Large Receptive Field Networks for Accurate High-Scale Image Super-Resolution

by

George Seif

Bachelor of Engineering, Electrical Engineering, Ryerson University, 2016

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Large Receptive Field Networks for Accurate High-Scale Image Super-Resolution

Master of Applied Science 2018

George Seif

Electrical and Computer Engineering

Ryerson University

Abstract

This thesis presents a novel convolutional neural network architecture for high-scale image super-resolution. In particular, we introduce two separate modifications that can be made to the convolutional layers in the network: one-dimensional kernels and dilated kernels. We show how both of these methods can be used to expand the receptive field and performance of super-resolution networks, without increasing the number of trainable parameters or network depth. We show that these modifications can easily be integrated into any convolutional neural network to improve performance. Our methods are especially effective for the challenging high scale super-resolution due to the expanded network receptive field. We conduct extensive empirical evaluations to demonstrate the effectiveness of our methods, showing strong improvements over the state-of-the-art.

Acknowledgements

Thanks to my supervisor, Dr. Androutsos for being the first to inspire my interest in the field of Image Processing and Computer Vision, and for always encouraging me to push for the best. Thanks to my parents for their unwavering support throughout both my academic life and personal life.

Contents

List of Tables

List of Figures

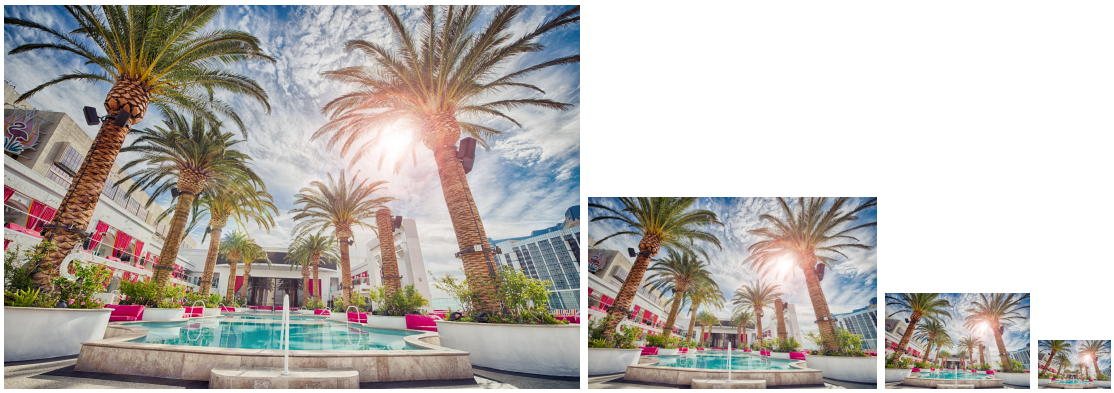
Chapter 1

Introduction

1.1 Problem Context

The resolution of a digital image is defined as the number of pixels that make up that image [1]. It is usually described in terms of width and height. For example, an image with a resolution of 500×500 has a width of 500 pixels and a height of 500 pixels. A High Resolution (HR) image is then defined as one that has many pixels relative to a Low Resolution (LR) image which has fewer pixels. In many digital imaging applications, HR images are highly desirable since HR images contain more details than their corresponding LR versions simply due to them being made up of more pixels and thus having a higher capacity for representing visual details and information. In the frequency domain, details correspond to high frequency components while smooth areas of the image correspond to low frequency components. Having more details aids in the improvement of pictorial information for human interpretation i.e aesthetics and helps with representation for automatic machine perception i.e machine vision [1, 2].

Single Image Super-Resolution (SISR) is a classic Computer Vision problem with the aim of recovering a High Resolution (HR) image from its LR version [2]. The Super-Resolution (SR) "scale" or "upscaling factor" refers to the relative difference in pixels between the LR and HR image in each dimension. For example, if the LR image has a size of 500×500 and the HR image has a size of 2000×2000, then the SR scale is said to be $\times 4$. This is illustrated in Figure 1.1 where we show examples of the HR image and corresponding LR images at $\times 2$, $\times 4$, and $\times 8$ scales. Upscaling from LR to HR becomes more challenging



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

Figure 1.1: Example of SISR. (a) The HR image with resolution 2040×1536. (b) LR image at $\times 2$ scale. (c) LR image at $\times 4$ scale (d) LR image at $\times 8$ scale.

with higher upscaling factors since less data is available from the LR image for reconstruction. For an upscaling of $\times 2$, we have a quarter of the original data to use for building the HR image. Similarly, for an upscaling factor of $\times 8$ with only have $\frac{1}{64}^{th}$ of the data making it much more challenging to reconstruct higher frequency details. The SISR process strives to increase the high frequency components and details as well as remove the degradation caused by the imaging process of the low resolution camera. SISR is a highly ill-posed problem because the LR image contains less data (i.e less pixels) than its corresponding HR image and thus there are many possible mappings from the LR to HR space. In particular, high frequency details present in the Ground Truth (GT) HR image are often missing from the corresponding LR version, especially with high upscaling factors [2]. With high upscaling factors, many fine image structures, details, and textures are often missing due to a lack of sufficient data to accurately represent them. Currently, state-of-the-art works in SISR are able to achieve high reconstruction accuracy on lower SR scales and thus are moving more towards working on the higher scales such as $\times 4$ and $\times 8$.

Recently, deep Convolutional Neural Network (CNN)s have largely dominated the state-of-the-art in SISR due to their high representational power and ability to learn complex mappings in an image-to-image transformation manner. Many of the most recent state-of-the-art CNNs draw ideas from the image classification domain, using the same techniques that have improved classification accuracy [3, 4, 5, 6, 7, 8] to also increase SR reconstruction accuracy. The trend has thus been towards constantly adapting the latest techniques from image classification and building deeper models with more layers and parameters.

While deeper models can yield higher SR reconstruction accuracy in terms of Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity index (SSIM) they also come with a number of drawbacks. Firstly, deeper models tend to have a very large number of parameters which leads to them requiring a larger memory bandwidth. Recursion has been successfully used in past works [9, 10] to increase depth while mitigating memory consumption. However, in order to maintain state-of-the-art accuracy while minimizing memory consumption, a high number of recursions tend to be required and thus many layers. When using many layers a second drawback presents itself: some speed must be sacrificed for accuracy, even when using recursion to minimize memory consumption. Thirdly, due to both the large number of parameters and layers, deep networks are difficult to train, often requiring careful design of the learning rate schedule and gradient clipping [9, 10, 11].

1.2 Scope and Contributions of this Thesis

In this thesis, we move in a new direction and discuss how to increase the receptive field of SR networks without increasing the network depth or parameter count to have a more efficient use of parameters while focusing on the more challenging high upscaling factors. In particular, we propose two different methods to expand the receptive field: atrous convolutions and one-dimensional filters inspired by separable filtering. Atrous convolutions expand the receptive field by using spacing between the weights in the convolutional filters [12]. One-dimensional filters can expand the receptive field by use of large kernels. We show how both techniques can be applied to SR networks to increase reconstruction accuracy while maintaining layer and parameter count, and without major sacrifices in speed unlike current state-of-the-art methods. We probe through the design space of our proposed models, exploring several different one-dimensional filter and atrous convolution arrangement schemes. Our proposed Large Receptive Field Network (LRFNet) demonstrates state-of-the-art accuracy in terms of PSNR and SSIM, while being easy to train and requiring low memory.

1.3 Overview of this Thesis

The rest of this thesis is structured as follows. Chapter 2 reviews the recent literature on Super-Resolution as well as other works in different domains of deep learning and computer vision that relate

to our proposed methods. Chapter 3 describes our model explorations and proposed methods in detail. In Chapter 4 we present our implementation details and experimental results on benchmark SISR image datasets. We conclude our work and discuss future research in Chapter 5.

Chapter 2

Related Work

Here we review the recent literature that is related to this thesis. We begin by looking at the methods before deep learning in Section 2.1 which we divide into two main categories: interpolation and sparse coding based methods. We then present some background information on convolution neural networks in Section 2.2 as well as current state-of-the-art techniques that are used in CNNs across many computer vision domains. Finally, in Section 2.3 we review the recent deep learning methods for SR, their major contributions, and their strengths and weaknesses.

2.1 Pre-Deep Learning Super-Resolution

Early methods in SR leveraged image edges to perform an edge-guided interpolation [13, 14, 15]. An interpolation in image processing is essentially a simple weighted averaging of neighbouring pixels; in the case of bilinear interpolation, the weighting of all pixels is equal [1]. However, with edge-guided interpolation, the weights of the pixels are determined by some statistical structure that we can define algorithmically. The main idea behind these methods is to make sure that the upscaled image is not smooth perpendicular to edges and that it is smooth parallel to edges, since edge directions should be invariant to resolution.

The basic procedure of edge-guided methods is shown in Figure 2.1. Beginning in Figure 2.1a, the HR image is first initialized with pixels from the LR image (black pixels) and all other pixels are left unfilled (white pixels). Then the unfilled pixels pixels indexed by two odd values (current one is marked

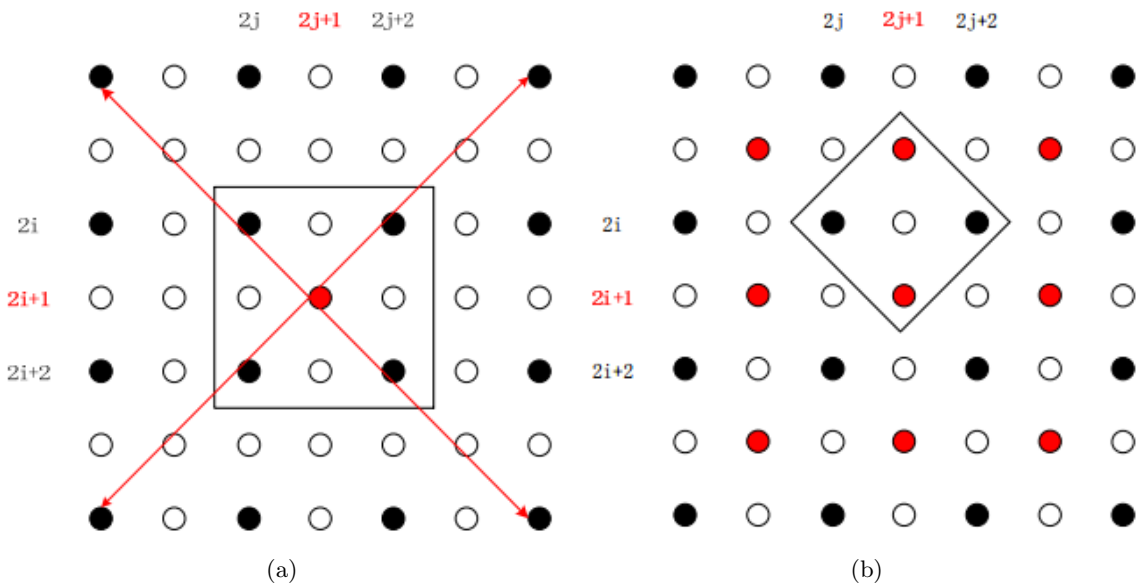


Figure 2.1: Illustrating the basic idea behind interpolation based methods. (a) Step 1, fill in the odd-indexed pixels using a weighted average of their four diagonal neighbours. (b) Step 2, fill in the even-indexed pixels using a weighted average of their two vertical and two horizontal neighbours.

as red) are then computed as the weighted average of its four diagonal neighbours (the black pixels from the original LR image). The reason we compute the odd-indexed pixels first is because they actually have four diagonal neighbours, where as the even-indexed pixels do not. Once the odd-indexed pixels have been determined as shown in Figure 2.1b, we fill in the even pixels by taking a weighted average of its vertical and horizontal neighbours. The weights are determined by the local covariance in a sliding window. If the pixels have positive covariance, then the area is likely smooth and the neighbouring pixels should be weighted equally. If the pixels have negative covariance, then there is likely an edge structure within our sliding window and thus some pixels are weighted more than others to maintain the edges.

Edge-guided interpolation methods usually lacked the ability to produce HR images with high-frequency, realistic textures since local covariance didn't given enough information about local structures. To address this limitation, sparse coding based approaches use dictionary learning to learn sparse signal representations for image patches [16, 17, 18, 19]. These methods leverage the image statistics from a training dataset of HR and corresponding LR images. The training dataset contains a set of HR and LR image pairs and for each image (whether HR or LR), a sparse signal representation is learned. The key is that the dictionaries for the respective HR and LR image pairs are trained jointly, and thus

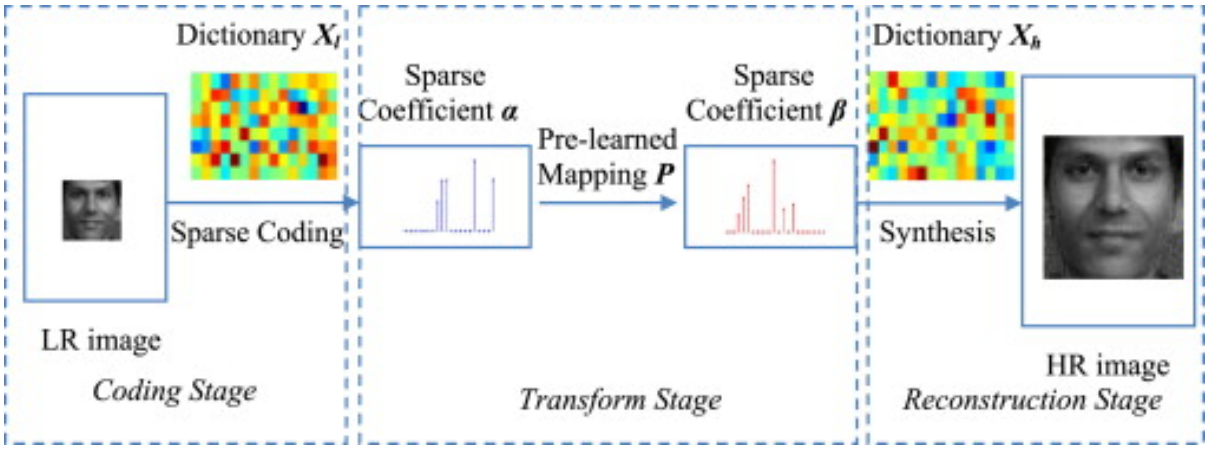


Figure 2.2: Illustrating the process of performing SISR using sparse signal dictionaries.

the similarity of sparse representations between the LR and HR image patch pairs with respect to their own dictionaries is enforced during training.

The procedure for performing Super-Resolution with these methods is shown in Figure 2.2. When the SISR operation is performed, patches are extracted from the LR images and then transformed into their sparse signal representations. The sparse signal is then matched to the most similar one in the learned sparse signal dictionary, either by a simple distance metric such as euclidean or using a statistical measure such as correlation. The image patch is then upscaled according to the LR-to-HR mapping that is associated with the retrieved sparse signals. The patch mappings are thus contained in a compact (i.e sparse) representation of the patch pairs in the learned dictionary, providing a strong reference for modelling natural image statistics. Such methods are more robust to noise and produce better edge structures and textures than interpolation based methods since they leverage a training dataset to model the image statistics, rather than only using the given LR image.

2.2 Convolutional Neural Networks

2.2.1 Basics of Convolutional Neural Networks

A Convolutional Neural Network (CNN) can be simply described as a multi-layered stack of convolution kernels with learned weights [20]. We illustrate the convolution operation in Figure 2.3. The 3x3 kernel in the figure is a matrix of weights which we move across the image from pixel to pixel in a sliding window

fashion. A CNN generally has many layers of these convolution operations where each layer typically has many unique convolution kernels. To compute the output pixel at each position we multiply the convolution weights by each corresponding pixel in the sliding window and compute the sum. This can be expressed mathematically as

$$z = \sum_n w_n x_n, \tag{2.1}$$

where x_n is an input pixel with a corresponding convolution weight w_n from the convolution kernel, and z is the single output pixel. The *kernel size* of a convolution refers to its spatial size; for example, a convolution with a kernel size of 3×3 has 9 weights arranged in a 3×3 array. The *stride* of a convolution refers to how the sliding window is passed over the image; for example, if we use a stride of 2, then the convolution is applied on every second pixel.

Convolutional Nets also typically have Activation Functions in-between each convolutional layer. An activation function is a non-linear mapping function that performs a simple operation to transform its single input value to a single output. Without these activation functions, a CNN would only be capable of performing linear transformations from input to output, since all of its convolution operations are made up of linear weighted sums. Thus, activation functions gives CNNs the ability to perform non-linear transformations by directly performing non-linear operations inside the network [20]. The most common non-linear activation function used in state-of-the-art CNNs is the *Rectified Linear Unit (ReLU)* [21]

$$z(x) = max(0, x), \tag{2.2}$$

where x is the input value i.e the input pixel and z is the output pixel. Earlier CNNs used the *tanh* or *sigmoid* functions as activations [22], but the recent state-of-the-art has consistently demonstrated improved performance using the ReLU function [3, 4, 6, 7, 21].

