [63]	DCSR [68]	GB-SQ [7]	RGB	RGB*	YCbCr
beans	47.92	23.28	36.02	**52.01**	48.41	40.04	38.33	44.21
bulldozer	34.94	22.82	32.05	**39.76**	35.96	33.53	32.42	38.27
bunny	42.44	26.22	40.66	**47.16**	47.01	37.32	33.39	39.44
eucalyptus	34.09	25.04	34.90	36.69	**39.09**	35.25	29.24	35.35
treasure	30.83	22.81	30.52	34.16	**37.51**	33.02	29.57	35.68
truck	36.26	25.77	37.52	39.11	**41.57**	37.66	35.92	***42.00***

The training data for these two models are synthetic, so the network performed well on

[2] The PSNR of the images that are trained and evaluated in *RGB* is marked as **RGB**. The results for the images which are converted back from the *YCbCr* color space are shown in the **RGB*** column and the results of the images that are trained and evaluated in *YCbCr* color space are in the **YCbCr** column of the tables.

[3] The **bold font** marks the best results in *RGB* color space. The results in *YCbCr* are marked in ***bold italic font*** if they are beyond the results in *RGB*.

HCI-blender dataset [67] where light fields are only synthetic scenes. This preliminary network is close to the other state-of-the-art approaches on the Stanford dataset [19]. The network trained in *YCbCr* produces better results than the *RGB* version on Stanford [19] and HCI-blender [67] datasets, furthermore, it outperforms the state-of-the-art in seven out of thirteen light fields in the term of PSNR. The results of this preliminary network quantitatively excel in SSIM in both color spaces, see Tab. 5.4. Additionally, it is notable that this preliminary network beat the single image super-resolution methods, namely the bilinear interpolation and DCSR [68] in all light fields of HCI-blender dataset [67] and in 4 out of 6 light fields of the Stanford dataset [19].

Table 5.4: *SSIM for the Super-Resolution Scale Factor* $\alpha = 2$

	RGB	RGB*	YCbCr
HCI Dataset			
buddha	0.97	0.97	**0.99**
buddha2	0.96	0.96	**0.98**
horses	0.92	0.92	**0.97**
medieval	0.91	0.91	**0.97**
mona	0.96	0.97	**0.99**
papillon	0.95	0.93	**0.97**
stillLife	0.87	0.88	**0.95**
Stanford Dataset			
beans	0.98	0.97	**0.99**
bulldozer	0.88	0.87	**0.97**
bunny	0.87	0.75	**0.98**
eucalyptus	0.84	0.44	**0.95**
treasure	0.84	0.64	**0.96**
truck	0.94	0.92	**0.98**

One can notice that the SSIM of the constructed high-resolution images with respect to the ground truth is great. The model trained in *YCbCr* color space outperforms the model trained in *RGB* in terms of SSIM on all light fields. Compared to state-of-the-art CNN-based single image super resolution approaches [4, 9, 52–54], according to the results reported there, our preliminary network in *YCbCr* color space outperformed them both in terms of PSNR and SSIM by far.

Four light fields (Fig. 5.8 - Fig. 5.11) are picked out to be visualized and zoomed to show the quality of the super-resolved center views. The **Medieval** has the highest PSNR in the comparison (Tab. 5.2) even after being converted to *RGB* color space. The *YCbCr* version of **Buddha**, **Buddha2** and **Horses** also took the leading positions in that comparison, while their converted *RGB* versions only have a small distance to the winners in terms of PSNR.

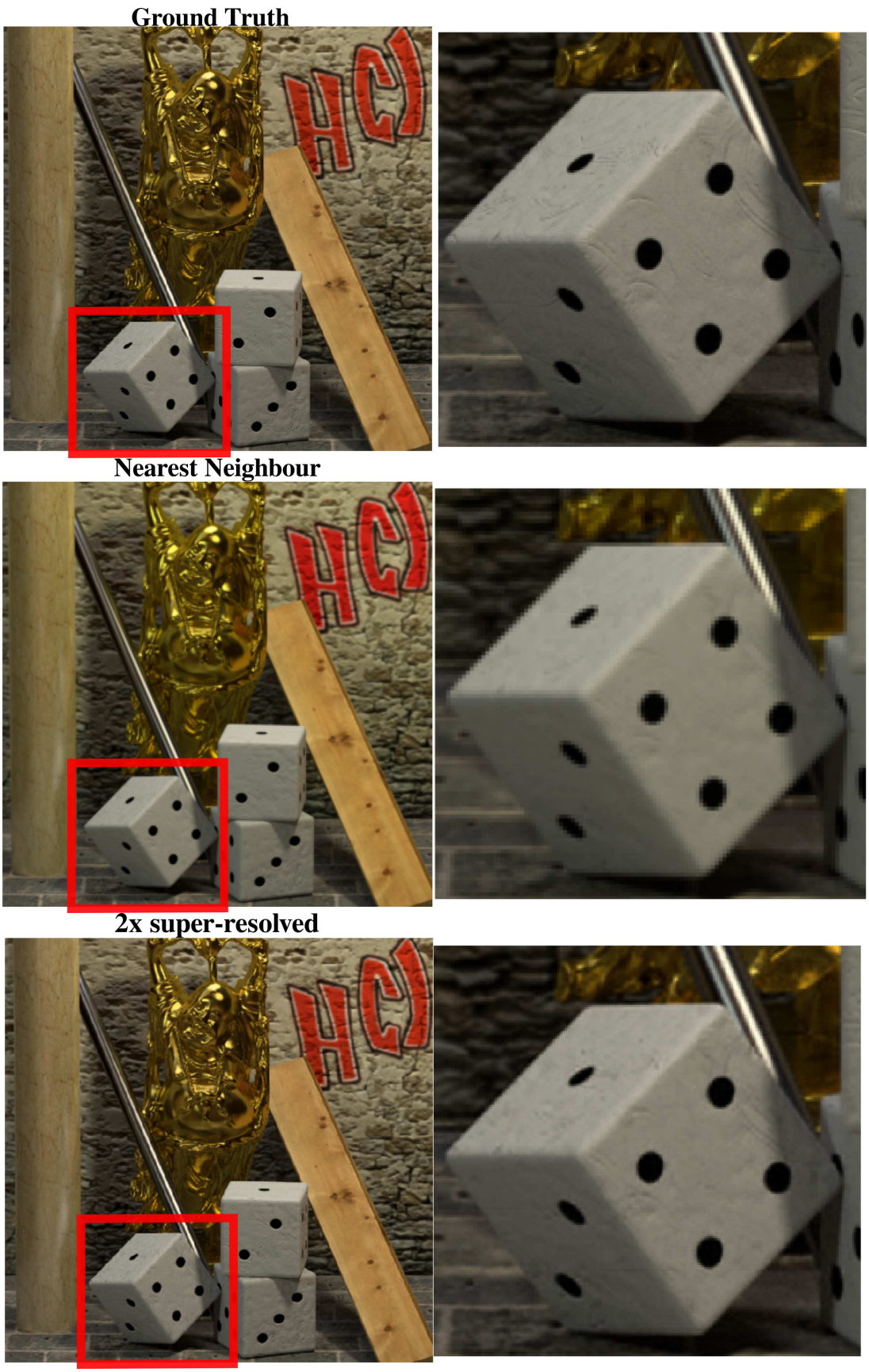


Figure 5.8: ***Buddha 2*** *with* $PSNR = 35.26, SSIM = 0.96$.
Top row: Ground Truth center view and zoomed-in area of the dice.
Middle row: 2x NN interpolated center view and zoomed-in area of the dice.
Bottom row: 2x super-resolved center view and zoomed-in area of the dice.

Figure 5.9: ***Medieval*** *with* $PSNR = 34.31, SSIM = 0.91$.
Top row: Ground Truth center view and zoomed-in area of the wooden window and gravel ground.
Middle row: 2x NN interpolated center view and zoomed-in area of the wooden window and gravel ground.
Bottom row: 2x super-resolved center view and zoomed-in area of the wooden window and gravel ground.

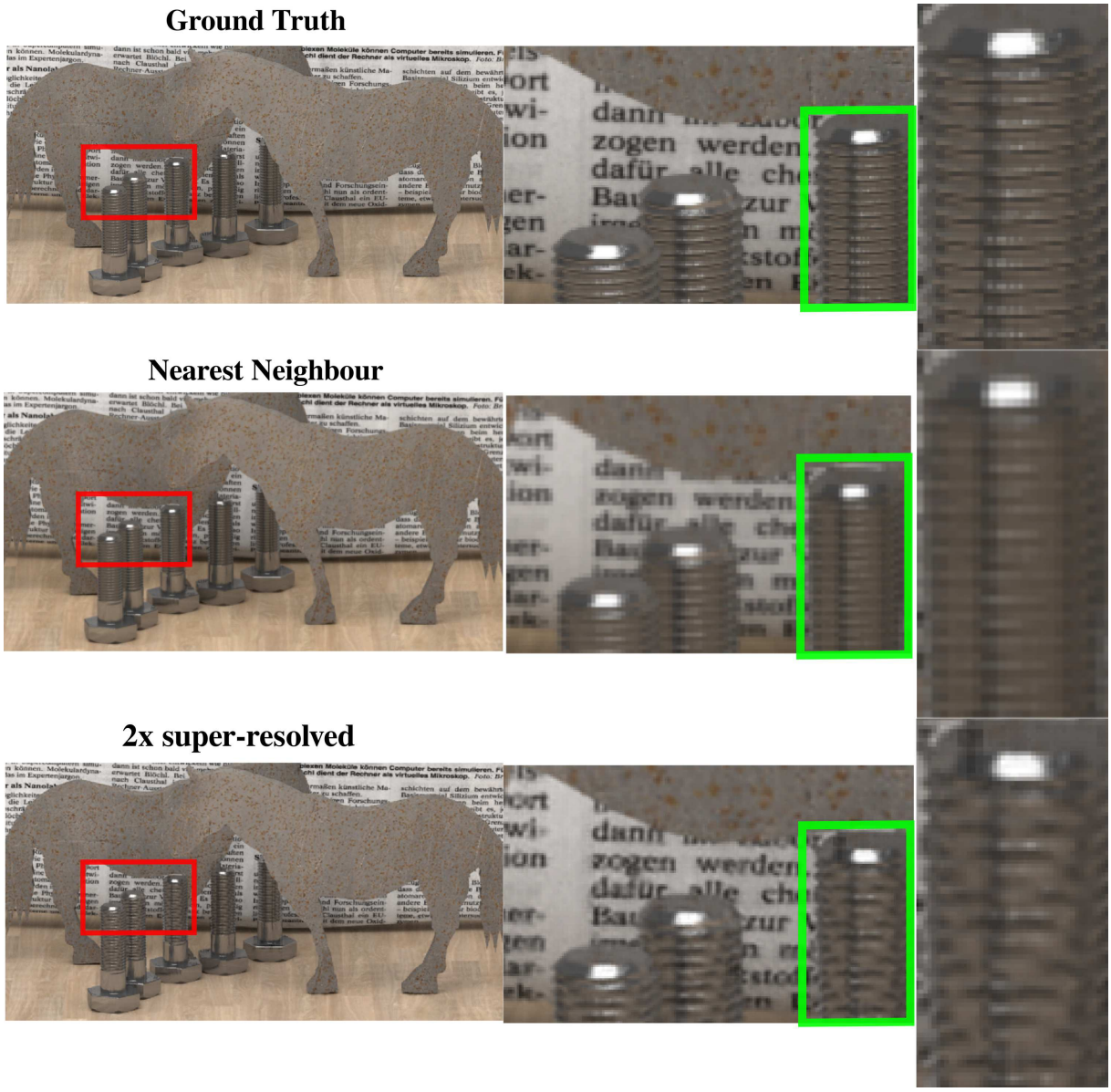


Figure 5.10: ***Horses*** *with* $PSNR = 30.87, SSIM = 0.92$.
Top row: Ground Truth center view and zoomed-in area with detail of the screw.
Middle row: 2x NN interpolated center view and zoomed-in area with detail of the screw.
Bottom row: 2x super-resolved center view and zoomed-in area with detail of the screw.