The goal of a CNN is to learn the most accurate transformation from input to output. To do this, the weights of its convolutional layers should be set to give the most optimal mapping possible. These weights are not set manually, but are in fact *learned* using the *gradient descent* algorithm [20]. Gradient descent is an iterative optimization algorithm for finding the minimum of a function, ideally the global minimum. To learn the weights of a CNN, we apply gradient descent to a pre-defined *loss function*. The loss is essentially a measure of how well the network is doing on the task it is being trained for as

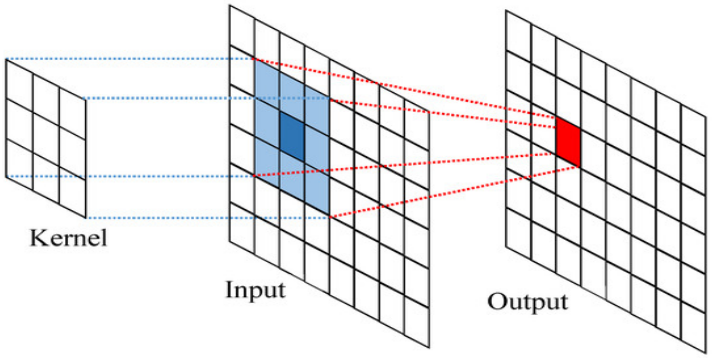


Figure 2.3: Illustrating the convolution operation.

evaluated on a labelled dataset. If we define our loss function as $\mathcal{L}$, the network weights on iteration n as w_n, and the *learning rate* as γ, the our gradient descent algorithm is expressed as

$$w_{n+1} = w_n - \gamma \nabla \mathcal{L}(w_n). \tag{2.3}$$

We are computing the gradient (partial derivative) of the loss function with respect to each convolution weight $\nabla \mathcal{L}(w_n)$ and subtracting it from the current weights because we want to move against the gradient, toward the loss function minimum. The learning rate γ controls how "fast" we move against the gradient. Having a higher learning rate could potentially find the minimum in less iterations, but our gradient descent could also end up diverging because the weights change too much and we "overshoot" the minimum.

Clearly, performing the gradient descent algorithm requires computing the gradient of the loss function with respect to network weights. For a CNN, this can be done using the *Backpropagation* algorithm [23]. Backpropagation is an iterative algorithm that applies the chain rule to propagate the error from the loss function back through all of the network layers to compute the gradient error for all of the convolutional layer weights. Gradient descent using backpropagation is a greedy algorithm and as such it is not guaranteed to find the true global minimum; but nonetheless, it has been shown to be effective in training CNNs to achieve state-of-the-art performance on many computer vision and image processing tasks.

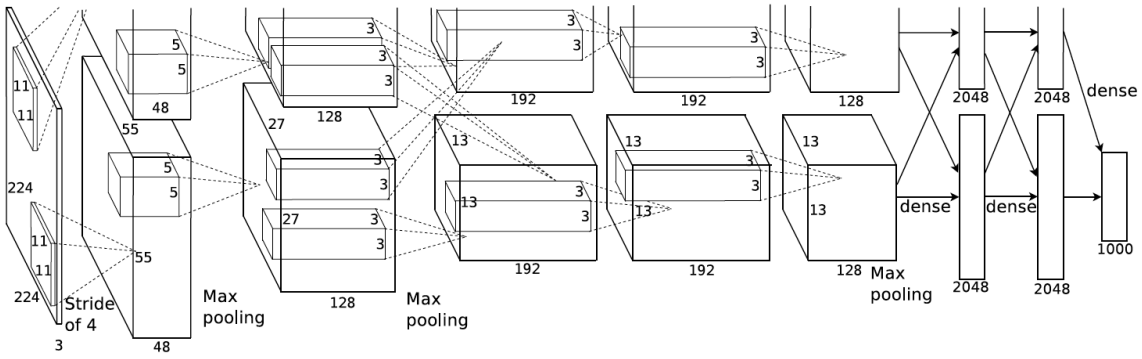


Figure 2.4: The AlexNet architecture.

2.2.2 State-of-the-art Techniques in Convolutional Neural Networks

The recent trend in applying CNNs to computer vision has been towards constantly adapting the latest techniques from image classification and applying them to other domains. Here we will review these techniques as they naturally provide the foundation knowledge of many state-of-the-art works in computer vision, including those in the SISR domain and that which is presented in this thesis.

The first major breakthrough in applying CNNs to computer vision tasks was made by Krizhevsky et al. with their AlexNet architecture, becoming the first to successfully use a deep network for large scale image classification. This was made possible because of the large amounts of labelled data from the ImageNet challenge dataset [24, 25], as well as training the model using parallel computations on two GPUs. They also used ReLU for the non-linearity activation functions, finding that they performed better and decreased training time relative to the tanh function. The ReLU non-linearity now tends to be the default activation function for deep networks. They also utilize data augmentation techniques that consisted of image translations, horizontal reflections, and mean subtraction, showing how the simple use of more training data results in improved performance. All of these techniques are very widely used today for many computer vision tasks. The AlexNet architecture is illustrated in Figure 2.4.

The VGG network introduced in [4] extended the ideas of AlexNet, building an even deeper network (more layers). In particular, their largest network has 19 layers vs AlexNet's 5. In addition, They do not use any large convolutional layers, only using kernels of size 3×3 (AlexNet's first two layers are 11×11 and 5×5 as shown in Figure 2.4). They show that two successive 3x3 convolutions create the equivalent network receptive field or "field of view" (i.e the pixels it sees) as a single 5x5 convolution; similarly,

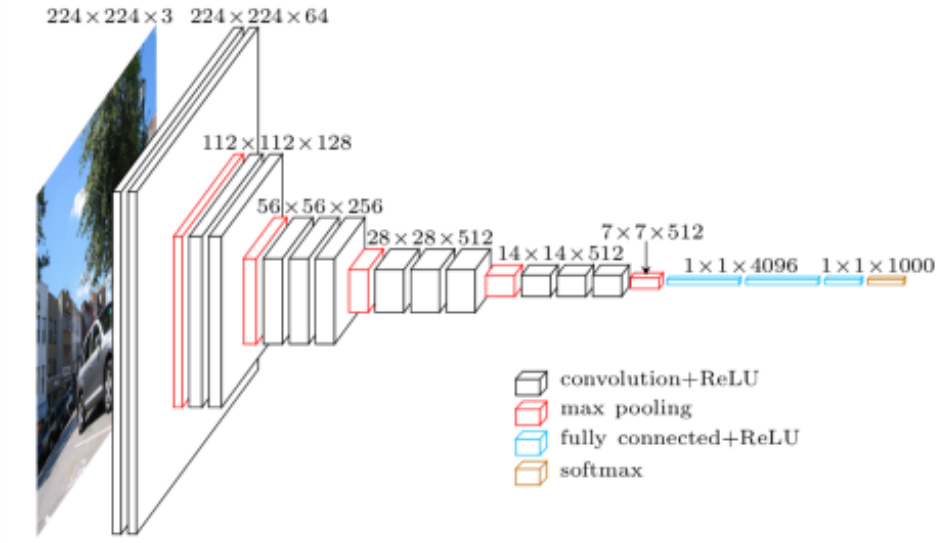


Figure 2.5: The VGGNet architecture.

three successive 3x3 convolutions are equivalent to a single 7x7. The advantage of this technique of using only small convolution kernels is that it simulates a larger kernel while keeping the benefits of smaller kernel sizes. The first benefit of smaller filters is a decrease in the number of parameters. The second is being able to use a ReLU function inbetween each convolution, that introducing more non-linearity into the network which makes the decision function more discriminative. The VGGNet architecture is illustrated in Figure 2.5.

The results of AlexNet and VGGNet may lead one to believe that simply stacking convolutional layers will always lead to better performance. However, He et al. [6] showed that this naive stacking of layers to make the network very deep won't always improve performance and can actually make the network perform worse. To address this issue, they proposed to use a residual learning technique with skip-connections, illustrated in Figure 2.6. The idea is that by using an additive skip connection as a shortcut, deep layers have direct access to features from previous layers. This allows feature information to be more easily propagated through the network [6, 26]. It also helps with training as the gradients can also more efficiently be back-propagated. He et al's ResNet was the first "ultra deep" network, where it is common to use over 100 - 200 layers.

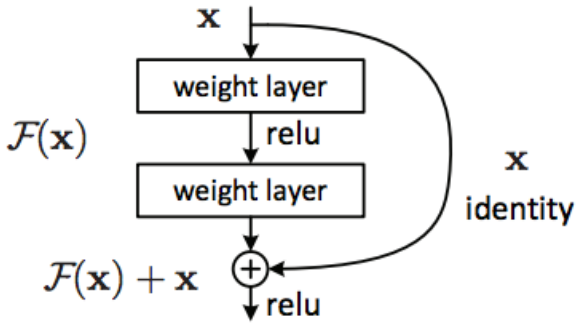


Figure 2.6: A Residual Block from ResNet.

2.3 Deep Networks for Super-Resolution

2.3.1 Introducing Convolutional Neural Networks for Super-Resolution

Dong et al. [27] were the first to train a CNN to learn the mapping from a LR image to its corresponding HR ground-truth. The Super-Resolution Convolutional Neural Network (SRCNN) architecture is shown in Figure 2.7. They use a three layer CNN with large kernels (9, 5, 5) with ReLU activations; they do not use any downsampling unlike the classification networks previously reviewed. For input to the network, LR image patches are upscaled to the HR image size using bicubic interpolation before being fed into SRCNN; this technique is adopted in a number of works following SRCNN. The SRCNN network was small and fast, yet outperformed all previous interpolation and sparse coding based methods in terms of PSNR and SSIM.

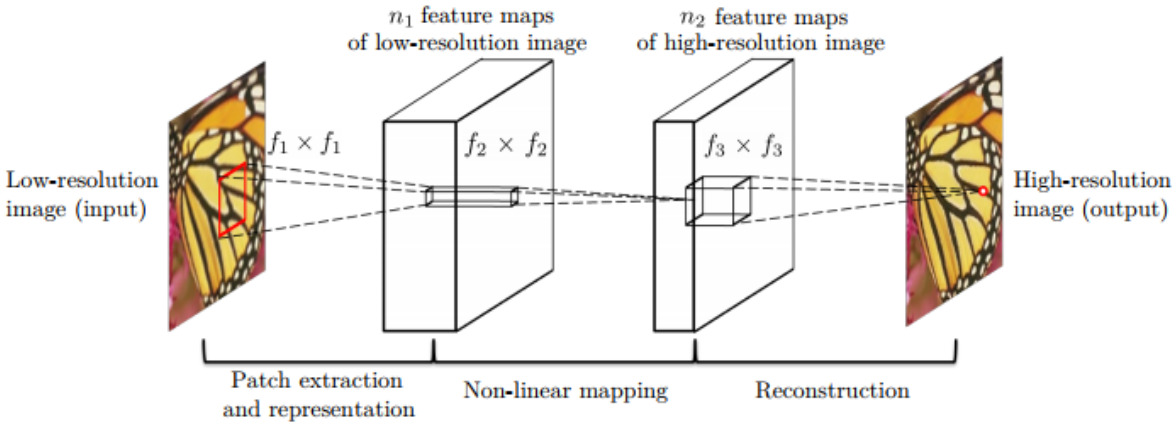


Figure 2.7: The SRCNN architecture.

In particular, the authors of SRCNN make a number of key contributions that are adapted in many future works on SSIM. They show consistent improvements in SR reconstruction accuracy when using more data. They also show that increasing the number of filters or kernel size both lead to performance improvements. Finally, they show that training on Red-Green-Blue (RGB) images achieves the highest performance as compared to training in other colour spaces. SRCNN used the Mean Squared Error (MSE) as the loss function

$$\mathcal{L} = \frac{\sum_{x=1}^{W} \sum_{y=1}^{H} (|Y_{i,j} - X_{i,j}|)^2}{WH}, \tag{2.4}$$

where X is the bicubic upscaled LR image with width W and height H, Y is the ground truth HR image.

The one drawback of SRCNN is that the simple adding of layers does not always lead to improved performance [27]. However, note that SRCNN was published before Residual Network (ResNet)s [6], and therefore had not seen the benefits of increasing depth with residual learning. Future works in SISR which will be reviewed next, do adopt residual learning techniques to increase depth and SR reconstruction performance substantially.

2.3.2 Improving Reconstruction Accuracy with Deeper Networks

The authors of SRCNN showed that a naive stacking of convolutional layers did not lead to better SISR reconstruction performance and in fact decreased performance. The Very Deep Super-Resolution network (VDSR) network proposed by Kim et al. addresses this limitation by adopting the residual learning idea of ResNets [6]. In particular, they proposed to use an additive global short-cut connection which summed the input image (bicubic upscaled LR image to the HR image size) with the output of their VDSR network as shown in Figure 2.8. Since the input and output images are usually quite similar in the case of SISR, designing the network to only predict the residual i.e the difference between the input and output eases training and improves performance. He et al. were able to have 20 convolutional layers in their network and achieved performance far superior to that of SRCNN.

In order to achieve fast convergence, the VDSR network was trained using a very high learning rate ($\gamma = 0.1$) in comparison to SRCNN ($\gamma = 0.0001$). To do this, gradient clipping is used where the gradient are kept in a certain range, such that a high learning rate can be used while the exploding

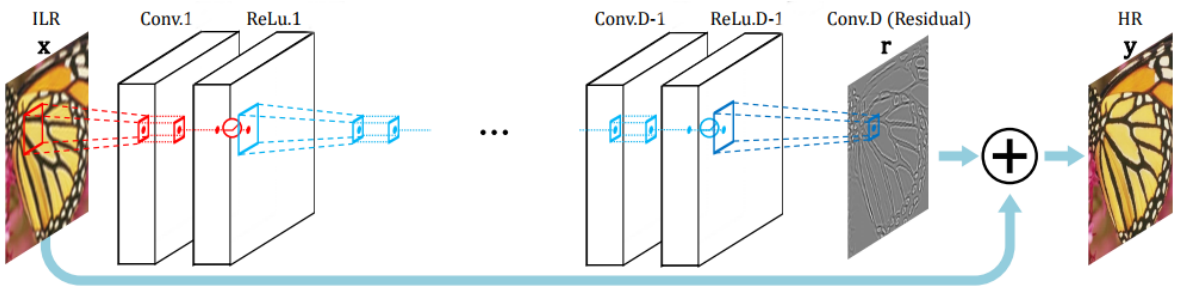


Figure 2.8: The VDSR architecture.

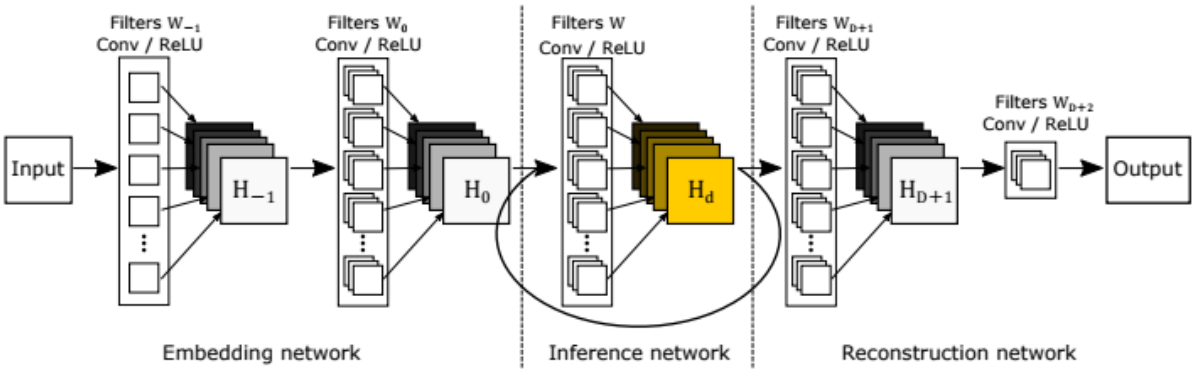


Figure 2.9: The DRCN architecture.

gradient problem [6] is avoided by the gradient clipping. Additionally, VDSR is trained using multiple SR scales ($\times 2, \times 3, \times 4$) to be able to use a single model for multiple upscaling factors; for SRCNN, a different model was trained for every upscaling factor. He et al. also demonstrated that improved performance is achieved when training with multiple scales rather than just a single one.

Although stacking layers with a shortcut connection does improve performance, it comes with the drawback of increasing the number of network parameters and thus memory requirements. To address this, Kim et al. proposed the Deeply Recursive Convolutional Network (DRCN) model [9] which used recursion to increase depth and performance while keeping the number of parameters constant. The basic DRCN model is shown in Figure 2.9. The key is that the inference network (the middle layer) is applied recursively i.e the same convolution is applied successively many times; in this way we are able to apply many more convolutions (i.e increased depth) without using more parameters. The DRCN model performs better than VDSR in terms of PSNR and SSIM. Both VDSR and DRCN use the MSE loss function.

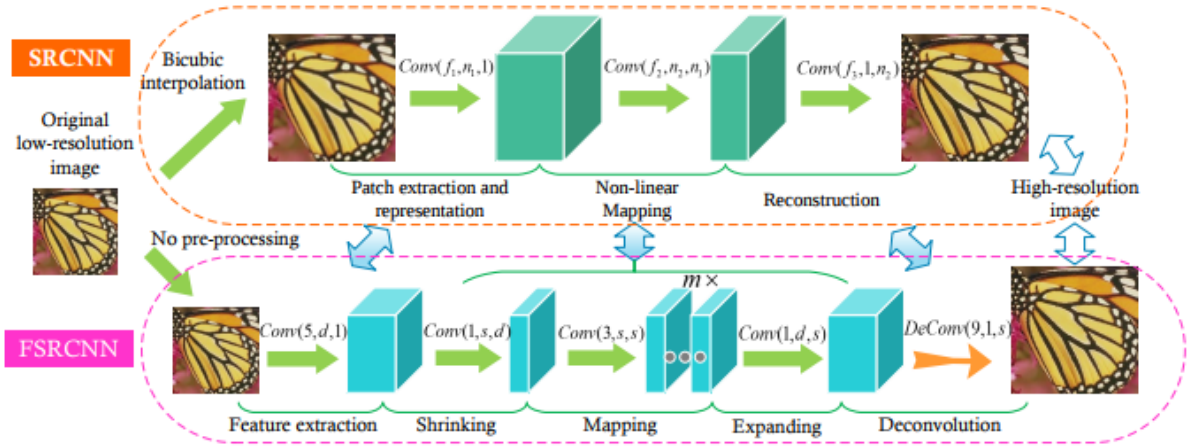


Figure 2.10: The FSRCNN architecture.

2.4 Fast Super-Resolution Processing

The previously reviewed deep CNNs for SISR used a bicubic upscaled image as input, where the LR image is scaled to the same size as the ground-truth HR image. Dong et al. proposed the Fast Super-Resolution Convolutional Neural Network (FSRCNN) model [28], arguing that upscaling with the bicubic was redundant and demonstrated that real-time speeds could be achieved by processing the LR image directly. The FSRCNN model is shown in Figure 2.10. The LR image is taken as input and directly processed through several convolutional layers. In the last layer, a strided transposed convolution [29] layer is used to upscale the LR image to the proper HR size. A transposed convolution essentially performs the opposite operation of a standard convolution; instead of going from multiple inputs to a single output, we go from a single input to multiple outputs. By using transposed convolutions at the end of the network, inference speed is directly proportional to the input LR image size and is faster for higher upscaling factors. Shi et al. used a similar technique with Efficient Sub-Pixel Convolutional Network (ESPCN) [30], again processing at the lower-resolution, but using a sub-pixel convolution to perform the upscaling. A sub-pixel convolution is simply a convolution with a fractional stride as in [31].

To strike a balance between performance and speed, the LapSRN model [32] shown in Figure 2.11 combines the ideas of VDSR and FSRCNN while using progressive upsampling. LapSRN uses a stack of sub-networks where each sub-network processes images at their lower-resolution and upscales at the end similar to FSRCNN, while using a global residual like VDSR for each sub-network to ease training.

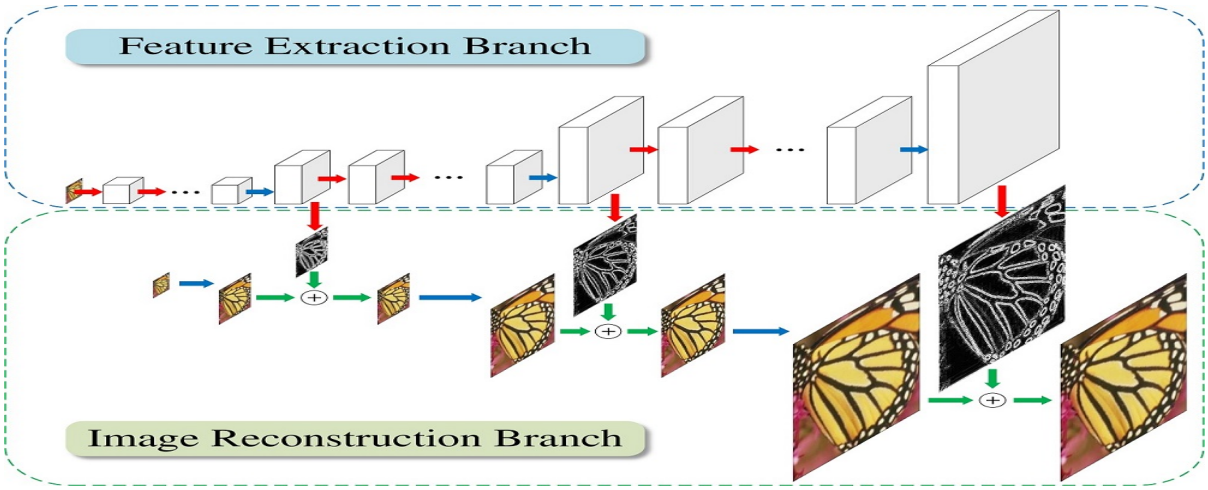


Figure 2.11: The Laplacian Super-Resolution Network (LapSRN) architecture.

In addition, a progressive upsampling structure is used where the LR image is processed and upscaled by a factor of 2 each time; thus, LapSRN is able to perform upscaling for $\times 2, 4, 8$ upscaling factors. All of these scales are trained *jointly* with *deep supervision* i.e on each forward propagation, the loss for all $\times 2, 4, 8$ scales is computed and summed which enables the network to achieve high performance on all of these scales using a single model. The LapSRN also uses the Charbonnier loss function [33] rather than the MSE used in past works [27, 11, 9]:

$$\mathcal{L} = \frac{\sum_{x=1}^{W} \sum_{y=1}^{H} \rho \cdot (|Y_{i,j} - X_{i,j}|)}{WH} \tag{2.5}$$

where $\rho = \sqrt{X_{i,j}^2 + \epsilon^2}$. The Charbonnier loss has been shown to be more robust to outliers than MSE or the Mean Absolute Error (MAE) [33]. The authors of LapSRN show that training with the Charbonnier loss helps the network achieve better performance than when using MSE.

2.5 Advanced Techniques for Super-Resolution

The most recent state-of-the-art networks have been very deep and use both local and global shortcut connections. Ledig et al. used a Generative Adversarial Network (GAN) [34] to tackle the SISR problem with the Super-Resolution Residual Network (SRResNet) shown in Figure 2.12. They train a generator network to predict the HR image given the original LR image and additionally train a discriminator to

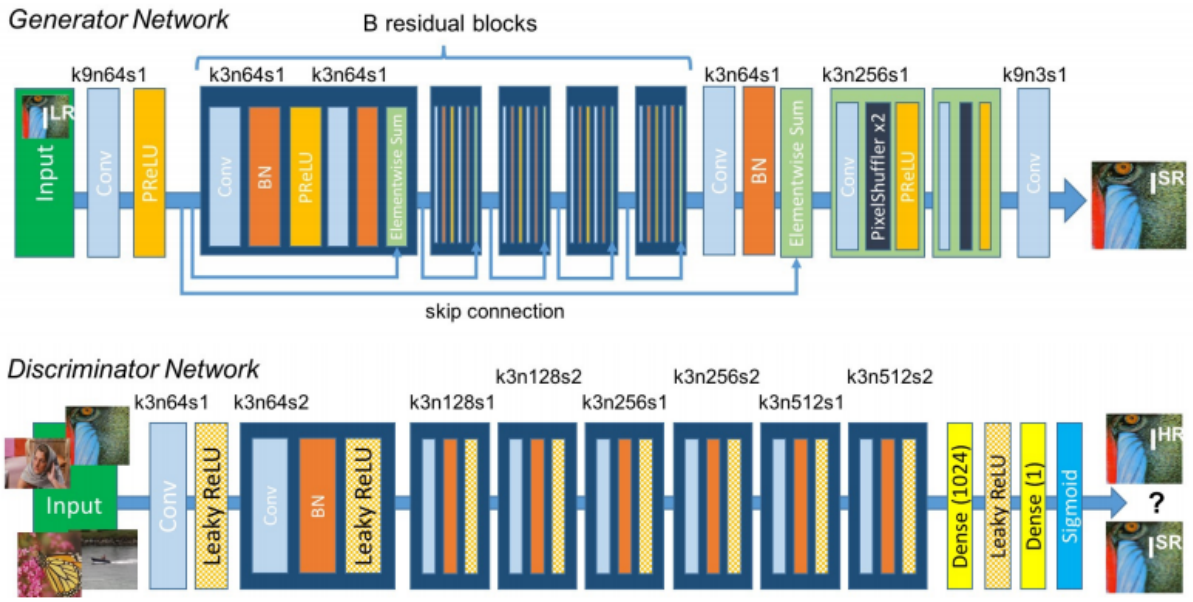


Figure 2.12: The SRResNet architecture.

distinguish between real HR images and those predicted by the generator. The input is processed at the low-resolution for higher speed and upscaled at the end using the sub-pixel convolution from ESPCN [30]. SRResNet uses local residuals inspired by ResNets [6, 7] to be able to stack many layers and a global residual like VDSR [11] to ease training. Their network is very deep with 16 residual blocks and a total of 37 convolutional layers. The one caveat is that SRResNet is only trained and benchmarked on $\times 4$ SR and thus its performance for other upscaling factors is unknown.

Tai et al. also use local residuals in their proposed Deep Recursive Residual Network (DRRN) [10], but drastically reduce the number of parameters compared to SRResNet by using recursion. The recursive residual block used in DRRN is shown in Figure 2.13. The local residual structure from ResNet V2 is used [7]. In the recursive block, the corresponding convolutional layers in the local residual units share the same weights i.e all green blocks have the same weights and all red blocks have the same weights and thus uses very few parameters. The full DRRN model only has four unique convolutional layers but has a total depth of 52 layers due to having 25 recursions for the local residual block. The use of local residuals is what allows the DRRN model to have such a high depth and achieve top performance. DRRN also uses a global residual, multi-scale training, and bicubic upscaled LR images for input exactly like VDSR. DRRN achieves state-of-the-art accuracy while only having a single unique residual block

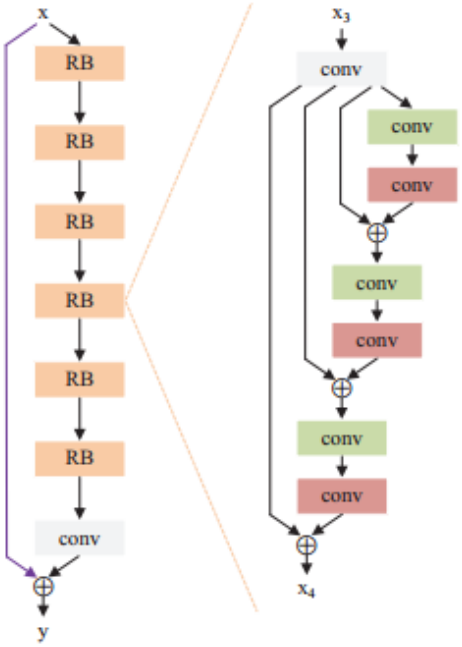


Figure 2.13: A recursive residual block used in the DRRN architecture.

and far less parameters than other models with comparable performance. The major drawback is that DRRN is quite slow as it has 52 total layers to process the HR sized images through and uses a higher number of feature maps for each layer than most previous works [27, 28, 30, 11, 9, 32, 34].

2.6 Main Drawbacks of Previous Methods

While these methods have shown strong promise in addressing the SR problem they have a number of drawbacks. Expanding the network receptive field by adding more and more unique convolutional layers comes with the direct disadvantage of an increase in the number of parameters. The DRRN model addresses this by using recursive residual blocks, but when using less unique convolutional layers more recursions are required to achieve the same performance 2.13 resulting in a lower inference speed. As we will show, using atrous convolutions and one-dimensional kernels allows for lower parameter requirements and network depth while increasing performance in terms of PSNR and SSIM.

Chapter 3

Technical Approach

This chapter presents the technical design of our LRFNet. In particular, we present two different LRF networks: one using one-dimensional kernels and the other using atrous convolutions. We begin with the theoretical explanation of our LRFNet design in Section 3.1 describing and showing how we can expanding the receptive field of SR networks without increasing total network parameters or depth. We then describe our design space exploration in Section 3.2, in particular how we validated our network design and explored the various layer arrangement schemes. Since we are conducting a design space exploration to determine the best design schemes for our networks, we will be benchmarking the performance of these different design schemes in Chapter 4 with our Experimental Results. Through our extensive experimentation, we determine the most optimal design parameters for our LRFNet.

To compare the performance of our models using one-dimensional kernels and atrous convolutions to regular convolutions we first construct a baseline model as shown in Figure 3.1. This baseline model has many design features that are similar to those of the current state-of-the-art [27, 11, 10, 34] which will be explained here. The input to our model is a bicubic upscaled LR image to the ground-truth HR size, such that our network can handle multiple scales using a single model as other state-of-the-art works in SISR have done [11, 9, 10]. Both the Bilinear and Bicubic interpolation have been used in past works in a similar manner [27, 11, 10]. We select the bicubic since it generally provides better initial PSNR and SSIM scores when compared to the bilinear [1, 2]. We use both global and local residual learning through global and local additive shortcut connections. The structure of each local residual block is adopted from the DRRN and SRResNet models where we remove the Batch Normalization [35] layer

and first ReLU and is shown in 3.3a. We use 12 residual blocks and all convolutions in the network are of size 3×3 with 64 filters and ReLU non-linearities, except for the last layer before the output which has no activation function. We denote our baseline model as LRFNet-B.

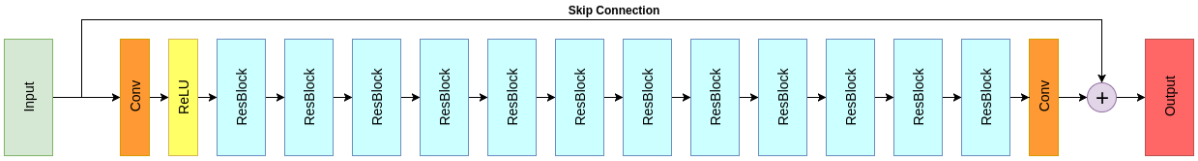


Figure 3.1: Our baseline model LRFNet-B. Our baseline model contains 12 residual blocks and a total of 26 convolutional layers. Each residual block is composed of a conv-relu-conv structure and an additive shortcut connection.

3.1 Receptive Field Expansion

We can formally define the receptive field of a CNN as the effective spatial field of view used to compute a single output pixel. Let us first consider the most simple case of a 3×3 convolution being applied to a single image as shown in a previous Figure 2.3 from Section 2.2. We can see that a 3×3 sliding window is used to compute a single output pixel. Thus, the receptive field of this basic network is 3×3.

There are two ways of expanding the network receptive field. Firstly, we could easily just use a larger kernel size. Referring again to Figure 2.3 where we only have a single convolutional layer, we can expand the receptive field to a spatial size of 11×11 just by using a convolution kernel size of 11×11. The drawback of doing this is that it is computational quite slow and requires many parameters. Expanding the receptive field from a size of 3×3 to 11×11 requires an increase in parameters and run time by a factor of $\frac{11 \times 11}{3 \times 3} = \frac{121}{9} = 13.44$. In addition, such models with larger kernels sizes for the convolutional layers have been found to be difficult to train [4, 27] due to being too complex and thus easily overfitting.

We can in fact expand the network receptive field in a much more parameter and computationally efficient manner by stacking multiple convolutions sequentially. Consider the illustration in Figure 3.2. Here two convolutions one after the other: a 5×5 kernel followed by a 7×7 kernel are used. As a result, this network has a receptive field of 11×11 but only uses $(5 \times 5) + (7 \times 7) = 74$ parameters and computations per pixel. If we simply use an 11×11 kernel to achieve the same receptive field of 11×11 then we would need 121 parameters and computations! Thus, stacking multiple convolutional layers gives us easy parameter and computational savings. In addition, using multiple convolutional

layers rather than just one big one to achieve the same receptive fields allows us to use more ReLU non-linearities between the convolutional layers, thus making the decision function for each pixel more discriminative. This stacking of smaller convolutional kernels to expand the network receptive field has been extensively studied by Simonyan and Zisserman with their proposed VGG network [4].

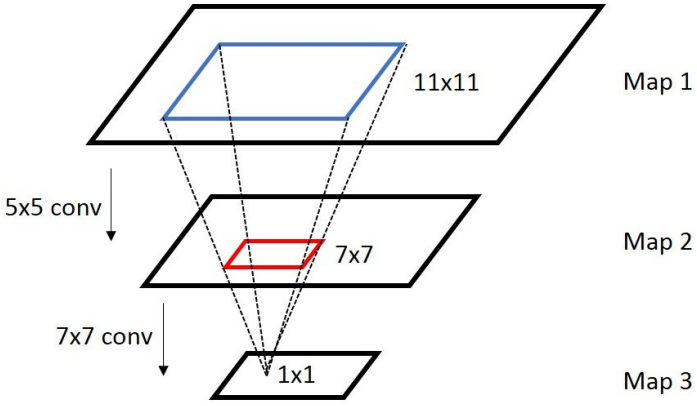


Figure 3.2: Illustrating the expansion of the network receptive field by stacking multiple convolutional layers.