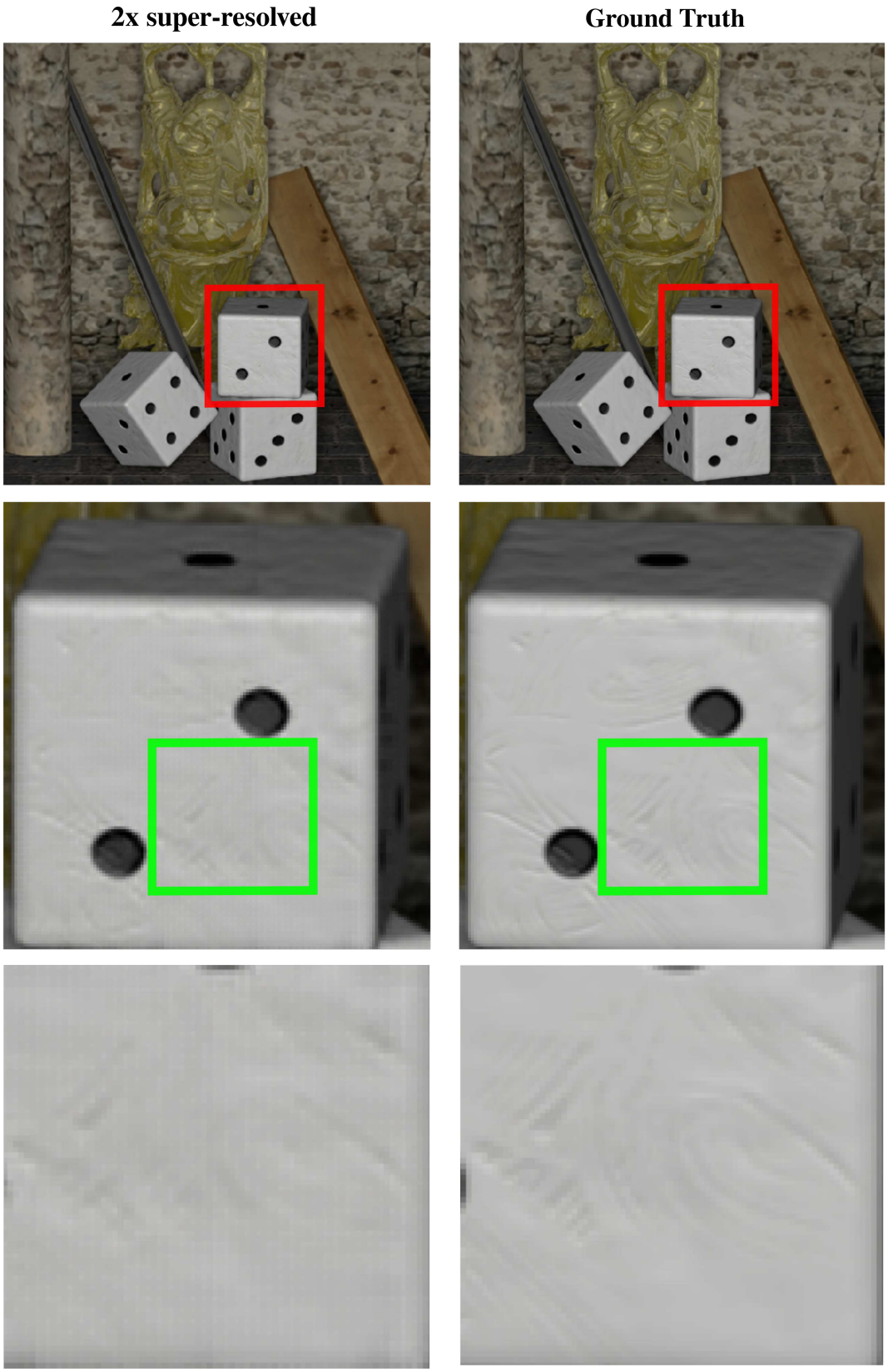


Figure 5.11: ***Buddha*** *with* $PSNR = 38.84, SSIM = 0.97$.
Left column: 2x super-resolved center view and zoomed-in area of the dice with detail.
Right column: Ground Truth center view and zoomed-in area of the dice with detail.

5.3.5. Findings in visualization

The super-resolved center views being visualized above have very good PSNR and SSIM, for example our preliminary network produced brilliant visual results on the Buddha2 light field [67] (Fig. 5.8), which matches its PSNR (35.26) and SSIM (0.96). However, in the visualization we can observe some phenomena that are not reflected by the PSNR. Fig. 5.9 shows the output center view of the Medieval light field [67]. The zoomed-in wooden window part is apparently not as sharp as that in the ground truth. In the second row of Fig. 5.10, we can see that the letters in the background are quite well reproduced while the screws in the foreground lost lots of details. The screw thread of the super-resolved center view is lack of contour and texture. Our preliminary network performed very well on the Buddha light field [67] in terms of PSNR (38.84), yet we unearthed the checkerboard artifacts in the results which is visualized in the bottom left of Fig. 5.11.

$$PSNR(I,U) = 10 \cdot log_{10}\left(\frac{MAX^2}{MSE}\right) = 20 \cdot log_{10}(Max) - 20 \cdot log_{10}(MSE) \tag{5.1}$$

Where I and U are the images to be compared. H and W represent the height and width of the image, respectively. MAX is the maximum in the pixel value range. Recall that MSE is equivalent to L^2 loss (equation 3.2), so the PSNR is just inversely proportional to it.

Compared to other machine learning tasks, such as object detection or instance segmentation, the definition of loss function for learning tasks in super-resolution is usually relatively simple, because the purpose of our reconstruction is to make the PSNR between the generated HR image and the real HR image as large as possible. So most of the deep learning based super-resolution studies will directly design the lose function as MSE, which is the mean square error between all the corresponding pixels of the two pictures. Since MSE loss requires one-to-one correspondence between the pixels, it is also called per-pixel loss. As technology evolved, researchers slowly discovered the limitations of per-pixel loss. Consider an extreme case, shifting the original HR image by one pixel in any direction. In fact, the resolution and style of the image itself have not changed much, but per-pixel loss will be shifted by this pixel. There is a significant rise of the per-pixel loss therefore it does not reflect the high-level characteristics of the image.

The visualization on the one hand reveals that PSNR might not comprehensively reflect the quality of the super-resolved image, i.e. the per-pixel L^2 loss is also not sensitive to the details in the image thus not sufficient in the image generating tasks. On the other hand, the checkerboard artifact is produced because we employed the stride-2 convolution transpose in the decoder and upscaling phase of the preliminary network [4] therefore should be avoided in the next work.

Chapter 6

Deep Autoencoder for Light Field Super-Resolution

It is proven in the preliminary experiments that the redundant structure of the light fields is competent for spatial super-resolution, where the neighbouring views contribute rich information around their center view. However, the information contained therein is much more than only for the spatial super-resolution, so we modified the network architecture in the preliminary experiment to achieve the light field super-resolution in the spatial as well as in the angular domain, see Fig. 6.1.

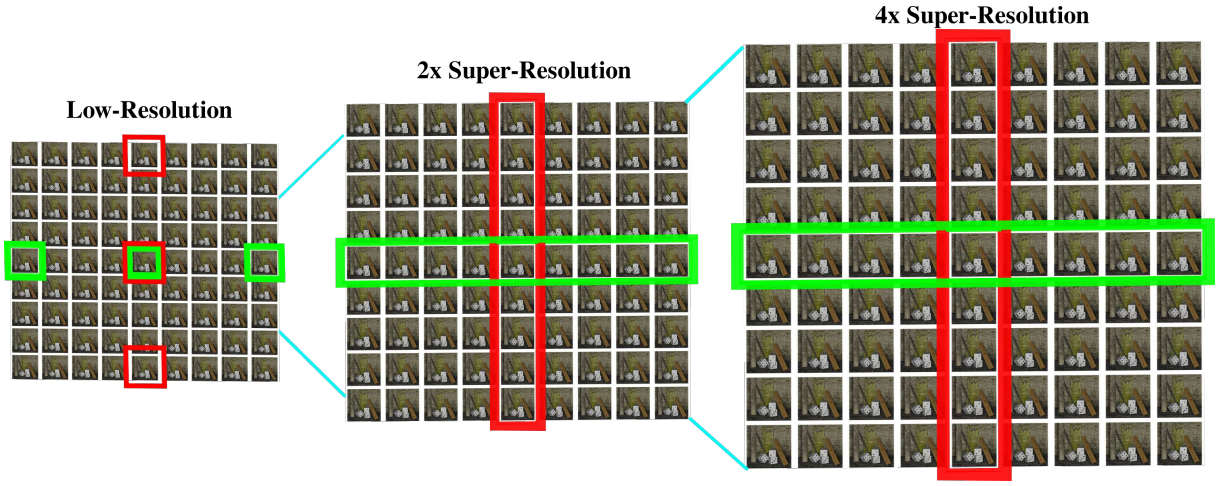


Figure 6.1: *Spatial and angular light field super-resolution: We use 3 low-resolution views in each epipolar volume as our input and generate the full "cross-hair" with 2x and 4x upscaled spatial resolution.*

In the meanwhile, the findings in the preliminary experiments are taken into account for the new architecture. First, we use the bicubic interpolation for upscaling instead of the stride-2 convolution transpose to avoid the checkerboard artifact (Sec. 6.2.2). Second, the DiffGAN (Sec. 6.1.2) is introduced to enhance the details of the image. Lastly, we train the model in the *YCbCr* color space and convert the output back to *RGB*.

6.1. Outline of model and losses

In this section, we present an overview of the general structure of our deep neural network model for the light field super-resolution, as well as the key formulas used to construct the loss functions of the network. We discuss our modifications to the typical encoder-decoder model and discriminator loss as introduced in [59].

6.1.1. Encoder-decoder model for light fields

Let L_l be the low-resolution version of the light field L. With the convolutional encoder network E, we project L_l onto the latent variable space Z to obtain the latent representation $z = E(L_l)$. Using z as an input, we generate the output high-resolution light field $U(z)$ with the convolutional decoder network U. The concatenation of encoder and decoder we term the generator G, which generates high-resolution light fields from low-resolution ones. Its output should be the same as the high-resolution ground truth L_h provided to the network.

6.1.2. DiffGAN discriminator for light fields

On top of the output of the generator, we model the discriminator network D. Overall, we use a deep GAN architecture that was originally proposed by Goodfellow et al. [59] with a few modifications. First, we use epipolar volumes instead of plain images, thus the generator and discriminator networks use three-dimensional convolutions. Second, we propose an additional input to the GAN. Besides the generated patch, we also feed the discriminator with the angular and spatial derivatives. We believe that for the super-resolution task, the high-frequency components are very helpful to obtain good quality and aesthetically pleasing results. In previous work [15], it was shown that GANs help with enhancing details in the high-resolution version of the image, however, it might create some unwanted distortions and hallucinate details that are not present in the original image. Since the light field in theory has more information about the scene due to its sub-aperture views, we want to focus the attention of the discriminator in particular on the sharpness of the reconstructed details.