Expanding the receptive field of deep CNNs has been shown to increase performance on a variety of Computer Vision tasks [36, 37, 4, 6, 7, 12]. A larger network receptive field directly captures more spatial context by using more pixels over a larger spatial area. This enhances the network's ability to model more complex representations and mappings. Specifically in the context of SISR, this increases the ability of the network to reconstruct larger and more complex edge structures.

While increasing the receptive field of deep CNNs can increase performance, there are a number of drawbacks that come with using the technique of stacking many convolutional layers. Firstly, deeper models tend to have a very large number of parameters which leads to them requiring a larger memory bandwidth. Naturally, when we stack more and more convolutional layers to expand the receptive field, we are introducing more convolution weights / parameters. Recursion has been successfully used in past works [10] to increase depth while keeping the parameter usage constant. This is done by applying recursion to the same residual block many times, thus reusing the same parameters many times to increase the depth and receptive field. However, in order to maintain state-of-the-art accuracy while minimizing memory consumption, a high number of recursions tend to be required and thus many layers [9, 10]. When using many layers a second drawback presents itself: some speed must be sacrificed for

accuracy, even when using recursion to minimize memory consumption. This is a direct result of having to pass feature maps through many convolutional layers which require many computations. Thirdly, due to both the large number of parameters and layers, deep networks are difficult to train, often requiring careful design of the learning rate schedule and gradient clipping [11, 9, 10].

In the next section we introduce our technical approach. We will discuss how to increase the receptive field of SR networks without increasing the number of layers or parameter count to have a more efficient use of parameters. One-dimensional filters can expand the receptive field by use of large kernels. Atrous convolutions expand the receptive field by using spacing between the weights in the convolutional filters. We show how both techniques can be applied to super-resolution networks to increase accuracy while maintaining layer and parameter count, without major sacrifices in speed, and while being easy to train and requiring low memory.

3.1.1 One-Dimensional Kernels

Consider Figure 3.3a which shows the receptive fields of two successive 3×3 convolutions; note that the ReLU non-linearities make no difference to the network receptive field since they are applied to every pixel individually and thus are not shown in the figure. Each 3×3 convolution kernel has 9 parameters (excluding the bias) and thus with two successive $3 \times 3s$ we achieve a receptive field of 5 in both the vertical and horizontal directions using a total of 18 parameters. However, we can achieve the same receptive field with less parameters by using one-dimensional kernels. In Figure 3.3b we use two successive 3×1 convolution kernels. This achieves the same vertical receptive field as the two square $3 \times 3s$ but with only $\frac{2}{3}^{rds}$ of the parameters and computations.

We can go even further and use two successive 5×1 convolutions as shown in 3.3c. This expands the receptive field to a size of 9 in the vertical dimension yet uses less parameters with 10 than the previous case of the two successive 3×3 convolutions which used 18. As an example extension of this idea, one could use two 1×5 convolutions and two 5×1 convolutions to achieve a receptive field of 9 in both directions using only 20 total parameters (10 parameters for each pair of one-dimensional convolutions). To achieve the same receptive field of 9 using only 3×3 convolutions (Figure 3.3a) we would need to use 4 successive $3 \times 3's$ which has a total of 36 parameters.

There are a few big advantages that come with using large one-dimensional kernels over small square kernels used in current state-of-the-art. Firstly, increasing the network receptive field using only two-

dimensional square kernels comes with the severe drawback of increasing the number of parameters by a large amount. The use of one-dimensional kernels gives us a much more relaxed trade-off between the network receptive field and the number of parameters without as much negative consequence. Secondly, using one-dimensional kernels results in less required computations to get the same output. For example, to get a single output pixel from two successive 3×3 convolution kernels we have to perform $(3\cdot3)+(3\cdot3) = 18$ multiplications and one addition. On the other hand, computing a single output pixel using one-dimensional kernels only requires $2 \cdot (1 \cdot 3) + 2 \cdot (3 \cdot 1) = 12$ multiplications and two additions if we want to achieve the same receptive field.

The use of the one-dimensional kernels is inspired by separable filtering used in Image Processing [1]. A separable filter is a convolutional filter that can be written as a matrix product of two other convolutional filters of lower dimensionality. This is typically a two-dimensional filter that can be separated into a product of two separate one-dimensional filters, thus reducing the computational load and number of parameters. For example, consider an integer approximated Gaussian filter which can be used to perform image smoothing

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}. \tag{3.1}$$

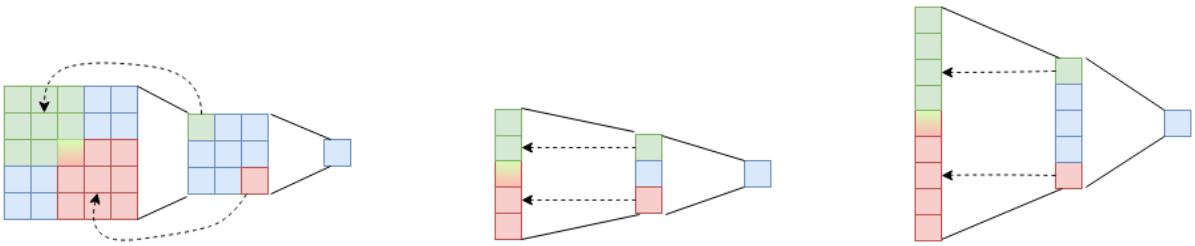
Such a filter has 9 parameters and requires 9 multiplications. However, we can re-write this kernel using one-dimensional kernels, by separating the square kernel into two vectors of one dimension each

$$\frac{1}{4}\begin{bmatrix}1\\ 2\\ 1\end{bmatrix} \times \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}. \tag{3.2}$$

By re-writing the kernel in this way we only have 6 parameters (vs 9 previously) and 6 multiplications (vs 9 multiplications previously), yet we are essentially performing the same mathematical operation.

One may question as to whether or not simply using only one-dimensional filters in a CNN will give the same performance as using square filters, since the one-dimensional filters may or may not turn out to be separable in a CNN. Indeed, we do not make any hard constraints on the training or the network to enforce the resulting filters to be separable. We instead rely on the network training and optimization to

converge to a minimum that can achieve strong performance from the larger receptive field, regardless of whether or not the final kernels end up being separable after the training. We show in our experiments that the network does in fact achieve greater performance through a more efficient use of parameters and computations with one-dimensional filters due to an expanded receptive field, perhaps making the corrections to adjust for only using one-dimensional filters during training. We denote the model using these one-dimensional kernels, inspired by separable convolutions as LRFNet-S.



(a) (b) (c)

Figure 3.3: Network receptive field from two successive convolutions. (a) With two 3x3 convolutions we get an overall receptive field of 5 in both the vertical and horizontal directions. (b) Here we use two 3×1 convolutions and get an overall receptive field of 6 in the vertical direction, the same as in (a) but with only $\frac{2}{3}^{rds}$ of the parameters and computations. (c) Here we use two 5×1 convolutions and get an overall receptive field of 9 in the vertical direction, much larger in the vertical than using a square 3x3 but with less parameters (10 total vs 18 total in the 3x3 case).

3.1.2 Atrous Convolutions

Atrous convolutions expand the network receptive field by using spacing between the weights in the convolutional filters [12]; this spacing is known as the dilation rate. Thus atrous convolutions have a wider view by looking at pixels that are further away from the center while using the same number of parameters as regular convolutions. A convolutional filter with a dilation rate of 1 has its weights separated by a distance of 1 and is equivalent to a regular convolutional filter; a convolutional filter with a dilation rate of 2 has its weights separated by a distance of 2 and so on. We denote the model using atrous convolutions as LRFNet-A.

The use of atrous convolutions is motivated by the fact that using dilated kernels allows for exponentially expanding network receptive fields without losing resolution or coverage. Consider the illustration of atrous convolutions in Figure 3.4. Figures 3.4a, 3.4b, and 3.4c show the convolutional weight spacings for dilation rates 1, 2, and 3. As can be seen, we can use dilation to easily and directly consider pixels

further away from the center all while using the same number of parameters and computations. Figures 3.4d, 3.4e, and 3.4f show the resulting receptive field after applying successive atrous convolutions with different dilation rates. As can be seen, we can use atrous convolutions to expand the receptive field quite substantially without increasing parameter count or computations. In the next section we will actually explore which dilation rates yield the biggest performance boosts.

Each dilation rate can be seen as considering a *scale* of contextual information. A convolution with dilation rate of 1 has the smallest scale, only taking into account information around the center of the window; i.e it has a smaller field of view and receptive field. But with higher dilation rates, for example 4, we are a much wider field of view since we are looking at pixels further away from the center i.e our network is viewing the spatial information at a higher scale. Since different dilation rates essentially correspond to different context scales, we can effectively use convolutions with different dilation rates throughout the network to explicitly capture multi-scale contextual information [12].

3.2 Design Space Exploration

Here we discuss our design space exploration using one-dimensional kernels and atrous convolutions for SISR. To build all of our proposed models, the square convolutional filters in each residual block are swapped out for convolutions using one-dimensional kernels (LRFNet-S) or atrous convolutions (LRFNet-A). For the 1-D kernels, each residual block has one $1 \times k$ and one $k \times 1$ convolution. For the atrous convolutions we simply dilate the convolutions within certain residual blocks where the two convolutions in the same residual block always have the same dilation rate.

Our baseline model uses the residual block structure shown in Figure 3.5a which has been shown to work well in SISR [10, 34]. However, this structure was successfully tested using only square 3x3 kernels. Thus we conduct experiments using three other residual block schemes shown in Figure 3.5 with our one-dimensional kernels. As the original inspiration for using one-dimensional kernels was the separable kernels from Image Processing, we attempt to replicate that exact computational process by having no extra non-linearity between the two 1-D kernels (in the vertical and horizontal directions). In Figure 3.5b we move the ReLU to be before the convolutions to test to see if allowing the feature maps to pass through the vertical and horizontal convolutions uninterrupted (i.e without the activation inbetween) improves performance. In Figures 3.5c and 3.5d we add the feature maps processed by the $1 \times k$ and

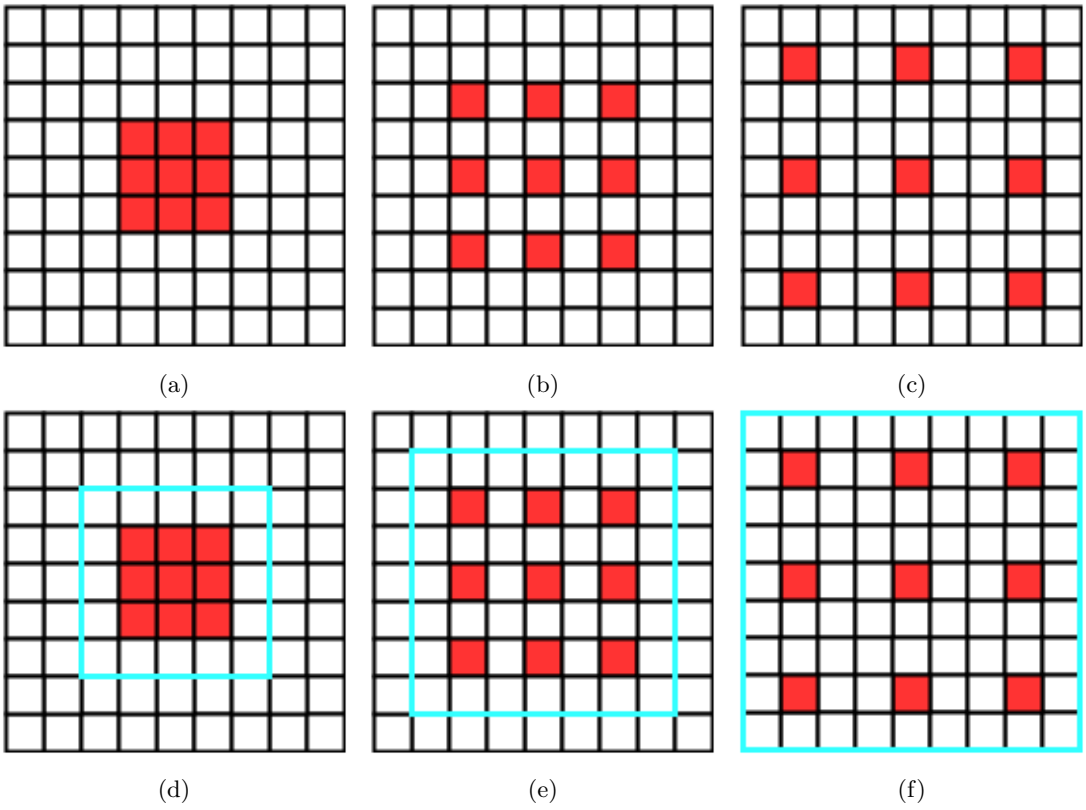
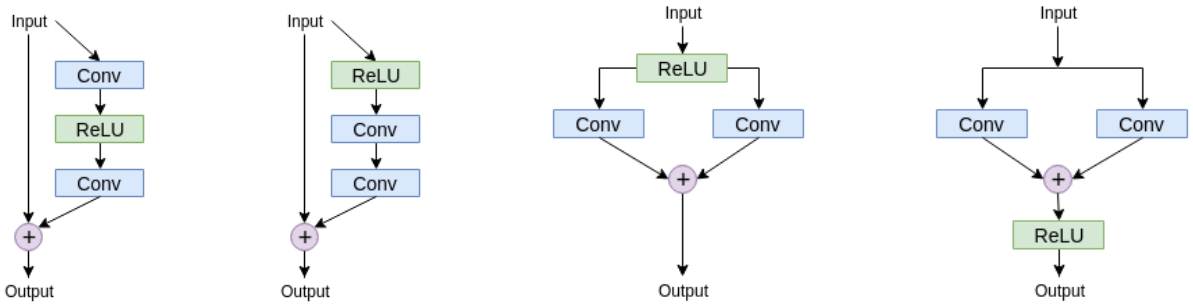


Figure 3.4: Illustration of Atrous convolutions and their receptive fields; we maintain all kernels at size 3×3. Red indicates the applied convolution (with corresponding dilation rate for the weight spacing) and blue shows the receptive field. (a) A single convolution with dilation rate 1. (b) A single convolution with dilation rate 2 (c) A single convolution with dilation rate 3. (d) Applying the convolution from (a) followed by the convolution from (d) and the resulting receptive field shown in blue. (c) Applying the convolution from (a) followed by the convolution from (e) and the resulting receptive field shown in blue. (f) Applying the convolution from (a) followed by the convolution from (f) and the resulting receptive field shown in blue.

the $k \times 1$ instead of performing sequential convolution. We test this to see if adding the feature maps from the $1 \times k$ and $k \times 1$ convolutions is more effective due processing in the vertical and horizontal directions. Activations before and after the convolutions for the additive case are also tested.

In addition to exploring different residual block schemes for one-dimensional kernels we also explore the design space for the atrous convolutions. The key questions we would like to answer are: How much dilation is required to increase performance? And does increasing the dilation rate always yield better performance? Specifically, we test out several settings of the dilation rate for the residual blocks. We denote the dilation rate as α. Our baseline model has 12 residual blocks; among these residual blocks



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

Figure 3.5: Exploring the network design space of local ResBlocks with one-dimensional kernels. (a) Baseline residual block used in DRRN and SRResNet models where we remove the Batch Normalization layer and first ReLU. (b) Moving the ReLU to before the one-dimensional kernels. We test this to see if allowing the feature maps to pass through the vertical and horizontal convolutions uninterrupted (i.e without the activation inbetween) improves performance. (c) and (d) Adding the feature maps processed by the $1 \times k$ and the $k \times 1$. We test this to see if adding the feature maps from the $1 \times k$ and $k \times 1$ convolutions is more effective due independent processing in the vertical and horizontal directions. We test placing the activation before (c) and after (d) the convolutions.

there are a total of 24 layers. We have tested the following dilation rate schemes: 1-2, 1-2-3, 1-3-5, and 1-4-8. A scheme with two dilation rates, for example 1-2, has the first 6 residual blocks with $\alpha = 1$ and the last 6 with $\alpha = 2$. A scheme with three dilation rates, for example 1-2-3, has the first 4 residual blocks with $\alpha = 1$, the second 4 with $\alpha = 2$, and the third with $\alpha = 3$.

Finally, we test to see if combining one-dimensional kernels with dilation further enhances performance. We conduct this test using multiple one-dimensional kernel sizes combined with the single best performing dilation scheme. We denote the model using both one-dimensional convolutions combined with dilation as LRFNet-SA. All models tested in our design space exploration are shown in Table 3.1.

Model	Residual Block Scheme	Kernel Size	Dilation Rate Scheme
LRFNet-B	A	3	1
LRFNet-B	A	5	1
LRFNet-B	A	7	1
LRFNet-B	A	9	1
LRFNet-B	A	11	1
LRFNet-S	A	7	1
LRFNet-S	B	7	1
LRFNet-S	C	7	1
LRFNet-S	D	7	1
LRFNet-S	A	3	1
LRFNet-S	A	5	1
LRFNet-S	A	7	1
LRFNet-S	A	9	1
LRFNet-S	A	11	1
LRFNet-A	A	3	1-2
LRFNet-A	A	3	1-2-3
LRFNet-A	A	3	1-3-5
LRFNet-A	A	3	1-4-8
LRFNet-SA	A	3	1-4-8
LRFNet-SA	A	5	1-4-8
LRFNet-SA	A	7	1-4-8
LRFNet-SA	A	9	1-4-8
LRFNet-SA	A	11	1-4-8

Table 3.1: Models tested in our design space exploration. The Residual Block Schemes correspond to those in Figure 3.5. For all of these models, we record the PSNR, SSIM, Run Time, and Parameter Count.

Chapter 4

Experimental Results

This chapter presents an empirical evaluation of the proposed algorithms. In the first Section 4.1 an overview of the datasets, metrics, and implementation used is provided. In Section 4.2 we conduct initial experiments to test and benchmark the effectiveness of one-dimensional kernels and atrous convolutions, as well as the various arrangement schemes, dilation rates, and kernel sizes. Finally, in Section 4.3 we compare our method to the current state-of-the-art for high-resolution SISR.

Our proposed models are built by swapping out the convolutions in each residual block for one-dimensional kernels (for LRFNet-S) or atrous convolutions (for LRFNet-A). It is critical to note that all of our models: LRFNet-B, LRFNet-S, LRFNet-A throughout our experiments here have a constant total of 26 layers but vary in their total number of parameters.

4.1 Experimental Implementation

4.1.1 Datasets

We train our models using the New Trends in Image Restoration and Enhancement (NTIRE) DIVerse 2K (DIV2K) dataset [38]. This dataset consists of 800 training images and 100 validation images all of high-quality at 2K resolution. Such high-resolution and high quality images have been shown to be very effective in training SISR networks [38] due to the diversity and high entropy of the images in the dataset. The NTIRE dataset is currently used for training most state-of-the-art networks and is therefore most

appropriate to use for training and validation [38]. The images are all of high quality both aesthetically and in the terms of having minimal amounts of corruptions like blur and noise. Since the images are very large the dataset provides sufficient training data to reach state-of-the-art performance.

Since super-resolution training data does not exist naturally (i.e we don't naturally have LR and HR images taken of the exact same information), we must create the data artificially using the DIV2K dataset.In order to generate the training data itself, we down sample each HR image in the dataset according the the super-resolution scale we are using, and then upscale that image back to the HR size using bicubic interpolation. We then crop out corresponding patches from the images for training where patches from the bicubic interpolated images are the input and patches from the HR images are the target output. This procedure is the same as many other past state-of-the-art works have done [27, 11, 10]. To take full advantage of the larger network receptive field, we use patches of 128x128 pixels which is larger than most works (the same as LapSRN [32], larger than all of the other methods reviewed in the previous chapters), cropping the patches from the training data with no overlap. Using patches that are too small may limit the ability of a network that's specifically designed with a large receptive field to achieve any performance advantage like our model.

We use random rotation and flipping for data augmentation as many other works have done to artifically create more training data and improve performance [11, 9, 32, 10]. The DIV2K dataset also contains 100 2K resolution validation images which we use for evaluation of our model in the RGB colour space. We additionally evaluate our model on four well-known SISR benchmark datasets: Set5 [39], Set14 [40], BSDS100 [41], and Urban100 [42]. The Urban100 dataset is particularly challenging due to the fine, high-frequency edges that tend disappear with downscaling when generating the input images.

4.1.2 Metrics

The two most common methods of evaluating the performance of SISR algorithms are Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index (SSIM) [43]. The PSNR measure is effective for evaluating the reconstruction performance of the algorithms for the raw pixel data since it is derived from the Mean Square Error (MSE) $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y} - Y)^2$. Given an image I the PSNR is defined

as

$$\text{PSNR} = \log 10 \frac{\max(I)^2}{\text{MSE}}, \tag{4.1}$$

where $max(I)$ is the maximum pixel value in the image I. Since the PSNR is derived from the MSE and thus really only measures the performance in terms of raw pixel accuracy, it does not take into account any information about the image structure or local context. Therefore, the SSIM is also used to evaluate SISR reconstruction performance as a more perception-based metric that considers image degradation as perceived change in structural information rather the only difference in raw pixel data. The full algorithm for the SSIM is provided by [43].

4.1.3 Implementation Details

We conduct initial experiments to test the effectiveness of one-dimensional kernels and atrous convolutions, as well as the various arrangement schemes, dilation rates, and kernel sizes. For these initial tests we use the first 100 images of the DIV2K dataset at $\times 4$ and $\times 8$ scale for training. As we will show, this is enough data to achieve results on par with state-of-the-art models and thus is sufficient for benchmarking of our proposed methods with the baseline without having to train using all of the training data which would take a very long time. Following these experiments we select the best performing models and fully train them along with the baseline model using the entire DIV2K dataset of 800 images on both $\times 4$ and $\times 8$ scales. All scales are used to train the same model and thus both $\times 4$ and $\times 8$ SR is achieved using a single network.

Our models are implemented in Keras with TensorFlow backend [44]. We use the Adam optimizer [45] to train our model and the MSE as our loss function. The initial learning rate is set to 10^{-4} and divided by 2 every 50 epochs, training for a total of 300 epochs on a 1080Ti GPU. Our models all take about 8 days to train on a single 1080Ti GPU using the entire DIV2K dataset.

Scheme	Scale	PSNR / SSIM
A	× 4	**28.32 / 0.804**
	× 8	**24.76 / 0.670**
B	× 4	28.28 / 0.803
	× 8	24.74 / 0.670
C	× 4	28.27 / 0.802
	× 8	24.73 / 0.668
D	× 4	28.27 / 0.803
	× 8	24.73 / 0.668

Table 4.1: Residual block design for one-dimensional kernels. Tests are run on the NTIRE validation set.

4.2 Benchmarking LRFNet

4.2.1 One-Dimensional Kernels: LRFNet-S

Here we benchmark the use of one-dimensional kernels in SISR. Our first experiment is determining whether the baseline residual block shown in Figure 3.5a is still optimal when using 1-D convolutions. We test out all of the schemes shown in Figure 3.5 using a constant kernel size of 7 for all schemes, where all kernels are one-dimensional. We denote them as schemes A to D for the Figures 3.5a to 3.5d. The results of our tests on the NTIRE validation set are shown in Table 4.1. The baseline residual block structure in scheme A still has the best performance even when using one-dimensional kernels. Moving the ReLU to before the two convolutions is noticeably detrimental to performance for scheme B. Processing the feature maps separately using the horizontal and vertical kernels for schemes C and D does not improve performance either. Thus, the original baseline residual block structure from scheme A is used for the rest of our experiments.

We now demonstrate how expanding the network receptive field by increasing kernel size using one-dimensional kernels noticeably and consistently improves the performance of SISR models. Again the number of layers remains fixed in our experiments. We train our 1-D kernels model LRFNet-S using different kernel sizes and evaluate the model performance on the NTIRE validation set. We also do the same for the baseline model with square convolutions. Our results are shown in Table 4.2 for PSNR / SSIM and Table 4.3 for the Parameters / Inference Time.

We can observe in Table 4.2 that increasing the size of 1-D kernels consistently improves performance in terms of PSNR and SSIM even up to a kernel size of 11. However, when using square convolutions

Kernel Size	Scale	Baseline PSNR / SSIM	LRFNet-S PSNR / SSIM
3	× 4	28.26 / 0.801	28.07 / 0.795
	× 8	24.69 / 0.668	24.54 / 0.662
5	× 4	28.29 / 0.803	28.23 / 0.800
	× 8	24.74 / 0.670	24.69 / 0.668
7	× 4	**28.31 / 0.803**	28.34 / 0.804
	× 8	**24.75 / 0.670**	24.76 / 0.671
9	× 4	28.29 / 0.803	28.42 / 0.806
	× 8	24.72 / 0.670	24.77 / 0.671
11	× 4	28.25 / 0.800	**28.45 / 0.808**
	× 8	24.69 / 0.668	**24.79 / 0.672**

Table 4.2: Comparing the performance of using different kernel size with 1-D kernels and the baseline. LRFNet-S indicates the model with 1-D kernels. Tests are run on the NTIRE validation set.

with the baseline model, performance only increases up to a kernel size of 7 and then decreases for sizes 9 and 11. We can also see in Table 4.3 that when we increase kernel size for the baseline model the number of parameters and inference time rise rapidly; the consequence then of increasing kernel size for better performance is quite severe when using square kernels. When using 1-D kernels however we don't have as extreme of a tradeoff and we can exceed the performance of the model with square kernels using far less parameters. As an example, using kernel size 7, LRFNet-S outperforms LRFNet-B (Table 4.2) while using less than 15% of the parameters and running at over 3 times the inference speed (Table 4.3). The results from both tables together show that as we increase the kernel size, the advantage of using 1-D kernels becomes more and more prominent in terms of performance (PSNR / SSIM), inference speed, and parameter count. For our final experiments with LRFNet-S, we set the kernel size to 9 to facilitate fair comparison to the baseline since both networks will have the same number of parameters this way.

4.2.2 Atrous Convolutions: LRFNet-A

We now turn our attention to atrous convolutions for SR with LRFNet-A. The receptive field of atrous convolutions is controlled by the dilation rate where a higher dilation rate creates a larger receptive field. We conduct experiments to reveal how much dilation should be used and if increasing the dilation rate, which directly increases the receptive field, consistently improves performance. The use of multiple dilation rates within our network to capture multi-scale context is also tested. Results are shown in

Kernel Size	Baseline Params / Time	LRFNet-S Params / Time
3	889k / 0.989	299k / 0.854
5	2,462k / 2.14	496k / 1.00
7	4,821k / 3.68	693k / 1.08
9	7,967k / 6.39	889k / 1.21
11	11,899k / 8.38	1,086k / 1.35

Table 4.3: Comparing the parameter count and inference time of using different kernel size with 1-D convolutions and the baseline. LRFNet-S indicates the model with 1-D kernels. Time is in seconds. Tests are run on the NTIRE validation set.

Scheme	Scale	PSNR / SSIM	Time
Baseline	× 4	28.26 / 0.801	0.989
	× 8	24.69 / 0.668	
1-2	× 4	28.34 / 0.804	1.27
	× 8	24.74 / 0.670	
1-2-3	× 4	28.36 / 0.806	1.37
	× 8	24.80 / 0.672	
1-3-5	× 4	**28.38 / 0.806**	1.38
	× 8	24.81 / 0.672	
1-4-8	× 4	28.36 / 0.805	1.45
	× 8	**24.84 / 0.673**	

Table 4.4: Comparing the performance of using different dilation schemes with Atrous convolutions all using 3x3 kernels. Time is in seconds. Tests are run on the NTIRE validation set.

Table 4.4. Even with our most basic scheme of 1-2 using a dilation rate $\alpha = 2$ for the second set of 6 residual blocks, we observe better performance both in terms of PSNR and SSIM compared to the baseline, still using the same number if parameters. As we introduce more dilation into the network with 3 different rates and as we increase each rate we observe consistently improving performance due to an expanded receptive field from schemes 1-2-3, to 1-3-5, to 1-4-8 all while using the same number of parameters as the baseline. We select the 1-4-8 scheme for our final LRFNet-A network due to its top performance on $\times 8$ scale and strong performance on $\times 4$ scale.

4.2.3 Can we get the Best of Both?: LRFNet-SA

In the previous sections it was shown that using large 1-D kernels and atrous convolutions independently improves the performance of SISR networks. This naturally raises the question as to if we can get the best of both worlds by combining the two techniques i.e do dilated 1-D kernels lead to further performance

Kernel Size	Scale	LRFNet-SA PSNR / SSIM	LRFNet-S PSNR / SSIM
3	× 4	28.04 / 0.794	28.07 / 0.795
	× 8	24.50 / 0.660	24.54 / 0.662
5	× 4	**28.16 / 0.798**	28.23 / 0.800
	× 8	**24.62 / 0.665**	24.69 / 0.668
7	× 4	28.14 / 0.797	28.34 / 0.804
	× 8	24.60 / 0.664	24.76 / 0.671
9	× 4	28.10 / 0.796	28.42 / 0.806
	× 8	24.58 / 0.663	24.77 / 0.671
11	× 4	28.03 / 0.793	**28.45 / 0.808**
	× 8	24.54 / 0.661	**24.79 / 0.672**

Table 4.5: Comparing the performance of using different 1-D kernel sizes with dilation. All models use the same dilation rate scheme of 1-4-8. Time is in seconds. Tests are run on the NTIRE validation set.

improvements? We train models with 1-D kernels, each with a different kernel size but all using the same dilation scheme of 1-4-8 which had the best performance out of all tested dilation schemes. The results are shown in Table 4.5. As can be seen, combining dilation with 1-D kernels provides no benefit, in fact performing worse than just using 1-D kernels without dilation.

4.3 Comparison to the state-of-the-art

We compare our proposed methods to several state-of-the-art networks for SISR: SRCNN [27], VDSR [11], DRCN [9], LapSRN [32], DRRN [10]. For the ×8 scale, we use the quantitative results and images of Lai et al. [32] who re-trained all of these state-of-the-art models for ×8 using the original settings from the papers since these models were originally only trained up to ×4 scale. To facilitate fair comparison against the state-of-the-art on the benchmark datasets (Set5 [39], Set14 [40], BSDS100 [41], Urban100 [42]), we convert the output images to the YCbCr colour space and compute PSNR and SSIM based on the Y-Channel only. The quantitative results for these benchmark datasets are shown in Table 4.7. We also report our quantitative results for the DIV2K validation set in Table 4.6; in this case we evaluate our model in the RGB colour space.

There are a number of key observations to make from our results in Table 4.7. Expanding the receptive field using one-dimensional kernels or atrous convolutions consistently improves SR performance, all while maintaining the number of parameters and depth of the network. For both ×4 and ×8 scales, our LRFNets achieve top performance by a large margin. Due to being able to increase performance while

Algorithm	Scale	DIV2K Val. PSNR / SSIM
Baseline	× 4	28.63 / 0.812
	× 8	25.03 / 0.679
LRFNet-S	× 4	28.76 / 0.816
	× 8	25.14 / 0.682
LRFNet-A	× 4	28.68 / 0.814
	× 8	25.24 / 0.684

Table 4.6: Comparing the performance of our fully trained models on both × 4 and × 8 scales for the NTIRE validation set. LRFNet-S uses 1-D kernels and LRFNet-A uses atrous convolutions.

maintaining depth, we also find that our models are also very easy to train using the Adam optimizer and do not require gradient clipping or any other modifications. The performance improvement using our method on the BSDS100 dataset is not as large as on the other datasets. Indeed, this is consistent with the other advancements made in deep learning SISR over the years. Many fine textures and small edges are completely removed when downsampling at a high-scale and so recovery of these is challenging. Substantial improvements on such a highly-textured dataset of images may not be possible using a standard pixel-based learning rate such as the MSE used in this work.

An interesting observation is that our LRFNet-S performs better than LRFNet-A at ×4 scale, but LRFNet-A performs better at the ×8 scale. In fact, this can also be seen in our preliminary experimental results from Tables 4.2 and 4.4. This is likely due to the use of large dilation rates in LRFNet-A (the 1-4-8 scheme). Having such large gaps in-between the convolution weights may be beneficial with ×8 scale, since the LR image itself for ×8 was obtained by skipping so many pixels. However, with ×4 scale, such large gaps may become detrimental and thus considering pixels that are adjacent to one another as in LRFNet-S helps to achieve better performance.