For any light field L'_h, our discriminator $G(L'_h)$ actually takes as additional input all of the derivatives of L'_h explicitly, i.e.

$$G(L'_h) = G(L'_h,\, \partial_s L'_h,\, \partial_t L'_h, \partial_y L'_h,\, \partial_x L'_h\,). \tag{6.1}$$

We omit this explicit dependency in the following to not clutter notation. The discriminator loss is built such that the target output for generated light fields is one, the target output for real light fields is zero. Thus, for a single low-/high-resolution training pair (L_l, L_h), the discriminator loss becomes

$$E_{\text{gan}}(D) = \log D(L_h) + \log(1 - D(G(L_l)). \tag{6.2}$$

In addition, the generator G is modelling the outputs such that they are similar to

ground truth according to the discriminator D. The generator tries to take the output of the discriminator to zero, and thus minimizes

$$E_{\text{gan}}(G) = -\log(1 - D(G(L_l)). \tag{6.3}$$

Note that in the actual implementation, the GAN losses are split up into a sum of contributions for horizontal and vertical epipolar volumes. Minimization steps for discriminator and generator are performed in an alternating fashion.

6.1.3. Loss functions of our network

As the main loss function of the network, we still use L^2-loss between the network's output and the high-resolution ground truth. We found that GAN training becomes more stable if we introduce a weighted pixel-wise L^2-loss. The weight is chosen according to the brightness of the pixel in the ground truth epipolar volume, and penalizes dark regions more, since usually dark regions are more noisy and the L^2-loss is less sensitive there. To further enhance the detail of the output, we also calculate the L^2-loss between derivatives of the output and the ground truth. Again for a single training light field, the total reconstruction loss from directly comparing the generated to the ground truth high resolution light field reads

$$E_{\text{rec}}(G) = (1 - \exp(-L_h/0.5))\|G(L_l) - L_h\| + \|\nabla G(L_l) - \nabla L_h\|, \tag{6.4}$$

where the gradient is computed spatially.

To improve the quality of the output and speed up the training process, we additionally employ a perceptual loss [58]. For this, we pass the output and the ground truth through the convolutional layers of a pre-trained Inception-v3 network [60] and accumulate the L^2-loss of the features in all the layers to compute $E_{\text{perceptual}}(G)$. Finally, we add the loss of the DiffGAN for both vertical and horizontal stacks to our generator network, as described in the previous subsection. The total loss of the generator network for a single light field becomes

$$E_{\text{total}}(G) = E_{\text{rec}}(G) + E_{\text{perceptual}}(G) + E_{\text{gan}}(G). \tag{6.5}$$

In the next section, we detail the architecture of the sub-networks, which we follow with a detailed strategy we use to minimize the loss and prediction error.

6.2. Detailed network architecture

The light field has a very rigid structure linked to the scene geometry, see Fig. 2.3, which gives rich and redundant information about how to precisely match sub-aperture views, as required for super-resolution. In order to take full advantage of this structure efficiently, we propose a network architecture with a tailored 3D autoencoder, which is shown in Fig. 6.2. The inputs to our network are patch-wise vertical and horizontal epipolar volumes from the low-resolution light field "cross-hair", a cross-shaped subset of views around a reference view. By using the autoencoder structure and the patch-wise input instead of the whole image, we can significantly reduce computational cost. The vertical volume consists of the top and bottom views and the center view which are marked by red frames in the left-hand side of Fig. 6.2, while the horizontal volume contains the left- and rightmost views and the center view that are framed in green. The high-resolution cross-hair encompassing all views is fed as the ground truth.

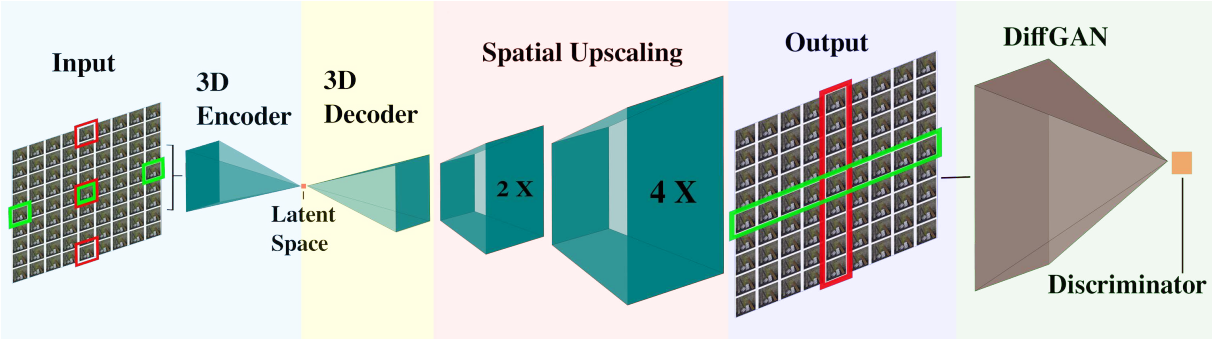


Figure 6.2: *Proposed network architecture for light field spatial-angular super-resolution. The input of the network consists of three views from the vertical and horizontal stacks in the cross-hair that are framed in red and green, respectively. All views are split into* 48×48 *patches with 16 overlapping pixels with each of their neighbours to decrease the dimensionality of the data. The volumes of size* $3 \times 48 \times 48$ *are downscaled spatially to* $3 \times 3 \times 3$ *when they reach the latent space of the 3D autoencoder, then upscaled spatially as well as angularly to* $9 \times 48 \times 48$ *at the end of the decoder. In the upscaling phase, the decoded volumes are again spatially upscaled to twice and four times the size of the input i.e.* $9 \times 96 \times 96$ *and* $9 \times 192 \times 192$ *to obtain our output. Finally, we introduce the DiffGAN to distinguish the output images and the ground truth by their pixel values, as well as the spatial and angular derivatives, thus improve the details in the super-resolved images.*

6.2.1. Architecture of the encoder

The same as in the preliminary experiments, each epipolar volume has a spatial resolution of 48×48. The horizontal volume is spatially transposed such that the images exhibit the same view point motion as the vertical patches. This way, the vertical and the horizontal volumes can share the convolution kernels in the whole network.

The 3D encoder has 9 layers, applying the residual block [55] to the features. The residual block in the encoder is detailed in the left part of Fig. 6.3. This time we only have 3 views in each input epipolar volume, so the even layers downscale the features spatially with stride-2 convolution in the residual block, while the odd layers just gradually increase the number of features. In the latent space, the vertical and horizontal volumes are downscaled from $3 \times 48 \times 48$ finally to $3 \times 3 \times 3$.

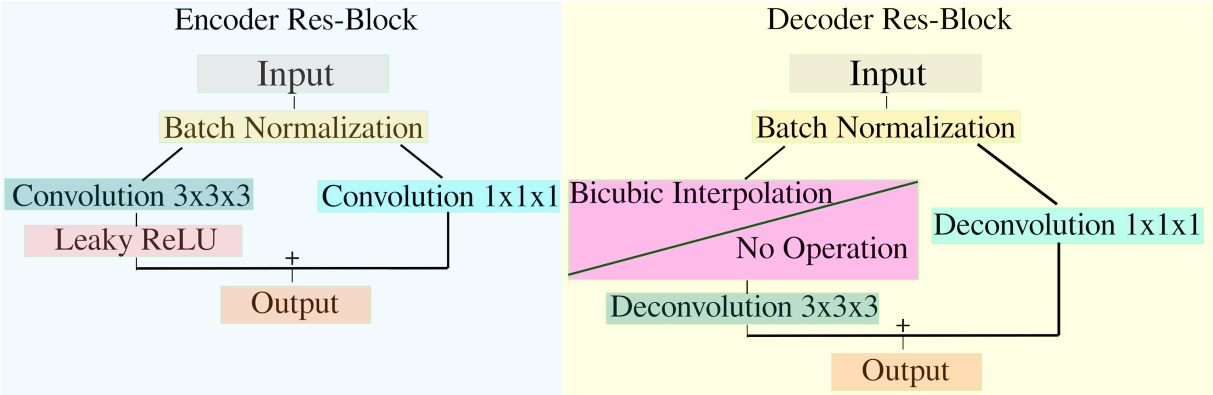


Figure 6.3: *Residual blocks in the encoder and decoder. Left: The residual block of the encoder performs batch normalization, $3 \times 3 \times 3$ convolution and a leaky ReLU with $\alpha = 0.2$. For the residual connection, the input features will be transformed with a $1 \times 1 \times 1$ convolution, if necessary with stride, to achieve the correct output size before addition. Right: The residual block of the decoder also employs batch normalization. If the features are going to grow spatially, bicubic interpolation will upscale them, else the features keep their resolution. $3 \times 3 \times 3$ deconvolution is then applied with stride 1 and "VALID" padding to avoid the fabricated pixels on the border of the image. Likewise, if the input features in this block are rescaled spatially or angularly by the left chain, they are convolved with a $1 \times 1 \times 1$ kernel with respective stride to allow addition.*

6.2.2. Architecture of the decoder

The first part of the 3D decoder has 9 layers as well, and generates a light field which is spatially the same size, but already has increased angular resolution. The features are spatially upscaled in the even layers and angularly upscaled in the odd layers of the decoder. The right side of Fig. 6.3 shows the structure of the decoding residual blocks. To spatially upscale the features, we grow the spatial size of them by the means of bicubic interpolation and apply unstrided transpose convolution to avoid the checkerboard artifacts in the spatial domain caused by strides [4].

For angular upscaling, we follow the procedure illustrated in Fig. 6.4. The spatial resolution of the features remains the same after the batch normalization. We manually add two views in the angular dimension to the features and also add "2" in the output size[1] of the deconvolution. Then we apply the unstrided transpose convolution with the pre-defined output size to the features. The $3 \times 3 \times 3$ transpose convolution with stride 1 and "VALID" padding will produce two more views at the boundaries which are discarded in the output of this residual block. At this point, the features are angularly super-resolved to the target number of nine views both in the vertical and horizontal direction.

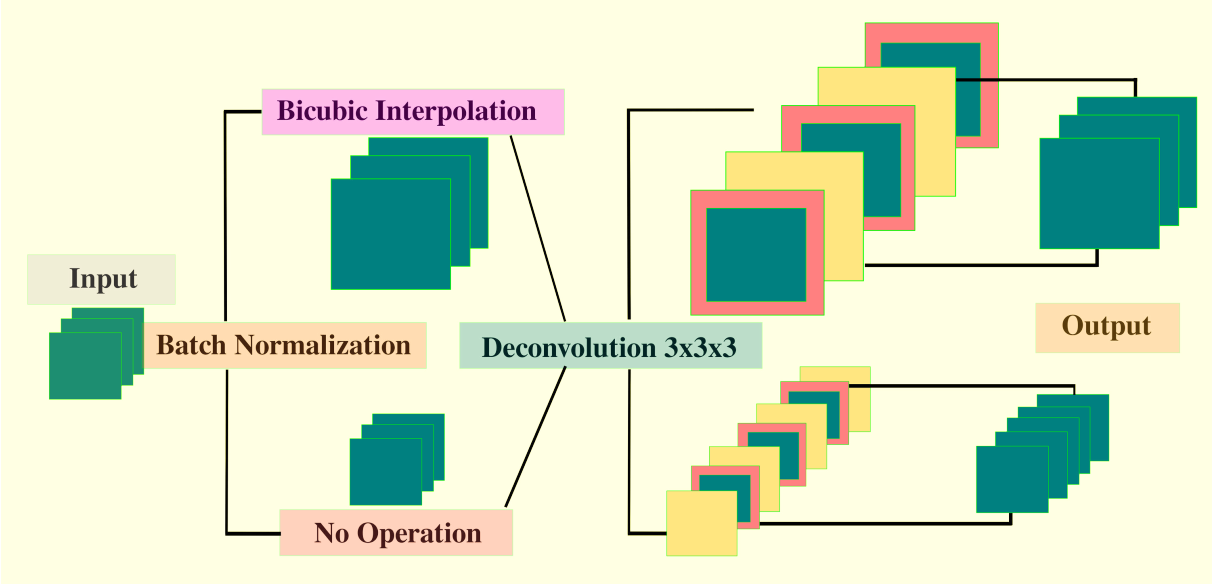


Figure 6.4: *Residual block of a decoder layer in detail. The input features are first batch-normalized. Only if they are to be upscaled spatially, bicubic interpolation will be applied to them. Afterwards, the features pass through a* $3 \times 3 \times 3$ *transpose convolution. We add 2 pixels in the spatial and 2 views in the angular output size and apply stride-1 with "VALID" padding. To obtain the final output of this block, we discard again one pixel from each side of the features and the additional views generated at the boundaries.*

[1] The 3D deconvolution operation in the TensorFlow accepts the output size (**batch, angular dimension, height width, channel**) as an argument. Here we add "2" to the angular dimension so that after applying the deconvolution to the features there will be two more views on the boundaries.