We will now further analyze how our model compares with the state-of-the-art in terms of the network depth, parameters, and run time tradeoffs with performance. We plot the PSNR vs network depth for ×4 and ×8 scales in Figures 4.1 and 4.2 on the challenging Urban100 dataset. For both scales, our models minimize the number of layers in the network while performing favourably over the state-of-the-art, even while those models use many more layers. The favourable performance of our models is even most evident on the $times$8 scale where having a larger receptive field is more effective since we have a high downsampling scale. With this graphical visualization, we can now also clearly observe how we use the techniques of large one-dimensional kernels (LRFNet-S) and atrous convolutions (LRFNet-A) to

Algorithm	Scale	Set5	Set14	BSDS100	Urban100
Bicubic	× 4	28.43 / 0.811	26.01 / 0.704	25.97 / 0.670	23.14 / 0.657
SRCNN [27]		30.50 / 0.863	27.49 / 0.750	26.90 / 0.710	24.52 / 0.722
VDSR [11]		31.35 / 0.883	28.01 / 0.767	27.29 / 0.725	25.18 / 0.752
DRCN [9]		31.54 / 0.884	28.03 / 0.768	27.24 / 0.725	25.14 / 0.752
LapSRN [32]		31.54 / 0.885	28.19 / 0.772	27.32 / 0.727	25.21 / 0.756
DRRN [10]		31.68 / 0.888	28.21 / 0.772	27.38 / 0.728	25.44 / 0.764
Baseline		31.68 / 0.888	28.29 / 0.775	27.36 / 0.729	25.45 / 0.764
LRFNet-S		31.91 / 0.890	28.44 / 0.778	27.47 / 0.733	25.70 / 0.773
LRFNet-A		31.82 / 0.889	28.38 / 0.777	27.39 / 0.730	25.61 / 0.769
Bicubic	× 8	24.40 / 0.658	23.10 / 0.566	23.67 / 0.548	20.74 / 0.516
SRCNN [27]		25.33 / 0.690	23.76 / 0.591	24.13 / 0.566	21.29 / 0.544
VDSR [11]		25.93 / 0.724	24.26 / 0.614	24.49 / 0.583	21.70 / 0.571
DRCN [9]		25.93 / 0.723	24.25 / 0.614	24.49 / 0.582	21.71 / 0.571
LapSRN [32]		26.15 / 0.738	24.35 / 0.620	24.54 / 0.586	21.81 / 0.581
DRRN [10]		26.18 / 0.738	24.42 / 0.622	24.59 / 0.587	21.88 / 0.583
Baseline		26.46 / 0.753	24.46 / 0.626	24.63 / 0.589	21.98 / 0.590
LRFNet-S		26.66 / 0.763	24.58 / 0.629	24.68 / 0.589	22.06 / 0.594
LRFNet-A		26.77 / 0.765	24.68 / 0.631	24.73 / 0.590	22.09 / 0.594

Table 4.7: Test results on benchmark datasets and the DIV2K validation set results (PSNR / SSIM). Red indicates the best performance and blue indicates the second best. Here, LRFNet-S uses a kernel size of 9 and LRFNet-A uses the 1-4-8 scheme.

improve the performance of the baseline without increasing the number of layers at all.

We then plot the PSNR vs number of network parameters for ×4 and ×8 scales in Figures 4.3 and 4.4 using the same Urban100 dataset. Our models use slightly more parameters than the LapSRN but have a substantially higher PSNR. DRRN uses very few parameters due to its heavy use of recursion vs our model which uses no recursion techniques. Still, our model has better reconstruction accuracy than DRRN especially on the ×8 scale where again the larger receptive field is more effective due to high downsampling.

Finally, we plot the PSNR vs run time for ×4 and ×8 scales in Figures 4.5 and 4.6 using the same Urban100 dataset. In both cases, our networks perform substantially better than DRRN, especially on the ×8 scale while being much faster at inference, running at about 3 times the speed of DRRN. DRRN uses 128 filters for every convolutional layer and has a total of 52 convolutional layers, all processed at high resolution making the network quite slow. On the other hand our network uses 64 filters for every convolutional layer and has only half the number of convolutional layers with 26. LapSRN does run faster at inference, but does not perform nearly as well as our proposed models.

From all of these quantitative experiments and visualizations we can conclude that our models per-

form favourably over the state-of-the-art on both $\times 4$ and $\times 8$ scales. Using one-dimensional kernels or atrous convolutions does slightly increase inference time, but not substantially so. The use of one-dimensional kernels or atrous convolutions offers strong performance improvements in terms of PSNR and SSIM with only a minor sacrifice in inference speed. Our models have a very efficient use of both parameters and depth in comparison to state-of-the-art methods.

We also show some visual results from Figures 4.7 to 4.16. There are images from the Set5 [39], Set14 [40], BSDS100 [41], and Urban100 [42] datasets. Across all of these datasets, the favourable performance over the state-of-the-art is visually evident. In particular, this favourable performance of our models is most evident when looking at images with sharp edges and well defined structures as the improved clarity and sharpness in the images can more easily be seen. Examples of such images and our improved reconstruction can be seen in Figures 4.7, 4.8, 4.11, 4.14, 4.15, 4.16. As mentioned before, fore images with very fine details and textures the improvements in SR reconstruction is less noticable both quantitatively and qualitatively. The images in Figures 4.9, 4.10, 4.12, 4.13 show examples of these types of images. Our models still produce visually better looking images, but the difference between the results of all of these models is less noticeable.

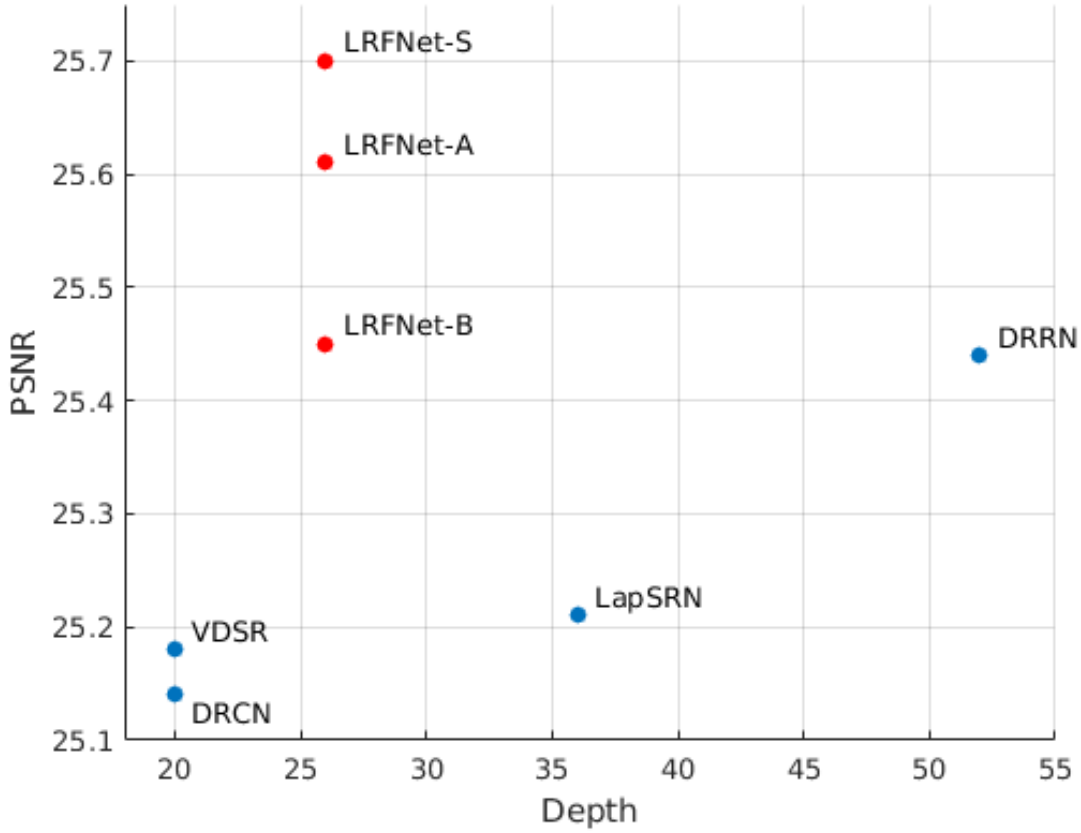


Figure 4.1: Plotting PSNR vs network depth for the state-of-the-art and our models for $\times 4$ scale on the Urban100 dataset.

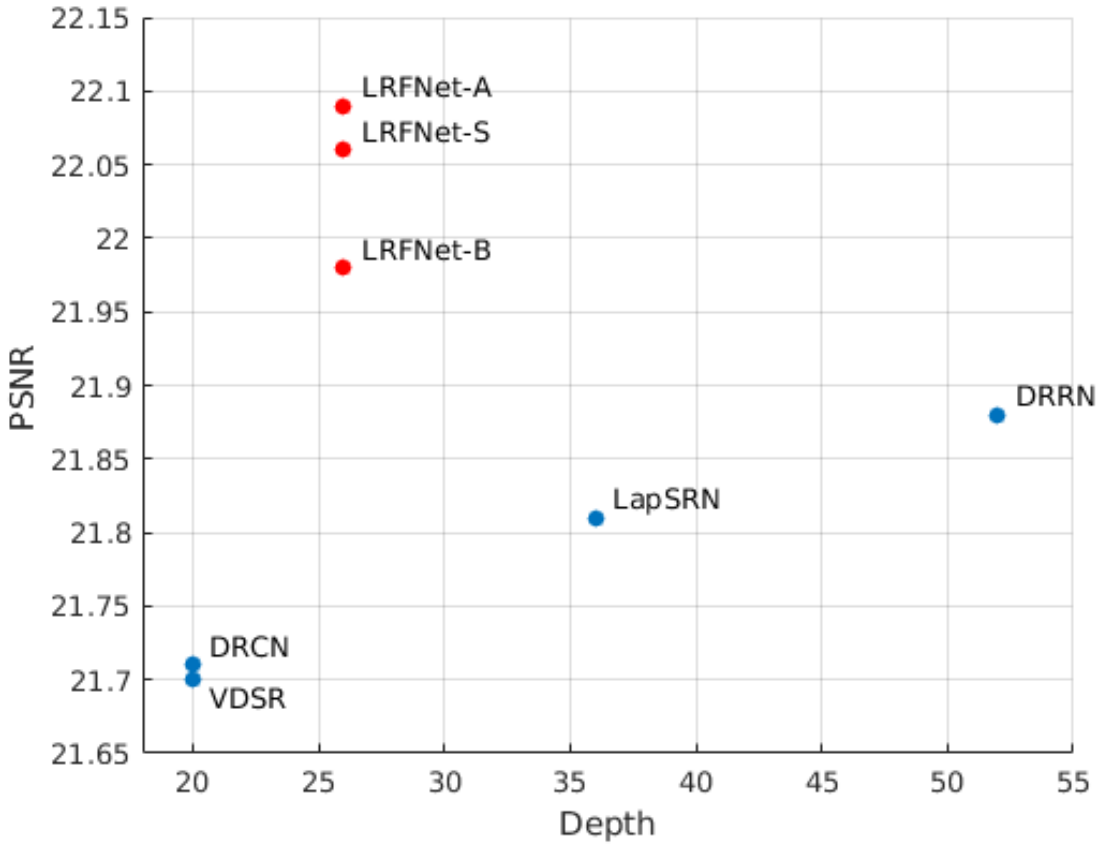


Figure 4.2: Plotting PSNR vs network depth for the state-of-the-art and our models for $\times 8$ scale on the Urban100 dataset.

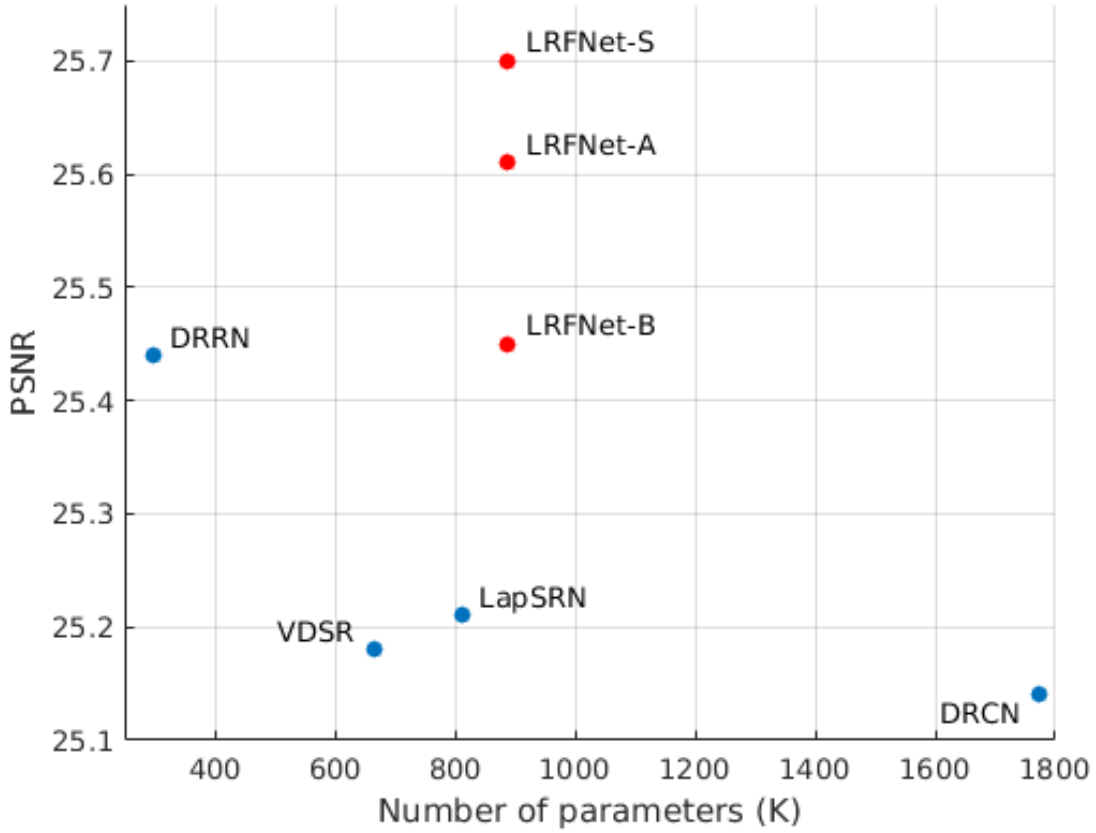


Figure 4.3: Plotting PSNR vs network parameters for the state-of-the-art and our models for $\times 4$ scale on the Urban100 dataset.

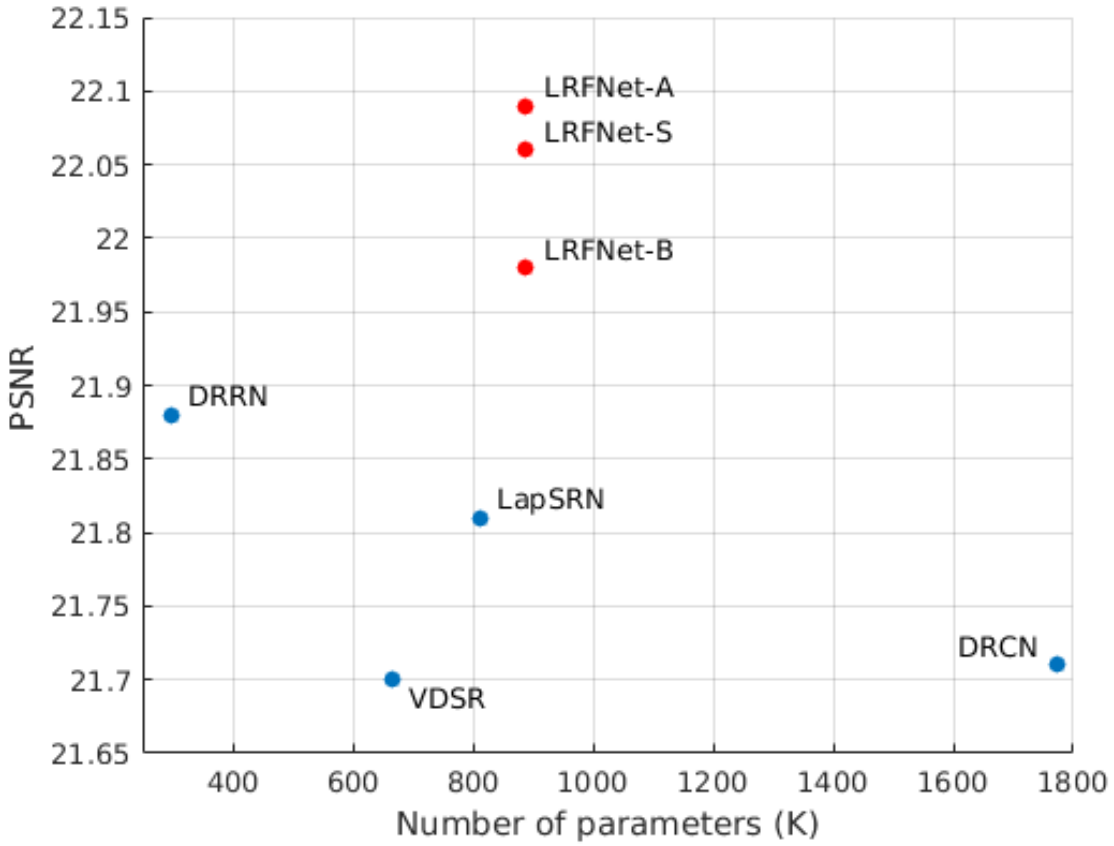


Figure 4.4: Plotting PSNR vs network parameters for the state-of-the-art and our models for ×8 scale on the Urban100 dataset.