The features of each layer in the encoder have only three vertical or horizontal views, so to prepare the skip connection volume for the decoder, we simply copy the encoded volume in the angular dimension and cut the surplus views to match the angular size of the decoded volume. Besides, we leave twice as many features in the decoded volume as in the skip connection volume, otherwise, the skip connection volume will dominate the decoding process.

6.2.3. Spatial upscaling part of the decoder

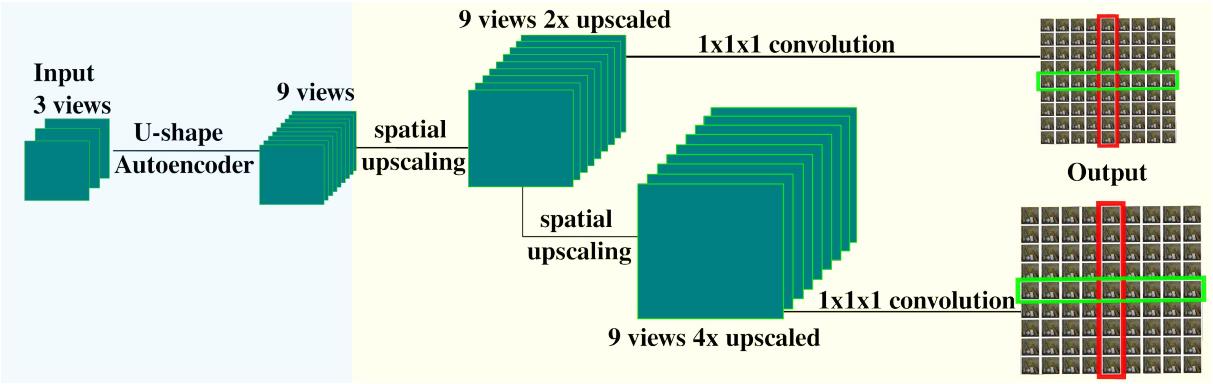


Figure 6.5: *Upscaling network. The features grow angularly from 3 views to 9 views after the U-shape autoencoder and go to the upscaling network. Here the features are 2x spatially upscaled firstly and then to four times.* $1 \times 1 \times 1$ *convolution brings the the number of features to the same as the input.*

On top of the output with increased angular resolution, we stack a spatial upscaling network. This network increases the spatial resolution of the features first to twice the size of the input, then to four times, to obtain the super-resolved output. All the spatial upscaling operations are carried out by a decoder residual block. In the second scaling phase, the features are spatially upscaled by the decoding residual block once and subsequently passed through other 2 decoding residual blocks, without their spatial resolution being changed. Afterwards, $1 \times 1 \times 1$ convolution is applied to the volumes to finally bring the number of features to the same as the input. The upscaling phase for scale factor 4 follows the same procedure, with its input coming from the intermediate results of scale factor 2 just before its $1 \times 1 \times 1$ convolution, see Fig. 6.5.

6.2.4. Architecture of the GAN discriminator

In order to enhance the details of the output, we add the DiffGAN discriminator on top of our network. The discriminator input consists of the stack of concatenated light field together with its derivatives, as previously detailed in section 6.1.2. The structure is similar to the encoder, but we combine two subsequent downsampling and feature expansion layers into a single layer (the 6th layer marked in purple in Fig. 6.6), making the network more shallow.

In fact, the discriminator has a much easier task than the encoder, so it is reasonable to reduce capacity substantially.

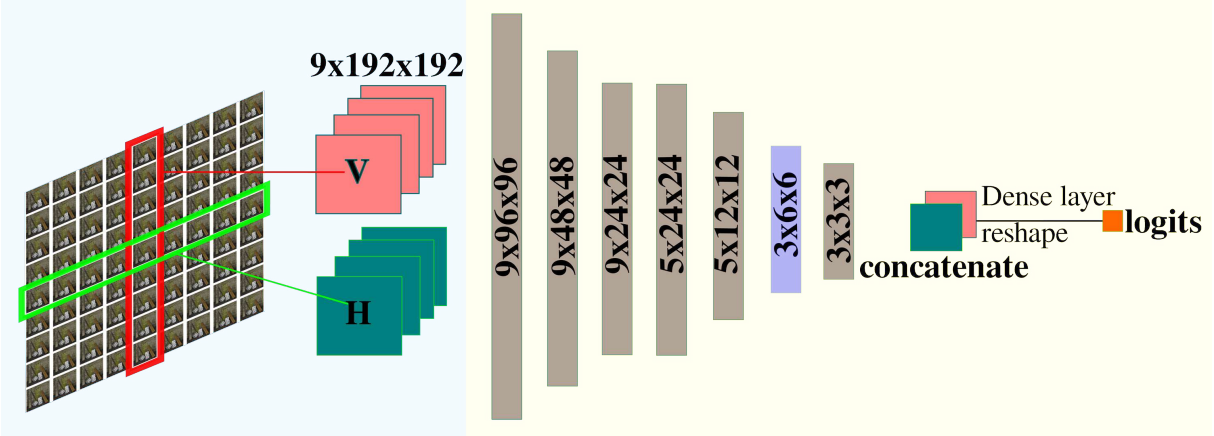


Figure 6.6: *DiffGAN in detail with 4x super-resolution as example. The vertical and horizontal volumes of size* $9 \times 192 \times 192$*, which consist of the high-resolution images and their derivatives, are sent to the convolutional layers of the DiffGAN. The first three layers reduce the size of the features in the spatial domain, followed by a angular downscaling from 9 views to 5 views in the 4th layer. The spatial resolution is decreased again in the 5th layer, while the 6th, which is marked in purple, shrinks the size of the features both spatially and angularly. The last layer brings the resolution finally to* $3 \times 3 \times 3$*. The output vertical and horizontal volumes are concatenated and reshaped in the dense layer to form the logits of size 1.*

The discriminator input horizontal epipolar volume is spatially transposed as before and goes together with the vertical volumes in parallel to the DiffGAN network. The spatial and angular resolution of them are reduced to $3 \times 3 \times 3$ by the 7 convolutional layers there. At this point, the vertical and horizontal outputs are concatenated to a single volume, which is then reshaped in the dense layer after the convolutional layers to form the logits for the discriminator. The procedure is detailed in Fig. 6.6.

The discriminator distinguishes the super-resolved output from the ground truth using a binary linear classifier as a final layer on top of the features of the modified encoder chain.

6.3. Experiments

6.3.1. Training configurations

Choice of color space It is shown in the preliminary experiments that the network performs better in the *YCbCr* color space. Furthermore, we can reduce the memory requirements drastically, by following Timofte et al. [8] and Yoon et al. [64] in their observation that the *YCbCr* color space is proven to lead to the best results, even if only the *Y* channel is being trained while the color components are only interpolated for super-resolution. In our approach we chose therefore the *YCbCr* color space, where the luminance component *Y* contains most of the spatial detail of the image. The channels *Cb* and *Cr* encode the blue-difference and red-difference chroma components, which are typically of low frequency. Training with only one channel in the images reduces the dimensionality of the data, and makes both the input and ground truth datasets substantially lighter.

Training process We perform training on an Intel Core i9 with 128 GB of RAM and 4 nVidia TITAN Xp GPUs. The optimization is performed with the Adam optimizer, with initial learning rate set to $1e-4$. Unfortunately, we are still limited to a small batch size of six. The GAN turned out to be easy to train and will easily dominate the training [69] to produce artifacts, so we assign a small weight of $1e-4$ to the DiffGAN loss to keep it balanced with respect to the other loss components. The loss becomes stable after approximately 10 hours, after which we dropped the learning rate to $1e-5$ and trained for another 8 hours. We then stopped training as the loss did not significantly decrease any more.

6.3.2. Results

Since our network is trained with patch-wise data, the output patches are reassembled to the complete high-resolution *Y* channel of the image. We compute PSNR and SSIM after the images are converted back to *RGB* color space. We evaluate our network both on publicly available light field datasets which are synthetically rendered or captured with a gantry [19, 66, 67], as well as our own data captured with Lytro Illum plenoptic camera. None of the datasets we use in evaluation has been seen during training.

Our approach super-resolves the light fields angularly from 3 views to 9 views and spatially with scale factor 2 and scale factor 4. For comparison, we picked the recent works for single image SRGAN [15] with scale factor 4, the variational approach VarSR [62] which is improved upon [2], and the graph-based method GB-SQ [7] with spatial scale factor 2. Complete numerical results can be observed in Tab. 6.1 and Tab. 6.2.

Table 6.1: *Comparison of PSNR (SSIM) for different methods, for super-resolution scale factor 2 and over a wide range of datasets from various sources.*

Lightfield	Bilinear	GB-SQ [7]	VarSR [62]	**Proposed**
sideboard	25.90 (0.83)	**29.61 (0.92)**	26.90 (0.87)	24.35 (0.73)
antinous	28.43 (0.95)	27.83 (0.96)	27.29 (0.94)	**33.51 (0.94)**
bicycle	27.36 (0.83)	**30.73 (0.90)**	28.58 (0.86)	26.55 (0.78)
tomb	29.59 (0.91)	31.81 (0.94)	30.65 (0.91)	**32.03 (0.85)**
bedroom	27.21 (0.86)	26.66 (0.91)	26.73 (0.87)	**30.59 (0.83)**
herbs	30.61 (0.83)	**33.61 (0.90)**	31.45 (0.86)	28.68 (0.76)
cotton	27.33 (0.95)	27.87 (0.96)	27.45 (0.95)	**39.54 (0.96)**
platonic	33.39 (0.90)	**38.42 (0.96)**	34.53 (0.92)	31.76 (0.82)
rosemary	32.24 (0.94)	**37.26 (0.98)**	33.75 (0.96)	30.03 (0.91)
origami	29.66 (0.91)	**35.72 (0.96)**	31.29 (0.93)	29.28 (0.89)
maria	30.05 (0.86)	**33.92 (0.92)**	32.78 (0.91)	31.75 (0.86)
koala	27.09 (0.94)	**32.80 (0.97)**	28.52 (0.95)	22.99 (0.88)
owl2	36.21 (0.97)	**41.04 (0.98)**	37.93 (0.97)	33.69 (0.92)
flowers	34.23 (0.96)	**36.98 (0.98)**	36.03 (0.97)	31.77 (0.93)
owl-str	32.14 (0.94)	**36.46 (0.97)**	32.86 (0.95)	29.77 (0.86)
origami	28.90 (0.93)	**32.03 (0.95)**	29.86 (0.94)	28.56 (0.92)
hedgehog	34.41 (0.95)	**39.07 (0.98)**	34.98 (0.95)	30.93 (0.90)
eucalyptus	33.98 (0.96)	**37.68 (0.98)**	34.93 (0.95)	29.77 (0.88)
truck	36.06 (0.96)	**39.80 (0.98)**	36.94 (0.96)	31.84 (0.90)

Table 6.2: *Comparison of PSNR (SSIM) for different methods, for super-resolution scale factor 4 and over a wide range of datasets from various sources.*

Lightfield	Bilinear	SRGAN [15]	VarSR [62]	**Proposed**
sideboard	21.93 (0.57)	20.58 (0.57)	**22.43 (0.64)**	20.82 (0.54)
antinous	28.03 (0.92)	**33.81 (0.91)**	26.86 (0.88)	33.45 (0.92)
bicycle	23.68 (0.63)	23.82 (0.63)	**24.06 (0.67)**	22.49 (0.60)
tomb	29.22 (0.83)	30.62 (0.67)	28.45 (0.75)	**30.96 (0.78)**
bedroom	26.02 (0.74)	**28.47 (0.72)**	24.76 (0.68)	27.69 (0.71)
herbs	27.28 (0.69)	26.80 (0.67)	**27.41 (0.72)**	25.90 (0.64)
cotton	27.34 (0.92)	34.83 (0.90)	26.91 (0.89)	**36.69 (0.93)**
platonic	29.07 (0.72)	27.38 (0.63)	**29.63 (0.77)**	28.15 (0.68)
rosemary	27.59 (0.84)	27.58 (0.85)	**27.64 (0.86)**	26.31 (0.80)
origami	25.38 (0.80)	**26.75 (0.82)**	26.27 (0.83)	24.29 (0.78)
maria	26.45 (0.69)	26.08 (0.65)	**27.55 (0.73)**	25.87 (0.62)
koala	20.93 (0.77)	**22.00 (0.81)**	21.20 (0.79)	19.05 (0.69)
owl2	29.74 (0.89)	29.26 (0.87)	**30.68 (0.90)**	27.62 (0.82)
flowers	28.64 (0.87)	29.51 (0.85)	**29.54 (0.89)**	27.41 (0.83)
owl-str	26.48 (0.79)	25.35 (0.75)	**26.66 (0.81)**	23.67 (0.70)
origami	24.09 (0.78)	22.27 (0.76)	**24.18 (0.76)**	22.14 (0.72)
hedgehog	**28.31 (0.81)**	26.24 (0.76)	28.10 (0.82)	25.85 (0.72)
eucalyptus	30.35 (0.90)	**30.48 (0.84)**	29.87 (0.86)	28.17 (0.84)
truck	**31.52 (0.90)**	31.34 (0.87)	27.99 (0.75)	29.53 (0.86)

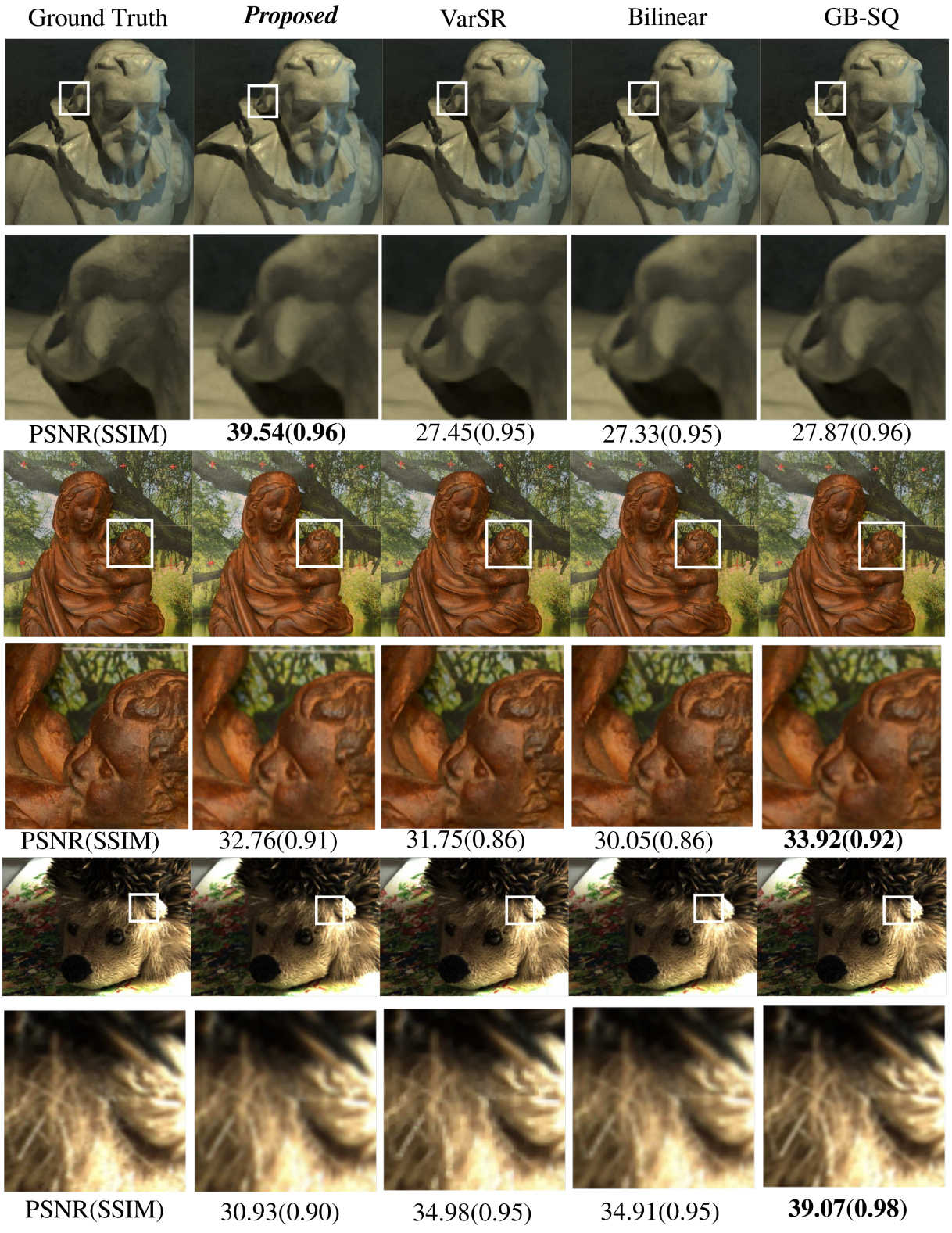


Figure 6.7: *Visualization and quantitative evaluation of the results of scale factor 2. The images are super-resolved center views of Benchmark Cotton, HCI Maria and real-world data Hedgehog.*

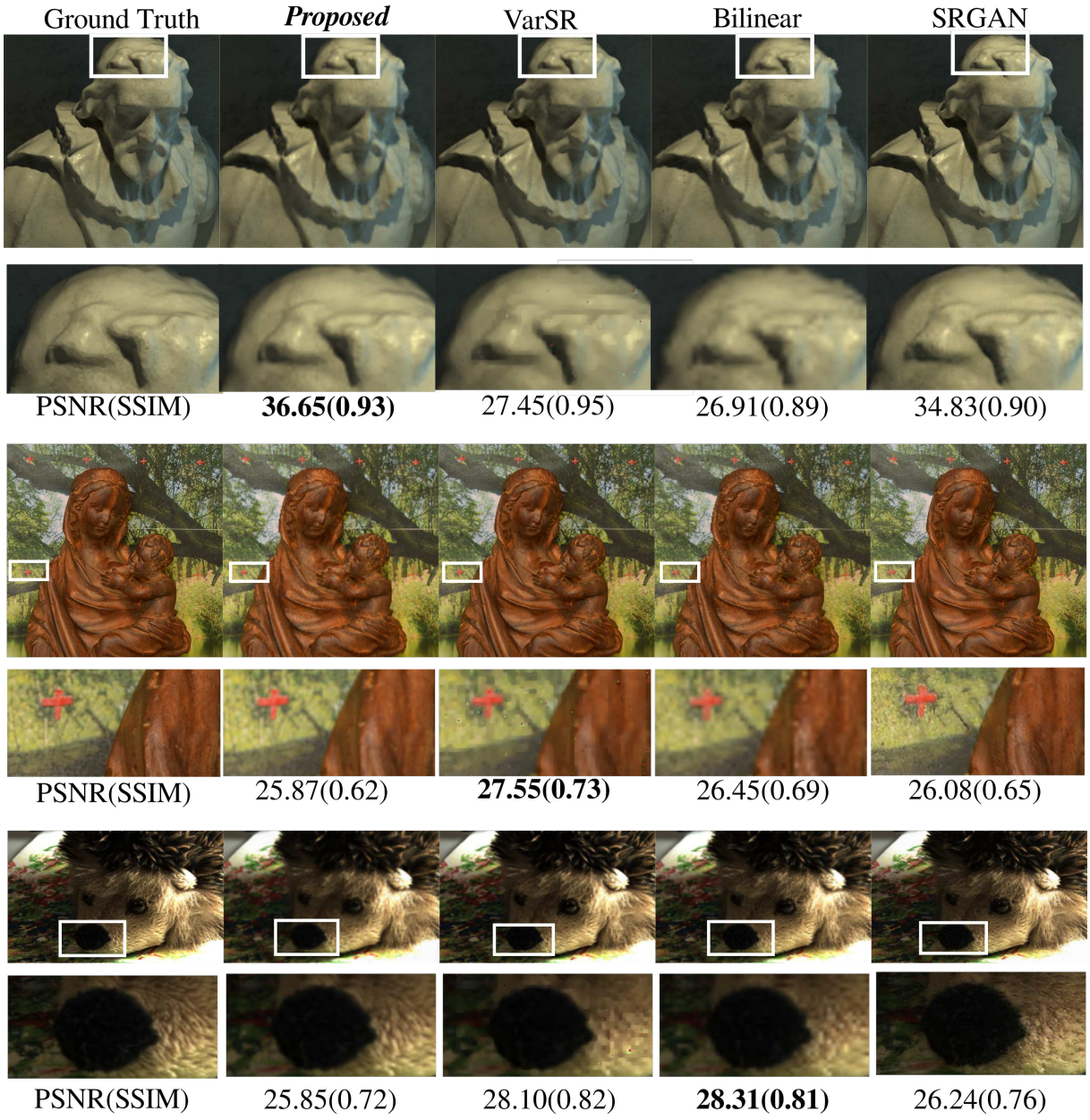


Figure 6.8: *Visualization and quantitative evaluation of the results of scale factor 4. The images are super-resolved center views of Benchmark Cotton, HCI Maria and real-world data Hedgehog.*