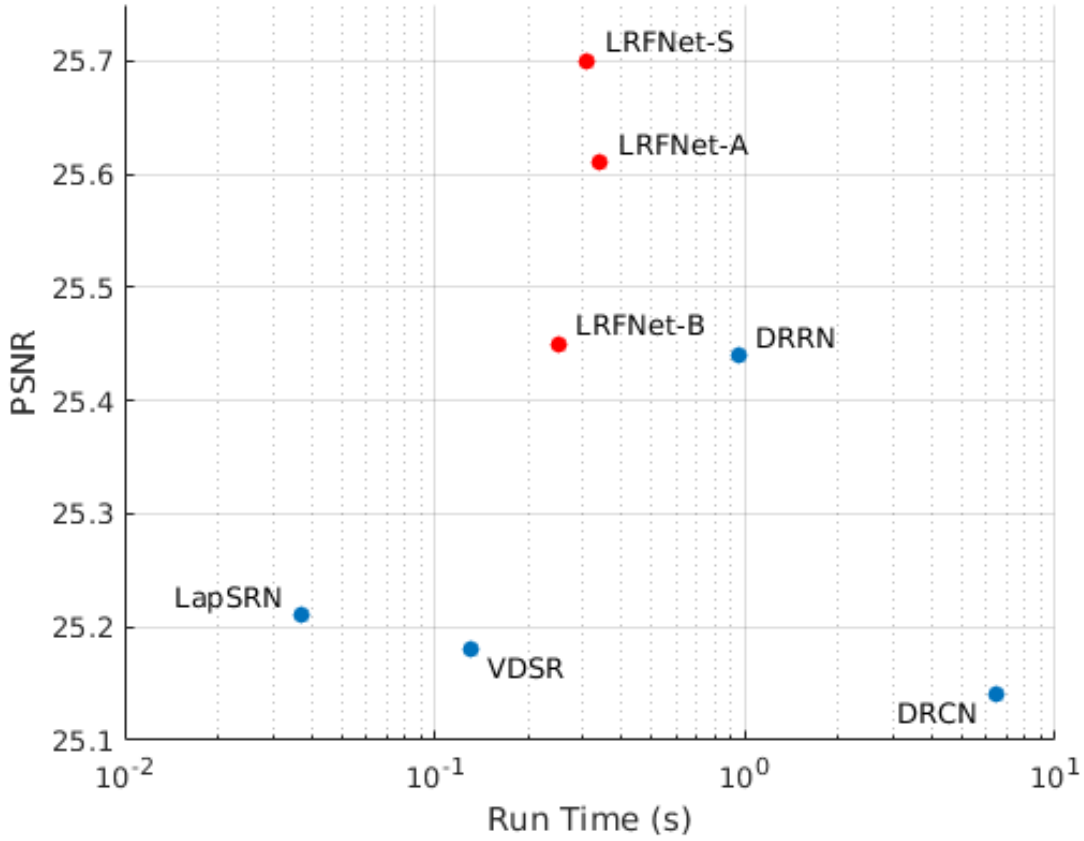


Figure 4.5: Plotting PSNR vs network run time for the state-of-the-art and our models for $\times 4$ scale on the Urban100 dataset.

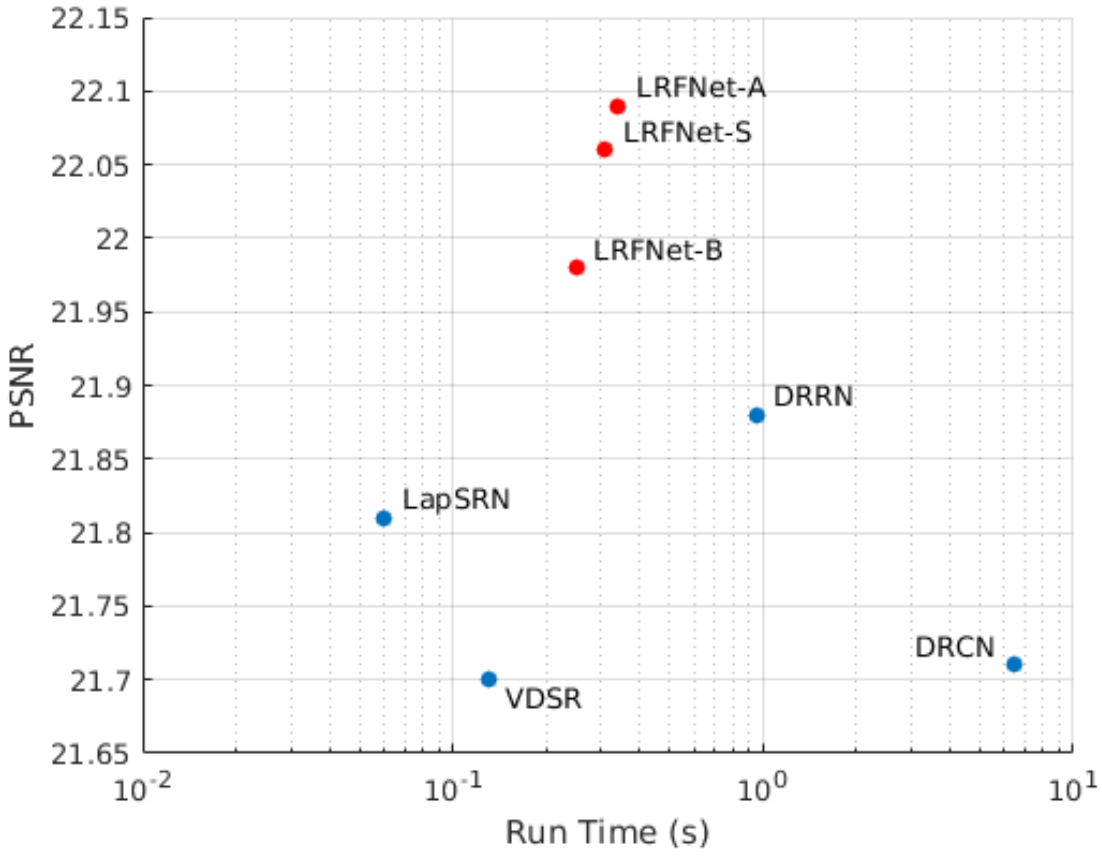
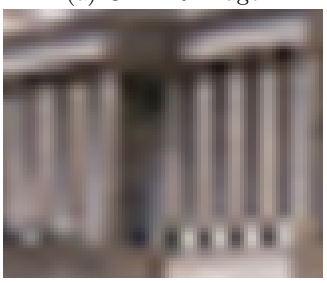


Figure 4.6: Plotting PSNR vs network run time for the state-of-the-art and our models for $\times 8$ scale on the Urban100 dataset.

(a) GT HR Image

(b) GT Patch



(c) Bicubic

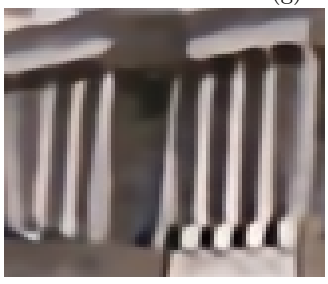
(d) SRCNN

(e) VDSR

(f) DRCN

(g) LapSRN

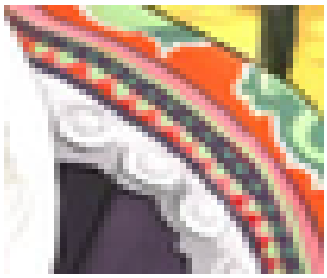
(h) DRRN



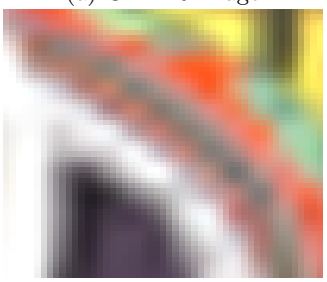
(i) LRFNet-S

Figure 4.7: Qualitative comparison of our best performing model on $\times 4$ scale with other works.

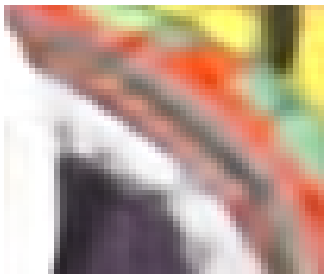
(a) GT HR Image



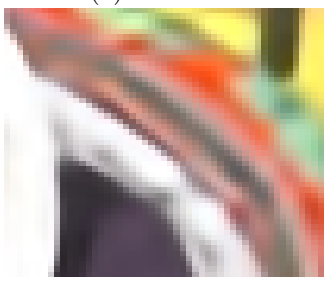
(b) GT Patch



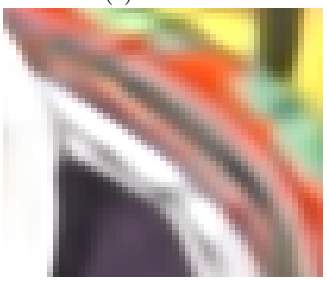
(c) Bicubic



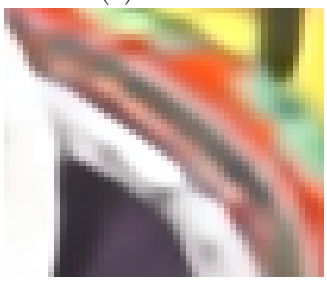
(d) SRCNN



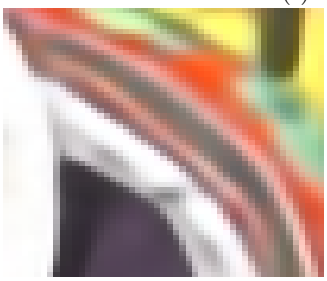
(e) VDSR



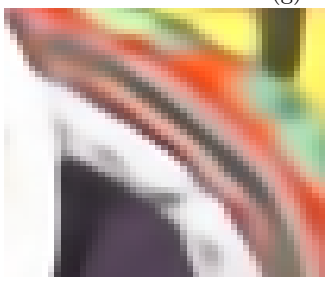
(f) DRCN



(g) LapSRN



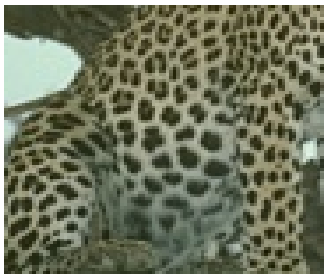
(h) DRRN



(i) LRFNet-S

Figure 4.8: Qualitative comparison of our best performing model on $\times 4$ scale with other works.

(a) GT HR Image



(b) GT Patch

(c) Bicubic

(d) SRCNN

(e) VDSR

(f) DRCN

(g) LapSRN

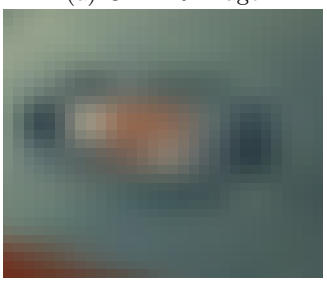
(h) DRRN

(i) LRFNet-S

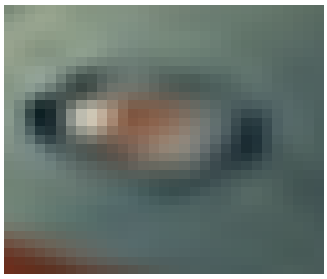
Figure 4.9: Qualitative comparison of our best performing model on $\times 4$ scale with other works.

(a) GT HR Image

(b) GT Patch



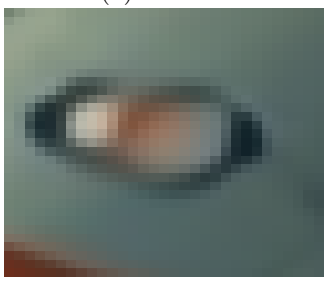
(c) Bicubic



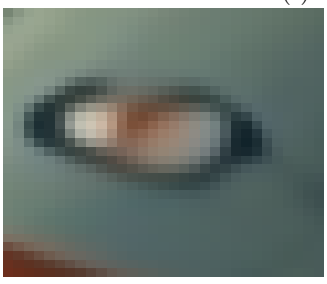
(d) SRCNN

(e) VDSR

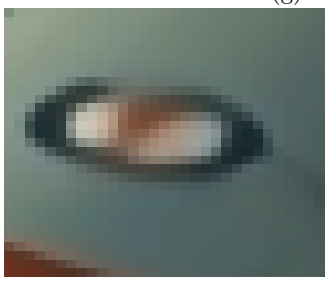
(f) DRCN



(g) LapSRN

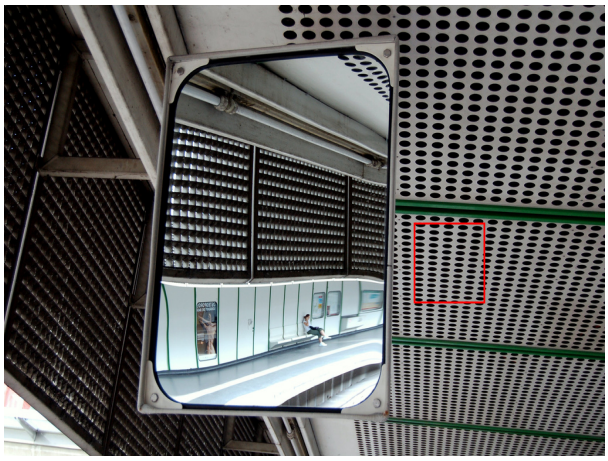


(h) DRRN

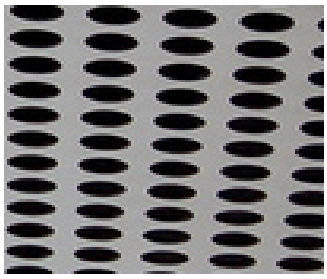


(i) LRFNet-S

Figure 4.10: Qualitative comparison of our best performing model on $\times 4$ scale with other works.

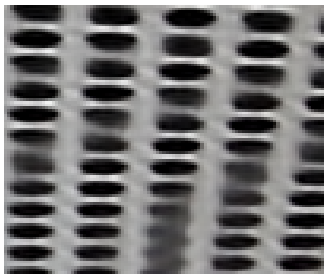


(a) GT HR Image

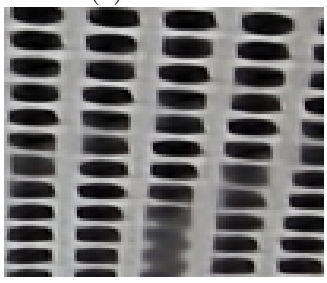


(b) GT Patch

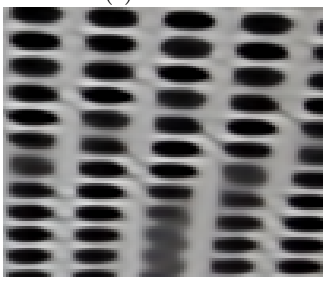
(c) Bicubic



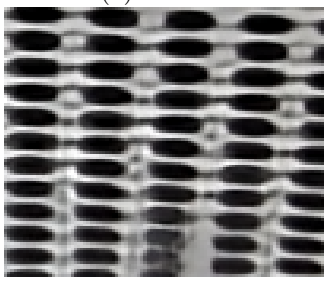
(d) SRCNN



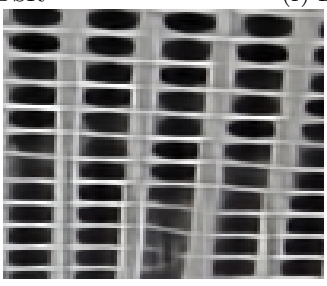
(e) VDSR



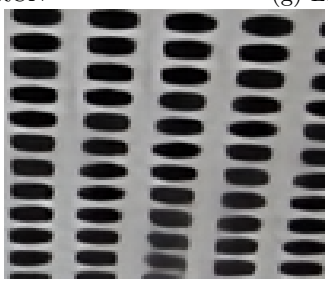
(f) DRCN



(g) LapSRN

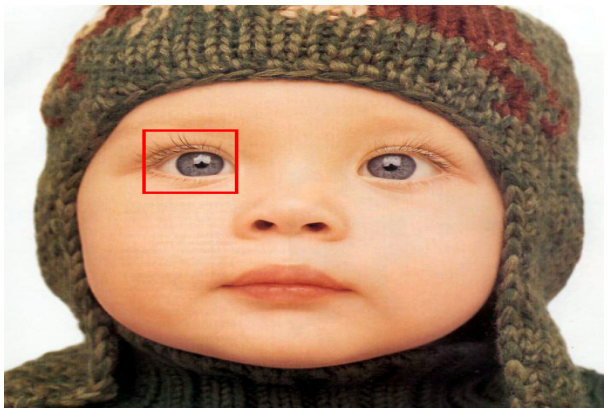


(h) DRRN

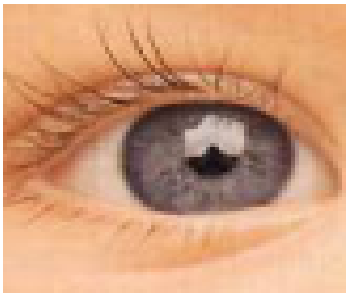


(i) LRFNet-S

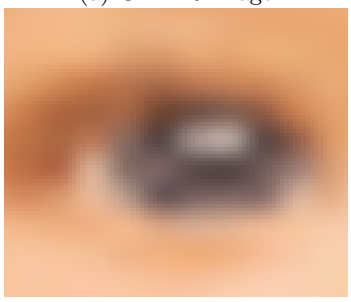
Figure 4.11: Qualitative comparison of our best performing model on $\times 4$ scale with other works.



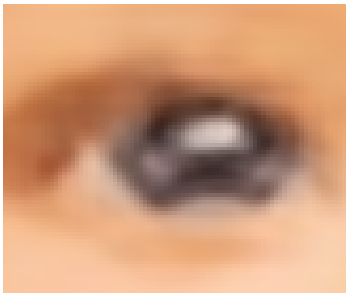
(a) GT HR Image



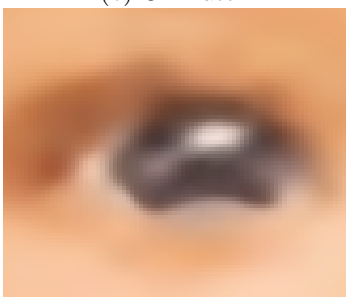
(b) GT Patch



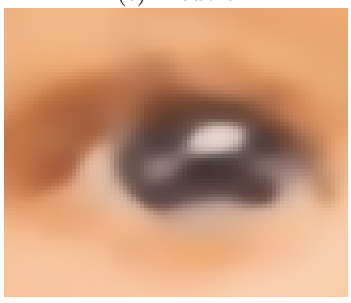
(c) Bicubic



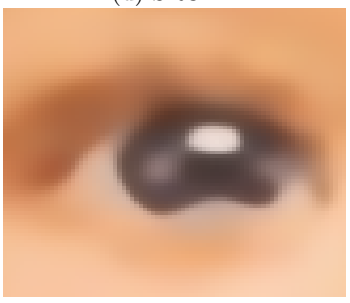
(d) SRCNN



(e) VDSR

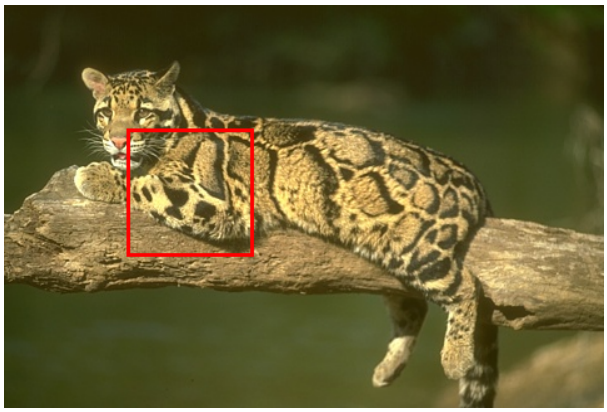


(f) LapSRN



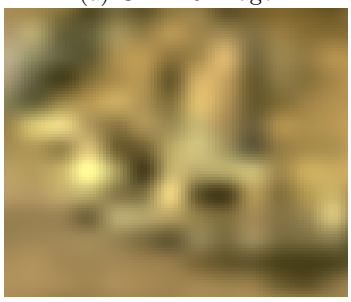
(g) LRFNet-A

Figure 4.12: Qualitative comparison of our best performing model on $\times 8$ scale with other works. Note that Lai et al's results for DRCN and DRRN were unavailable on their project page for $\times 8$ scale and so are not shown here.



(a) GT HR Image

(b) GT Patch



(c) Bicubic

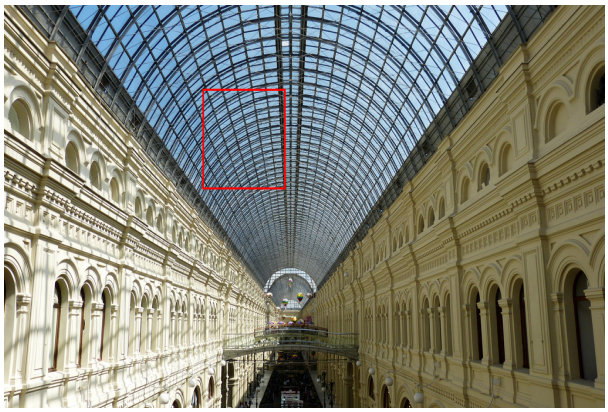
(d) SRCNN

(e) VDSR

(f) LapSRN

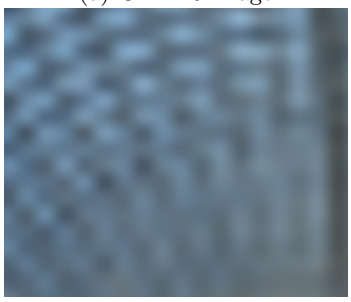
(g) LRFNet-A

Figure 4.13: Qualitative comparison of our best performing model on $\times 8$ scale with other works. Note that Lai et al's results for DRCN and DRRN were unavailable on their project page for $\times 8$ scale and so are not shown here.

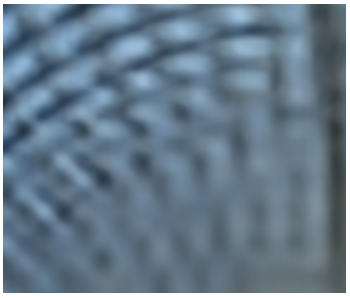


(a) GT HR Image

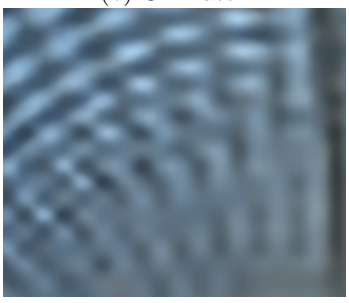
(b) GT Patch



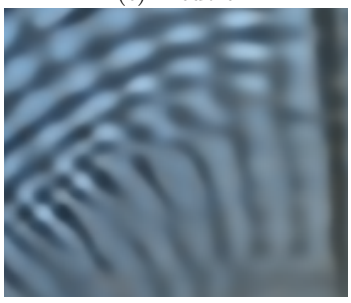
(c) Bicubic



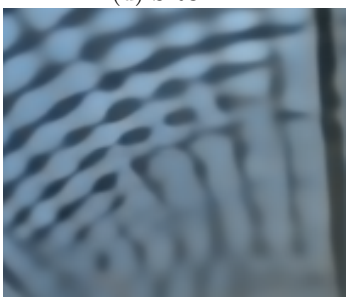
(d) SRCNN



(e) VDSR



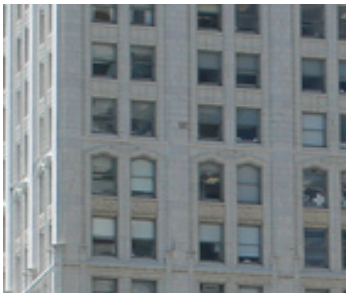
(f) LapSRN



(g) LRFNet-A

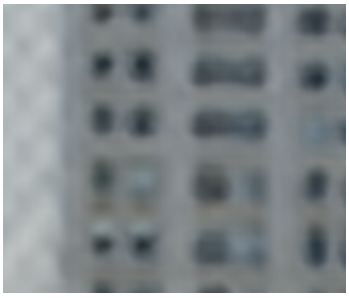
Figure 4.14: Qualitative comparison of our best performing model on $\times 8$ scale with other works. Note that Lai et al's results for DRCN and DRRN were unavailable on their project page for $\times 8$ scale and so are not shown here.

(a) GT HR Image

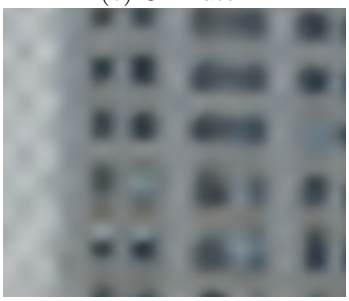


(b) GT Patch

(c) Bicubic



(d) SRCNN



(e) VDSR

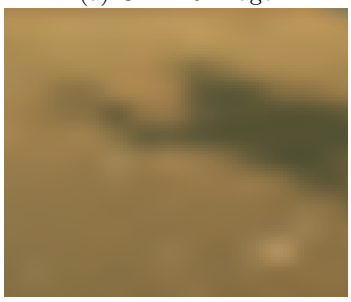
(f) LapSRN

(g) LRFNet-A

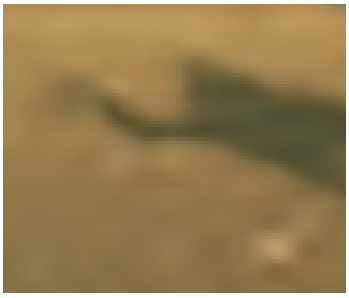
Figure 4.15: Qualitative comparison of our best performing model on $\times 8$ scale with other works. Note that Lai et al's results for DRCN and DRRN were unavailable on their project page for $\times 8$ scale and so are not shown here.

(a) GT HR Image

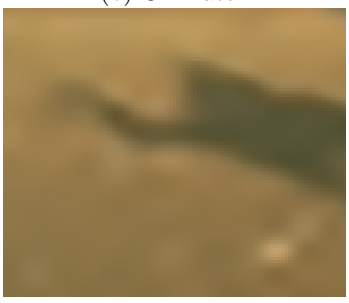
(b) GT Patch



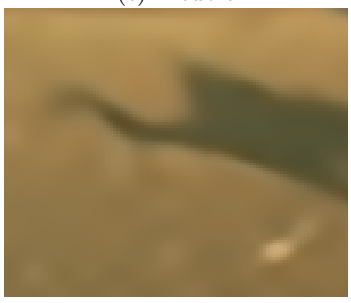
(c) Bicubic



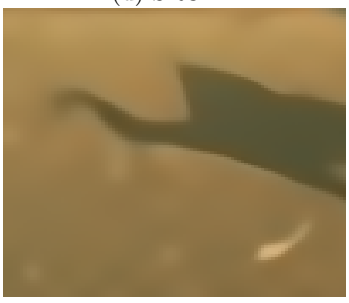
(d) SRCNN



(e) VDSR



(f) LapSRN



(g) LRFNet-A

Figure 4.16: Qualitative comparison of our best performing model on $\times 8$ scale with other works. Note that Lai et al's results for DRCN and DRRN were unavailable on their project page for $\times 8$ scale and so are not shown here.

Chapter 5

Conclusion

5.1 Thesis Summary

This thesis has presented two novel methods for improving CNNs for single image super-resolution. First, we have demonstrated how expanding the receptive field of CNNs allows the network to capture more spatial context and thus more information. Drawing our ideas from this, we propose two novel methods of expanding the receptive field on convolutional layers: atrous convolutions (LRFNet-A) and one-dimensional kernels (LRFNet-S). We have shown how atrous convolutions expand the receptive field by using spacing between the weights in the convolutional filters. We have also shown how one-dimensional filters inspired by kernel separability can expand the receptive field by use of large kernels.

We more specifically evaluated the performance of our proposed methods against a strong baseline through a large-scale design space exploration. Through this study, we have shown how the use of one-dimensional kernels provides a more relaxed tradeoff between the accuracy performance and parameters / runtime vs the severe tradeoff when using square kernels. One-dimensional kernels were also shown to be able to achieve better performance using the same or even less trainable parameters, thus having a smaller memory footprint. We also demonstrate the effectiveness of dilating square convolution kernels. Our empirical evaluations showed that increasing the dilation rate as well as using many different rates both improved SR network performance by increasing the receptive field while maintaining parameter count.

In general, our final results demonstrate how both techniques can be applied to high-scale super-

resolution networks to increase reconstruction accuracy while network depth and parameter count, and without major sacrifices in speed unlike current state-of-the-art methods. Our proposed networks also demonstrate substantial improvements in accuracy performance over current state-of-the-art methods in high-scale super-resolution.

5.2 Future Work

The most direct extension of this work is in the further design exploration and improvement of the proposed one-dimensional kernels to expand the receptive field. In the local residual blocks used in this work, there are two convolution operations. When using the baseline 3×3 kernel, a single residual block creates a receptive field of 5 in each direction (vertical and horizontal). However, with our proposed LRFNet-S we use one 9×1 kernel and one 1×9 kernel which should result in a final theoretical receptive field of 9 in each direction for that local residual block. This is almost double that of the 3×3 square kernel. While the one-dimensional kernels do improve the performance, such a largely expanded receptive field suggests that performance improvements can theoretically be even more substantial.

Thus, a promising direction for future work would be a further design space exploration of the use of one-dimensional kernels to study whether or not further performance improvements can be made. There may perhaps be a different design design scheme than that which was proposed in this work that achieves better performance. Such a study may even discover that there is a limit to which expanding the receptive field improves performance i.e too large of a receptive field may be detrimental. A part of this study may be introducing a boundary refinement module using square kernels similar to what was proposed with the Global Convolutional Network (GCN) [37]. This could potentially correct any (if any) performance draw backs from the exclusive use of one-dimensional kernels throughout the network residual blocks.

A more general direction to improving the performance of SISR models as a whole would be to focus on incorporating more high-level information into the models. As was discussed in Chapter 4 of the Experimental Results, all state-of-the-art models proposed thus far for SISR do not perform well on high-upscaling factors with images that have fine details and textures, since these are usually lost in the downsampling process. A direct optimization of a purely pixel-based loss won't be able to reconstruct such high-frequency components. Some works have proposed to use a feature-based loss function [46, 34]

based on the high-level features extracted by classification networks. Such SR networks do reconstruct sharp looking images but are not able to produce the same structures and textures as the original ground-truth. The actual structures and pixel values in the images produced by these models are often quite different from the ground-truth and thus the PSNR and SSIM scores are low. The design of a new model that balances the low-level (i.e low frequency components) pixel accuracy with the accuracy of the reconstructed high-level image structures and textures (i.e high-frequency) would be a strong contribution to addressing the SISR problem.

References

[1] Rafael C Gonzalez, Richard E Woods, et al. *Digital image processing*. Pearson, 2017.

[2] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. CVPR, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[8] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[9] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2017.

[11] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[12] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[13] Jan Allebach and Ping Wah Wong. Edge-directed interpolation. In *Image Processing, 1996. Proceedings., International Conference on*, volume 3, pages 707–710. IEEE, 1996.

[14] Xin Li and Michael T Orchard. New edge-directed interpolation. *IEEE transactions on image processing*, 10(10):1521–1527, 2001.

[15] Lei Zhang and Xiaolin Wu. An edge-guided image interpolation algorithm via directional filtering and data fusion. *IEEE transactions on Image Processing*, 15(8):2226–2238, 2006.

[16] Jianchao Yang, John Wright, Thomas Huang, and Yi Ma. Image super-resolution as sparse representation of raw image patches. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[17] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.

[18] Radu Timofte, Vincent De, and Luc Van Gool. Anchored neighborhood regression for fast example-based super-resolution. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1920–1927. IEEE, 2013.

[19] Radu Timofte, Vincent De Smet, and Luc Van Gool. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *Asian Conference on Computer Vision*, pages 111–126. Springer, 2014.

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

[21] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[26] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[27] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[28] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European Conference on Computer Vision*, pages 391–407. Springer, 2016.

[29] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[30] Wenzhe Shi, Jose Caballero, Ferenc Huszar, 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[32] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[33] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005.

[34] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[36] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[37] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[38] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee, et al. Ntire 2017 challenge on single image

super-resolution: Methods and results. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1110–1121. IEEE, 2017.

[39] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012.

[40] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010.

[41] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2011.

[42] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[43] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[44] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[45] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[46] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

Acronyms

CNN Convolutional Neural Network. 2, 5, 8–10, 12, 15, 21, 23, 55

DIV2K DIVerse 2K. vi, 29–31, 35–37

DRCN Deeply Recursive Convolutional Network. vii, ix, x, 14, 35, 37, 50–54

DRRN Deep Recursive Residual Network. vii–x, 17–19, 27, 35, 37, 50–54

ESPCN Efficient Sub-Pixel Convolutional Network. 15, 17

FSRCNN Fast Super-Resolution Convolutional Neural Network. vii, 15

HR High Resolution. 1, 2, 6, 7, 16–18

LapSRN Laplacian Super-Resolution Network. vii, 15, 16, 30, 35, 37

LR Low Resolution. 1, 2, 6, 7, 12

LRFNet Large Receptive Field Network. vi, 3, 19, 27–29, 32–37, 55, 56

MAE Mean Absolute Error. 16

MSE Mean Squared Error. 13, 14, 16, 30, 31, 36

NTIRE New Trends in Image Restoration and Enhancement. vi, 29, 32–36

PSNR Peak Signal-to-Noise Ratio. vi, ix, 3, 12, 14, 18, 19, 28, 30–44, 57

ReLU Rectified Linear Unit. viii, 8, 10, 12, 20, 25, 27, 32

ResNet Residual Network. 13, 17

RGB Red-Green-Blue. 13, 35

SISR Single Image Super-Resolution. 1, 2, 4, 7, 13, 15, 16, 19, 21, 25, 29–32, 34, 36, 56, 57

SR Super-Resolution. 1–3, 5, 13, 31, 57

SRCNN Super-Resolution Convolutional Neural Network. vii, 12–14, 35, 37

SRResNet Super-Resolution Residual Network. vii, viii, 16, 17, 19, 27

SSIM Structural Similarity index. vi, 3, 12–14, 18, 19, 28, 30–38, 57

VDSR Very Deep Super-Resolution network. vii, 13–15, 17, 35, 37