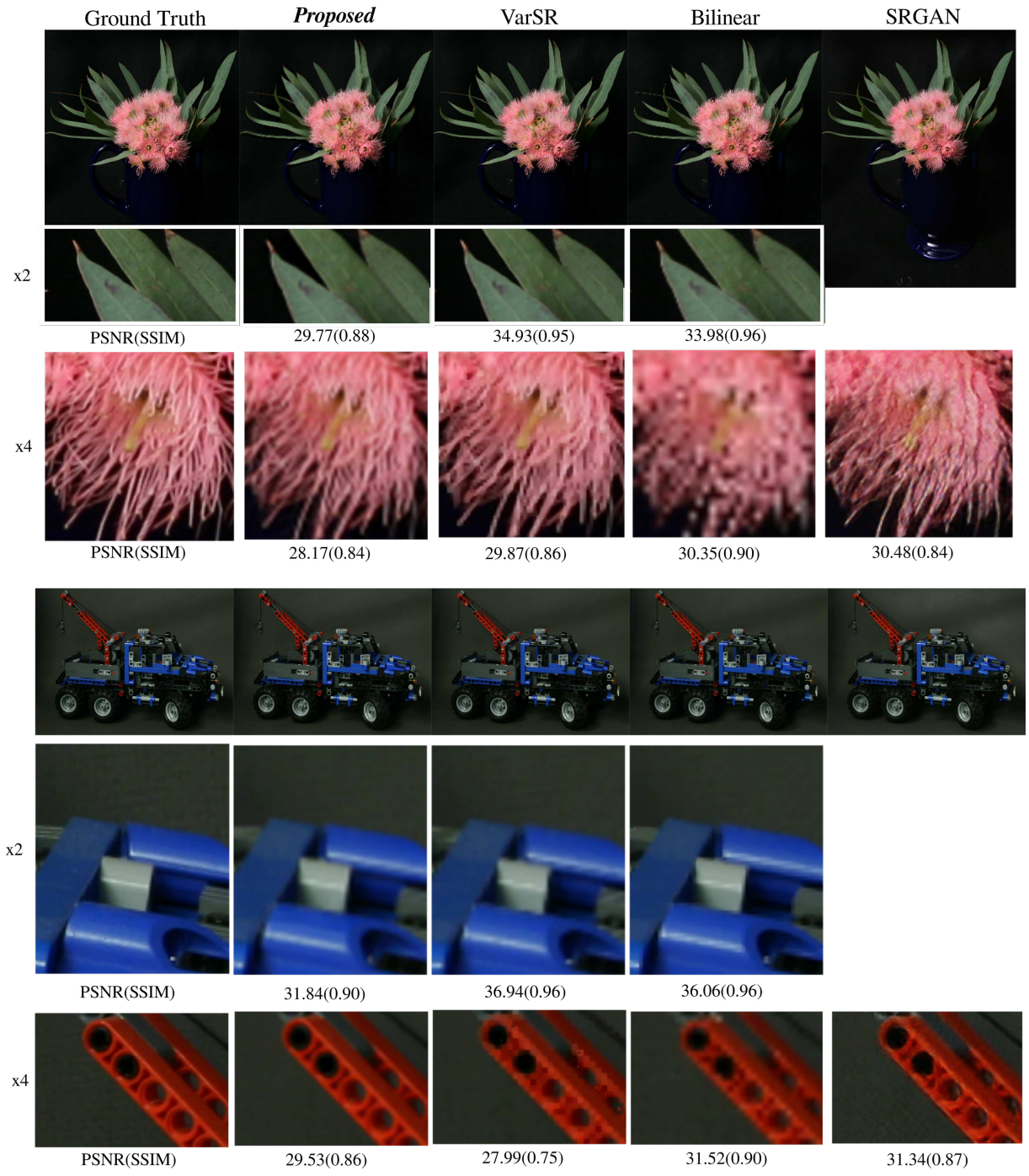


Figure 6.9: *Results on the Stanford datasets [19]. Our approach, as well as the VarSR [62] and Bilinear interpolation provide the 2x and 4x super-resolved light fields. The zoomed-in areas are shown with corresponding PSNR (SSIM) below the images, respectively. The SRGAN [15] only upscales the images to four times of their original size.*

In Fig. 6.7 and Fig. 6.8, we visualize the super-resolved center views of scale factor 2 and scale factor 4 of the Benchmark Cotton, HCI Maria and real-world data Hedgehog. The PSNR and SSIM of each approach are reported below the images. One can observe over-smoothing of VarSR [62] in the scale factor 2 results, while some severe high-frequency noise appears in their scale factor 4 results. Bilinear interpolation provides rapid single image super-resolution with acceptable PSNR and SSIM, but for scale factor 4 it is significantly more blurry than the other methods.

The graph-based approach GB-SQ [7] gives brilliant results in HCI Maria and real-world data Hedgehog and is visually also very sharp in Benchmark Cotton. As we can see in Tab. 6.1 and Tab. 6.2, its results also take the leading positions in most of the light fields in the test set. However, its computational time means it is next to inapplicable in practice compared to two to three minutes on average using our approach. In our experiments, it took around 8 hours to compute a 9×9 light field for scale factor 2 with a default parameter setting. We reduce the number of the optimization steps of GB-SQ [7] that significantly saves the computational time for the light fields with smaller resolution, like the Lytro dataset and Benchmark [66]. As the result, the output images experience an obvious degradation in terms of PSNR. For the light fields with large resolution, for example the HCI [67] and Stanford [19] ones, the reduction of the optimization step does not make a difference in the computational time.

SRGAN seems to sometimes hallucinate additional structure, for example in the zoomed-in area of Hedgehog (Fig. 6.8), the fur around the nose is sharp but apparently does not have the same shape as the ground truth.

Fig. 6.9 shows the results on the Stanford datasets. The bilinear interpolation provides more blurry results than the other three approaches. The VarSR [62] performs well on the Eucalyptus light field, but on the Truck, its scale factor 4 result has an obvious jagged distortion. SRGAN again produces excessive content in the results, for example the flower part is lack of fidelity, the zoomed-in area of the Truck also contains additional pattern in the background.

Visually, we perform very well in particular on Truck, although PSNR does not reflect this. Compared to SRGAN, our method does not hallucinate any new details. Compared to the other approaches, our results do not contain any artifacts such as salt and pepper noise like in the variational approach [62].

From Fig. 5.2 we can see that even we try to preserve the best quality in the low-resolution version images, they lose a lot of information during downscaling. The four-times downscaled image is much worse than that is 2x downscaled. Furthermore, the light fields from Benchmarks [66] and those captured with the Lytro Illum plenoptic camera have a very small baseline, the disparity range is quite small, usually within $[-2, 2]$. Downscaling such a light field four times might result in discarding the epipolar structure. Thus, all sub-aperture views will look the same with only minuscule view point change. One of the particular use cases of light field super resolution is to upscale noisy examples captured with the Lytro Illumm.

Here we give a showcase on the Antinous light field from Benchmark [66] on the following 3 pages. The first page (Fig. 6.10) and the second page (Fig. 6.11) contain ground truth qualitative evaluations for scale factors 2 and 4. To obtain the scale factor 2, we downscale the original light field twice and then run our network to obtain the result

with magnification factor x2. We then apply a similar procedure to compute light fields with the magnification factor x4. On page three (Fig. 6.12), we show the magnification factor x4 of the light field that was downscaled only two times. Thus, we do not have a ground truth comparisons in this showcase, but we present the real application of our method.

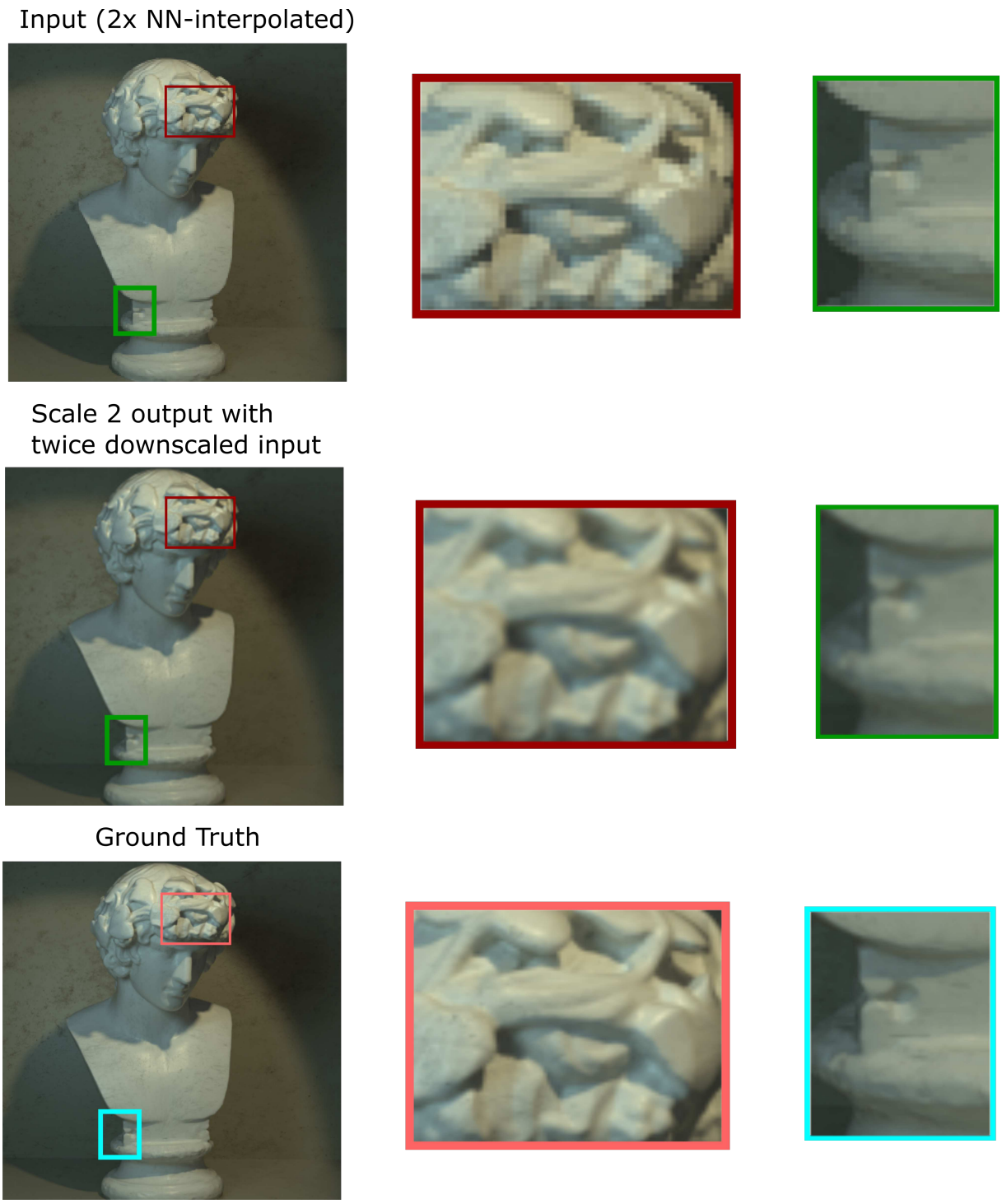


Figure 6.10: *Visualization of the Antinous light field from Benchmark [66]. The top row shows the 2x upscaled input by nearest neighbor interpolation and its zoomed-in areas. The next two rows shows our output and the ground truth with their zoomed-in areas, respectively.*

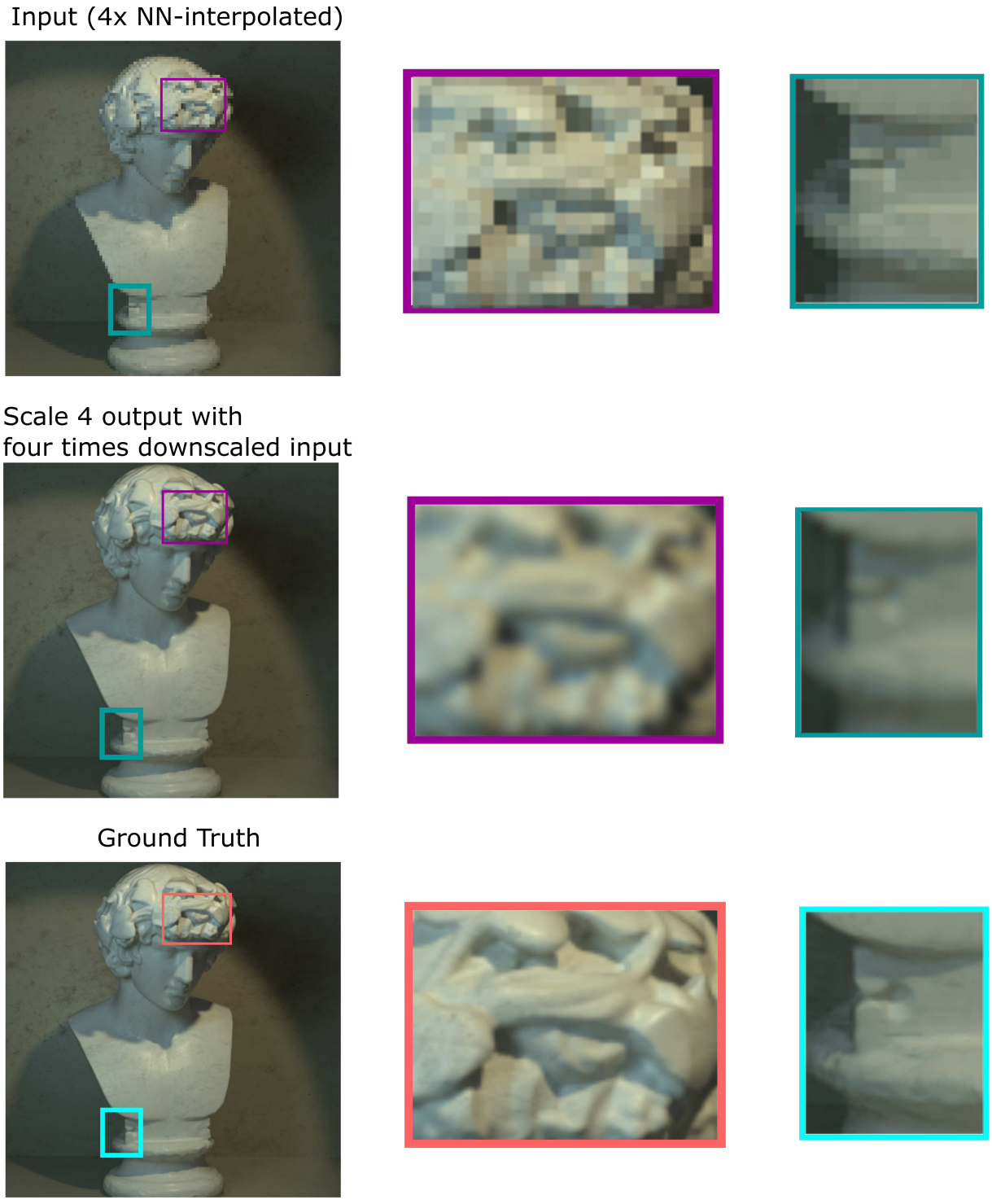


Figure 6.11: *Visualization of the Antinous light field from Benchmark [66]. The top row shows the 4x upscaled input by nearest neighbor interpolation and its zoomed-in areas. The next two rows shows our output and the ground truth with their zoomed-in areas, respectively.*

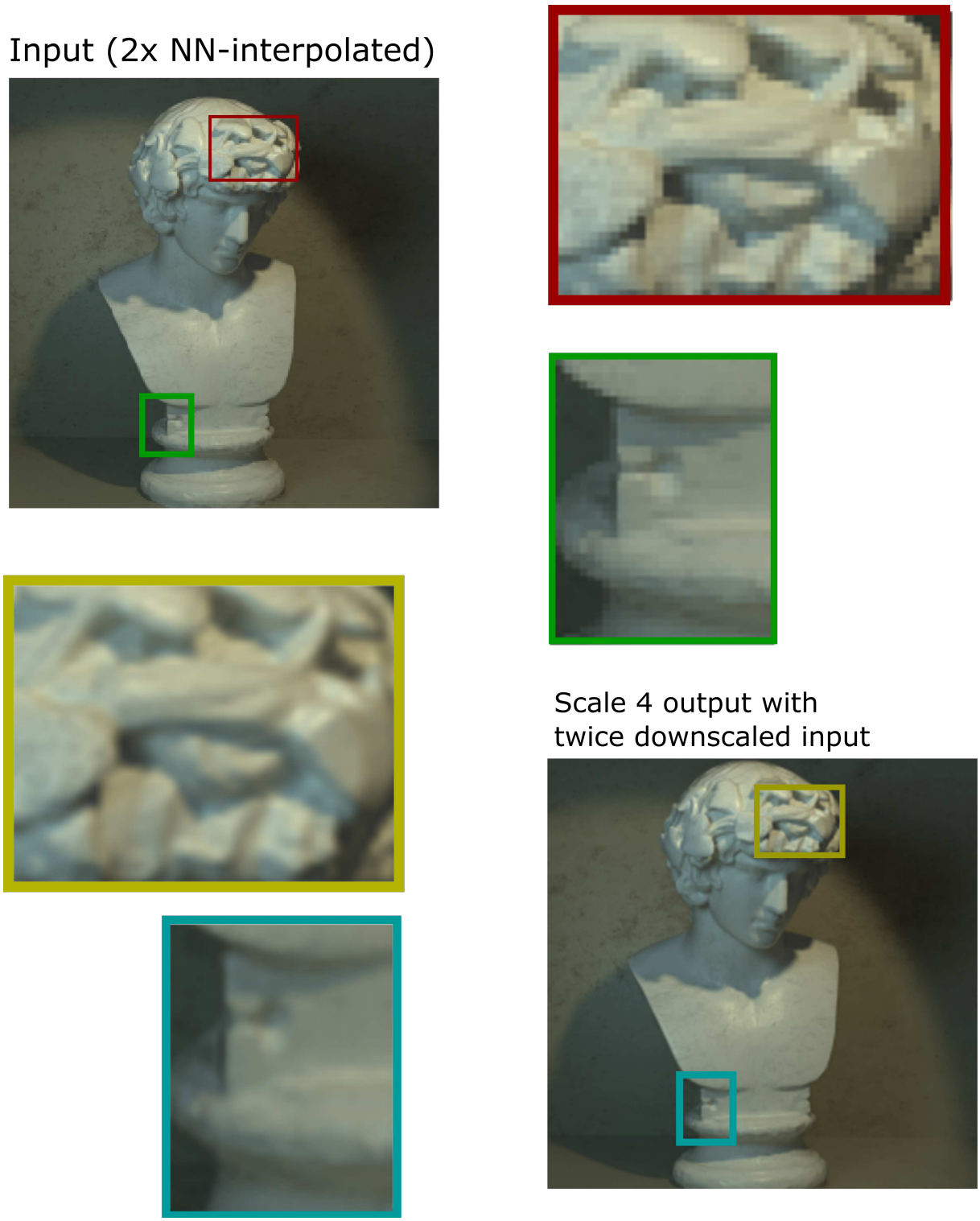


Figure 6.12: *Visualization of the Antinous light field from Benchmark [66]. The top row shows the 2x upscaled input by nearest neighbour interpolation and its zoomed-in areas. The second row illustrates the 4x output from a scale factor 2 input.*

6.3.3. Ablation study

To support our choice of the network architecture, we also perform an ablation study. We train the same network as in the Sec. 6.2 but without the DiffGAN loss. Also, we present numerical evaluations from network that is the same as in Sec. 6.3.2, but with five input views in the epipolar volumes.

Table 6.3: *Comparison of PSNR (SSIM) for different network configurations, for super-resolution scale factor 2 over a wide range of datasets from various sources*

	No GAN v.s. GAN		5 views v.s. 3 views	
Light field	3 views no GAN	**3 views GAN**	5 views GAN	**3 views GAN**
sideboard	24.25 (0.74)	**24.35 (0.73)**	**25.76 (0.82)**	24.35 (0.73)
antinous	**34.72 (0.94)**	33.51 (0.94)	**36.54 (0.96)**	33.51 (0.94)
bicycle	26.39 (0.79)	**26.55 (0.78)**	**27.14 (0.83)**	26.55 (0.78)
tomb	**36.14 (0.86)**	32.03 (0.85)	**36.41 (0.89)**	32.03 (0.85)
bedroom	**30.67 (0.84)**	30.59 (0.83)	**31.07 (0.86)**	30.59 (0.83)
herbs	27.83 (0.75)	**28.68 (0.76)**	**29.40 (0.80)**	28.68 (0.76)
cotton	37.97 (0.96)	**39.54 (0.96)**	37.20 (0.97)	**39.54 (0.96)**
platonic	31.40 (0.80)	**31.76 (0.82)**	**33.05 (0.88)**	31.76 (0.82)
rosemary	**30.63 (0.92)**	30.03 (0.91)	**31.85 (0.94)**	30.03 (0.91)
origami	28.33 (0.88)	**29.28 (0.89)**	**29.63 (0.91)**	29.28 (0.89)
maria	**31.86 (0.88)**	31.75 (0.86)	**32.64 (0.90)**	31.75 (0.86)
koala	22.75 (0.87)	**22.99 (0.88)**	22.81 (0.88)	**22.99 (0.88)**
owl2	32.11 (0.87)	**33.69 (0.92)**	**33.76 (0.89)**	33.69 (0.92)
flowers	28.59 (0.78)	**31.77 (0.93)**	31.05 (0.85)	**31.77 (0.93)**
owl-str	**32.86 (0.95)**	29.77 (0.86)	**30.90 (0.91)**	29.77 (0.86)
origami	**29.56 (0.85)**	28.56 (0.92)	**28.84 (0.92)**	28.56 (0.92)
hedgehog	30.00 (0.87)	**30.93 (0.90)**	**31.15 (0.90)**	30.93 (0.90)
eucalyptus	25.93 (0.62)	**29.77 (0.88)**	**30.53 (0.79)**	29.77 (0.88)
truck	30.21 (0.86)	**31.84 (0.90)**	**33.42 (0.91)**	31.84 (0.90)

Our study shows that the proposed architecture benefits from the adversarial loss. With our DiffGAN, the network accepting 3 views outperforms that without DiffGAN in 12 out of 19 light fields. An increase in the number of input views from 3 to 5 also improves the quality of the output light fields that the network with 3 views as input performs better only on 3 light fields than that with 5 views.

Chapter 7

Conclusions

In this thesis, we present an efficient approach to spatial and angular light field super-resolution based on an encoder-decoder architecture with a novel GAN loss.

The basics of the light fields are presented in the first part of the thesis, showing that a 4D light field has much richer information than single images and image sequences. With this advantage, people are capable of estimating the depth within the scene and accomplishing super-resolution tasks. Next, we gave a detailed introduction of deep learning to help better understand the working principle of convolutional neural networks, which is nowadays the most popular way to deal with computer vision tasks. In the related works we reviewed the state-of-the-art deep learning methods that successfully attacked those tasks, such as the residual network [55] for image classification, encoder-decoder network with skip connection [9] and the SRGAN [15] to super-resolve single images, to name but a few. Among these previous works, [10, 12] of Alperovich et al. gave us the biggest inspiration how to process light fields in deep learning.

With our preliminary experiment, we super-resolve the center view in the light field of satisfying quality. In the meanwhile, the influence of the colorspace on deep learning is studied. As a conclusion, we use the *YCbCr* instead of *RGB* images for the training and leads to an excellent result. Furthermore, the checkerboard artifacts by utilizing deconvolution with strides for spatial upscaling are confirmed in the visual results, which should be avoided in a CNN-based super-resolution task. All of this experience has contributed to the good performance of the proposed model, whose architecture is presented in detail in section 6.2.

Our proposed method adopts insights both from 2D [8, 15] and 3D [3, 64] CNN-based approaches to arrive at architecture with very competitive performance. We demonstrate this in numerous experiments on public datasets [19, 67], as well as on real-world light fields captured with a Lytro Illum plenoptic camera. Our architecture is simple and powerful in the sense that it fully utilizes the rich information inherited from the light field and proves that even very few sub-aperture views are sufficient to reconstruct a high-resolution dense light field of good quality. Competing state-of-the-art light field super-resolution algorithms that require many more input views and/or take a lot more time to compute, our network is applicable to various disparity ranges and can be used when only a few views are available. In addition, we propose the new DiffGAN loss that improves the visual quality of the results, which is also proven to be powerful for the addressed problem by the ablation study(section 6.3.3). From the experiments, we show that our method does not hallucinate missing information compared to the original SRGAN network.

Bibliography

[1] S. Baker and T. Kanade. Limits on Super-Resolution and How to Break Them. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(9):1167–1183, 2002.

[2] S. Wanner and B. Goldluecke. Variational light field analysis for disparity estimation and super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):606–619, 2014.

[3] Reuben A. Farrugia and Christine Guillemot. A simple framework to leverage state-of-the-art single-image super-resolution methods to restore light fields. *CoRR*, abs/1809.10449, 2018.

[4] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proc. CVPR*, pages 1874–1883, 2016.

[5] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar. Fast and Robust Multiframe Super Resolution. *IEEE Transactions on Image Processing*, 13(10):1327–1344, 2004.

[6] Lixin Shi, Haitham Hassanieh, Abe Davis, Dina Katabi, and Fredo Durand. Light field reconstruction using sparsity in the continuous fourier domain. *ACM Transactions on Graphics*, 34(1):12:1–12:13, 2014.

[7] Mattia Rossi and Pascal Frossard. Geometry-consistent light field super-resolution via graph-based regularization. *IEEE Transactions on Image Processing*, 27:4207–4218, 2018.

[8] Radu Timofte, Rasmus Rothe, and Luc Van Gool. Seven ways to improve example-based single image super resolution. In *Proc. CVPR*, pages 1865–1873, 2016.

[9] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Proc. NIPS*, 2016.

[10] A. Alperovich, O. Johannsen, M. Strecke, and B. Goldluecke. Light field intrinsics with a deep encoder-decoder network. In *Proc. CVPR*, 2018.

[11] Pratul P. Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. Learning to synthesize a 4D RGBD light field from a single image. In *Proc. ICCV*, pages 2262–2270, 2017.

[12] A. Alperovich, O. Johannsen, and B. Goldluecke. Intrinsic light field decomposition and disparity estimation with a deep encoder-decoder network. In *IEEE European Signal Processing Conference (EUSIPCO)*, 2018.

[13] Y. Yoon, H. Jeon, D. Yoo, J. Lee, and I. S. Kweon. Learning a deep convolutional network for light-field image super-resolution. In *ICCV Workshops*, pages 57–65, Dec 2015.

[14] M. Zhu, A. Alperovich, O. Johannsen, and B. Goldluecke. Conference submission to cvpr 2019. In *Proc. CVPR*, 2019.

[15] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *Proc. CVPR*, pages 105–114, 2017.

[16] Michael Faraday. Thoughts on ray-vibrations. *Philosophical Magazine Series 3*, 28(188), pp. 345–350, 1846.

[17] A. Gershun. The light field. *Journal of Mathematics and Physics*, 18(1-4):51–151, 1939.

[18] E.H. Adelson and J.R. Bergen. The plenoptic function and the elements of early vision. *Computational models of visual processing*, 1, 1991.

[19] V. Vaish and A. Adams. The (New) Stanford Light Field Archive. `http://lightfield.stanford.edu`, 2008.

[20] S. Kabanikhin. Definitions and examples of inverse and ill-posed problems. *JIIP*, 16(4), pp. 317-357., 2008.

[21] Thomas Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[24] Fei-Fei Li, Justin Johnson, and Serena Yeung. Slides from machine learning lecture.

[25] Prakhar Mishra. Understanding autoencoders - unsupervised learning technique, 2018.

[26] J. Clark, M. Palmer, and P. Lawrence. A transformation method for the reconstruction of functions from nonuniformly spaced samples. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(5):1151–1165, October 1985.

[27] S. P. Kim and N. K. Bose. Reconstruction of 2-d bandlimited discrete signals from nonuniform samples. *IEE Proceedings F - Radar and Signal Processing*, 137(3):197–204, June 1990.

[28] Yang-Ming Zhu. Generalized sampling theorem. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(8):587–588, Aug 1992.

[29] J. Brown. Multi-channel sampling of low-pass signals. *IEEE Transactions on Circuits and Systems*, 28(2):101–106, February 1981.

[30] Hanoch Ur and Daniel Gross. Improved resolution from subpixel shifted pictures. *CVGIP: Graphical Models and Image Processing*, 54(2):181 – 186, 1992.

[31] L. Landweber. An Iteration Formula for Fredholm Integral Equations of the First Kind. *American Journal of Mathematics*, 73:615–624, 1951.

[32] T. Komatsu, K. Aizawa, T. Igarashi, and T. Saito. Signal-processing based method for acquiring very high resolution images with multiple cameras and its theoretical analysis. *IEE Proceedings I - Communications, Speech and Vision*, 140(1):19–24, Feb 1993.

[33] M. S. Alam, J. G. Bognar, R. C. Hardie, and B. J. Yasuda. Infrared image registration and high-resolution reconstruction using multiple translationally shifted aliased video frames. *IEEE Transactions on Instrumentation and Measurement*, 49(5):915–923, Oct 2000.

[34] S. P. Kim, N. K. Bose, and H. M. Valenzuela. Recursive reconstruction of high resolution image from noisy undersampled multiframes. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(6):1013–1027, June 1990.

[35] S. P. Kim and W. . Su. Recursive high-resolution reconstruction of blurred multiframe images. In *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, pages 2977–2980 vol.4, April 1991.

[36] N. K. Bose, H. C. Kim, and H. M. Valenzuela. Recursive implementation of total least squares algorithm for image reconstruction from noisy, undersampled multiframes. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 269–272 vol.5, April 1993.

[37] Moongi Kang Seunghyeon Rhee. Discrete cosine transform based regularized high-resolution image reconstruction algorithm. *Optical Engineering*, 38:38 – 38 – 9, 1999.

[38] M. Elad and A. Feuer. Restoration of a single superresolution image from several blurred, noisy, and undersampled measured images. *IEEE Transactions on Image Processing*, 6(12):1646–1658, Dec 1997.

[39] Nhat Nguyen, P. Milanfar, and G. Golub. A computationally efficient superresolution image reconstruction algorithm. *IEEE Transactions on Image Processing*, 10(4):573–583, April 2001.

[40] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259 – 268, 1992.

[41] T. F. Chan, S. Osher, and J. Shen. The digital tv filter and nonlinear denoising. *IEEE Transactions on Image Processing*, 10(2):231–241, Feb 2001.

[42] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar. Fast and robust multiframe super resolution. *IEEE Transactions on Image Processing*, 13(10):1327–1344, Oct 2004.

[43] Longguang Wang, Zaiping Lin, Xinpu Deng, and Wei An. Fast multi-frame image super-resolution based on MRF. *CoRR*, abs/1706.06266, 2017.

[44] Xuelong Li, Yanting Hu, Xinbo Gao, Dacheng Tao, and Beijia Ning. A multi-frame image super-resolution method. *Signal Process.*, 90(2):405–414, February 2010.

[45] L. Zhang, H. Zhang, S. Zhang, and Y. Xue. Multi-frame image super-resolution reconstruction based on gpof registration and l1-norm. In *2010 Sixth International Conference on Natural Computation*, volume 7, pages 3601–3604, Aug 2010.

[46] H. Shen, L. Zhang, B. Huang, and P. Li. A map approach for joint motion estimation, segmentation, and super resolution. *IEEE Transactions on Image Processing*, 16(2):479–490, Feb 2007.

[47] S. Farsiu, M. Elad, and P. Milanfar. Multiframe demosaicing and super-resolution of color images. *IEEE Transactions on Image Processing*, 15(1):141–159, Jan 2006.

[48] G. K. Chantas, N. P. Galatsanos, and N. A. Woods. Super-resolution based on fast registration and maximum a posteriori reconstruction. *IEEE Transactions on Image Processing*, 16(7):1821–1830, July 2007.

[49] L Xu, Jimmy Ren, C Liu, and J Jia. Deep convolutional neural network for image deconvolution. In *Proc. NIPS*, pages 1790–1798, 2014.

[50] Jian Sun, Wenfei Cao, Zongben Xu, and Jean Ponce. Learning a convolutional neural network for non-uniform motion blur removal. *CoRR*, abs/1503.00593, 2015.

[51] C. J. Schuler, H. C. Burger, S. Harmeling, and B. Schölkopf. A machine learning approach for non-blind image deconvolution. In *Proc. CVPR*, pages 1067–1074, 2013.

[52] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. *CoRR*, abs/1803.09454, 2018.

[53] Wei Han, Shiyu Chang, Ding Liu, Mo Yu, Michael J. Witbrock, and Thomas S. Huang. Image super-resolution via dual-state recurrent networks. *CoRR*, abs/1805.02704, 2018.

[54] Muhammad Haris, Greg Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. *CoRR*, abs/1803.02735, 2018.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[56] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[57] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. 2015.

[58] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[59] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. NIPS*, pages 2672–2680, 2014.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. CVPR*, pages 2818–2826, 2016.

[61] T. Bishop and P. Favaro. The Light Field Camera: Extended Depth of Field, Aliasing, and Superresolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):972–986, 2012.

[62] S. Pujades, B. Goldluecke, and F. Devernay. Bayesian view synthesis and image-based rendering principles. In *Proc. CVPR*, 2014.

[63] K. Mitra and A. Veeraraghavan. Light field denoising, light field superresolution and stereo camera based refocussing using a GMM light field patch prior. In *Proc. CVPR Workshops*, pages 22–28, 2012.

[64] Y. Yoon, H. Jeon, D. Yoo, J. Lee, and I. S. Kweon. Light-field image super-resolution using convolutional neural network. *IEEE Signal Processing Letters*, 24(6):848–852, June 2017.

[65] Reuben A. Farrugia and Christine Guillemot. Light field super-resolution using a low-rank prior and deep convolutional neural networks. *CoRR*, abs/1801.04314, 2018.

[66] K. Honauer, O. Johannsen, D. Kondermann, and B. Goldluecke. A dataset and evaluation methodology for depth estimation on 4d light fields. In *Proc. ACCV*, 2016.

[67] S. Wanner, S. Meister, and B. Goldluecke. Datasets and benchmarks for densely sampled 4D light fields. In *Vision, Modelling and Visualization (VMV)*, 2013.

[68] C Dong, C.C Loy, K He, and X Tang. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision. Springer, pp. 184–199*, 2014.

[69] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017.

Declaration of independent work

I hereby affirm that I have independently written the Master's thesis on the topic:

Deep Autoencoder for Light Field Super-Resolution

and have not used any other aids or sources other than those I have indicated.
For parts that use the wording or meaning coming from other works (including the Internet and other electronic text and data collections), I have identified them in each case by reference to source or the secondary literature.
Furthermore, I hereby affirm that the above mentioned work has not been otherwise submitted as a thesis for a Master's examination. I further understand the pending completion of the review process I must keep the materials available that can prove this work was written independently.
After the examination process has been completed, the work will be submitted to the Library of the University of Konstanz and catalogued. It will thus be available to the public through viewing and lending. The described data given, such as author, title, etc. are publicly available and may be used by third parties (for example, search engine providers or database operators).

As author of the respective work, I consent to this procedure.

Konstanz, Feb. 2019 Signature: