# FPGA-based Real-Time Super-Resolution System for Ultra High Definition Videos

Zhuolun He*§, Hanxian Huang*§, Ming Jiang†, Yuanchao Bai‡, and Guojie Luo*,
* *Center for Energy-efficient Computing and Applications*
† *Department of Information Sciences, School of Mathematical Sciences*
‡ *School of Electronics Engineering and Computer Science*
*Peking University, Beijing 100871, China*
*Email: {leonhe, gluo}@pku.edu.cn*

***Abstract*—The market benefits from a barrage of Ultra High Definition (Ultra-HD) displays, yet most extant cameras are barely equipped with Full-HD video capturing. In order to upgrade existing videos without extra storage costs, we propose an FPGA-based super-resolution system that enables real-time Ultra-HD upscaling in high quality. Our super-resolution system crops each frame into blocks, measures their total variation values, and dispatches them accordingly to a neural network or an interpolation module for upscaling. This approach balances the FPGA resource utilization, the attainable frame rate, and the image quality. Evaluations demonstrate that the proposed system achieves superior performance in both throughput and reconstruction quality, comparing to current approaches.**

***Keywords*-Field-Programmable Gate Arrays; Ultra High Definition; Super-Resolution; Real-time**

## I. Introduction

Ultra high definition (UHD) technology has been changing the entertainment industry significantly. However, UHD content is severely short of supply due to limited content creators or hard to access due to insufficient network bandwidth. Hence, it is highly in demand to upscale video content of conventional full high-definition (2K FHD) resolution of $1920\times1080$ into the 4K UHD version of $3940\times2160$.

Estimating a fine-resolution image/video from a coarse-resolution input is often referred to as super-resolution. This fundamentally important problem in image processing and computer vision has become particularly attractive as high-definition displays dominate the market. Previous works using interpolations [1], model-based methods [2]–[4], and example-based methods [5]–[11] will be elaborated in Section II-A. Furthermore, a variety of neural network solutions [8]–[10] achieved satisfying reconstruction quality. However, most of these CPU-based methods are far from reaching ideal performance as well as energy efficiency.

Given the huge storage expense of UHD content and inspired by the aforementioned state-of-the-art super-resolution techniques, we propose a super-resolution generation solution in real-time with FPGA in this work. Our work makes the following contributions:

1) It combines an accurate but complex neural network with a fast but naive interpolation algorithm. In this way, we generate outputs in both high speed and quality for large-size inputs.

§ Contributed equally.

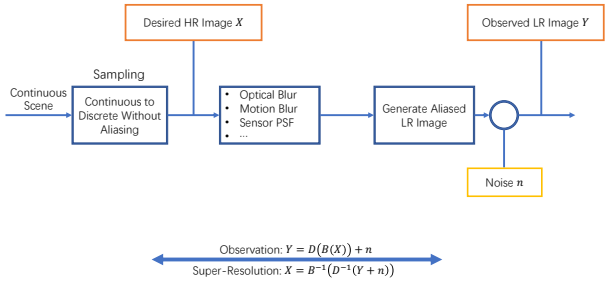


Figure 1. Observation Model that Relates LR Image to HR Image

2) We propose a quantitative model for analysis and optimization to balance the utilization of limited hardware resources, the attainable frame rate, and the visual performance.
3) Our super-resolution system generates a higher resolution video than reported in existing literature, namely $3940\times2160$ UHD videos from $1920\times1080$ FHD sources at a frame rate of approximately 30fps on an embedded FPGA board.

## II. Related Work

### A. Super-Resolution

Super-resolution has generated a wide spectrum of studies since the seminal work [12]. And we refer readers to [13] for a comprehensive literature review.

The most straightforward methods for super-resolution are those based on interpolations, including nearest-neighbor, bilinear, bicubic, and Lanczos algorithms [1]. These methods usually run fast and are easy to implement, but inevitably produce excessively blurry results [2].

Model-based methods [2], [3] aim to restore high-resolution scenes according to the observation model in Figure 1 and with priors (regularizations). Most of the works (e.g., [4]) assume known blur kernels and noise levels, but in reality they can be arbitrary [14].

Example-based approaches learn the mapping between low- and high-resolution patches. These approaches either exploit internal similarity of the same image [6], [7], or learn the mapping function from external exemplar pairs [5], [11]. It is worth noticing that deep learning techniques have been

successfully applied in super-resolution [8]–[10] and often achieve state-of-the-art restoration quality.

### B. FPGA-based Neural Network Accelerators

FPGA-based accelerators for neural networks are gaining popularity because of its higher energy-efficiency comparing to GPUs and shorter development cycles comparing to ASICs. Since convolution operations often take up a large proportion of the total operations in neural networks, most of the previous works focus on optimizing convolutions.

Many accelerators focus on improving the computational efficiency. They explore parallelism, computing sequences (pipelines), and computation-communication balance by loop optimization techniques like loop unrolling and loop tiling [15]–[18]. Theses techniques are analyzed in depth in [19].

Some efforts have also been put on reducing the computational demands through frequency domain acceleration [20], [21], binarized/tenarized networks [22]–[24], and network compression [25].

Other studies have put forward hardware abstractions [26], [27] and end-to-end automated frameworks [28], [29].

### C. Super-Resolution System on FPGA

Real-time super-resolution systems [30], [31] based on the iterative back projection algorithm are presented. It combines and slightly modifies a model-based super-resolution algorithm [32] that assumes identical blur between frames (for computational efficiency), and an iterative one [33] that uses L1-norm minimization (for robustness). Fixed-point precision is used, and a highly pipelined architecture is proposed for the real-time purpose. Szydzik et al. [34] reduces logic occupation when implementing the Non-Uniform Grid Projection Algorithm.

In [35] a learning-based super-resolution system is presented. It implements the A+ algorithm [36] using only a few line buffers (and without a frame buffer). The system consists of three pipelined stages, which are an interpolation stage for generating low-frequency parts, a mapping stage to select high-frequency patches by pre-trained regression functions, and a construction stage that enhances and overlaps the low-frequency image patches with high-frequency information. Noticing that the second stage handles massive computations and introduces long latency, the operation period in the second stage is doubled, and the system is designed with multiple clock domains.

In [37] a convolutional neural network for super-resolution based on FRCNN [8] is implemented on FPGA. Instead of enlarging the input beforehand, it applies horizontal and/or vertical flips to the network input images. This flip prevents the information decrease which occurs in the pre-enlargement process, enabling the network to utilize the most of its input image size [37].

## III. Super-Resolution Algorithm

### A. Overall Algorithm

For run-time limitation and resource constraints, we come up with a super-resolution algorithm that combines a neural network and an interpolation-based method.

Given a low-resolution (LR) image $\mathbf{X}$, we first crop the image into $N\times N$-pixel sub-images with a stride of $k$. For each sub-image, we calculate its importance index through a measurement function $\mathrm{M} : \mathbb{R}^{N\times N} \to \mathbb{R}$. The sub-images with high importance indices are upscaled using a neural network, while the rest are simply upscaled by interpolation. Finally, the upscaled sub-images are assembled into a high-resolution (HR) image $\mathbf{Y}$.

The pseudo code for super-resolution is listed in Algorithm 1.

**Algorithm 1** Overall Super-Resolution Algorithm

**Input:** LR image $\mathbf{X}$, upscaling factor $n$, threshold $T$
**Output:** HR image $\mathbf{Y}$

```
Crop X into sub-images {x} with a stride k
for all sub-image x do
    if M(x) ≥ T then
        y ← Upscale(x)
    else
        y ← CheapUpscale(x)
    end if
end for
Mosaic Y with upscaled sub-images {y}
```

The selection of sub-image stride $k$ will be discussed in Section IV-E.

### B. Total Variation-based Masking

In our practice, we adopt the total variation (TV) [38] as the masking measure M in Algorithm 1. Note that an anisotropic version of TV is employed for easier computation.

We use some of the notations in [39]. Let us consider an $N\times N$ image as a 2-dimensional matrix in $\mathcal{X}$, where $\mathcal{X}$ is the Euclidean space $\mathbb{R}^{N\times N}$. To define the discrete total variation, we introduce a discrete (linear) gradient operator $\nabla : \mathcal{X} \to \mathcal{X} \times \mathcal{X}$. If $x \in \mathcal{X}$, $\nabla x$ is a vector in $\mathcal{X} \times \mathcal{X}$ given by:

$$(\nabla x)_{i,j} = ((\nabla x)^V_{i,j}, (\nabla x)^H_{i,j}) \tag{1}$$

with

$$\begin{aligned} (\nabla x)^V_{i,j} &= \begin{cases} x_{i+1,j} - x_{i,j} & \text{if } i < N \\ 0 & \text{if } i = N \end{cases} \\ (\nabla x)^H_{i,j} &= \begin{cases} x_{i,j+1} - x_{i,j} & \text{if } j < N \\ 0 & \text{if } j = N \end{cases} \end{aligned} \tag{2}$$

for $i, j = 1, 2, \ldots, N$.

Then, the total variation of $x$ is defined by

$$J(x) = \sum_{1\le i,j\le N} ||(\nabla x)_{i,j}||_1 \tag{3}$$

with $||y||_1 := |y_1| + |y_2|$ for $y = (y_1, y_2) \in \mathbb{R}^2$.

We select TV as the masking method for the following several reasons:

1) TV value reveals the high-frequency intensity of an image block. High TV value comes with more high-frequency information, like edges and textures, which cannot be restored well by interpolation methods.
2) The distribution of TV value over natural image blocks is close to Rayleigh distribution. As a result, we can effortlessly sift a portion of blocks out by setting a reasonable threshold value. The Rayleigh-like distribution of gradient in images is also mentioned in previous studies [40].
3) TV value is easy to calculate. In Section IV-B, we propose a micro-architecture which computes TV of an image while reading in image pixels.

### C. Convolutional Neural Network for Super-Resolution

We adopt the hourglass-shaped convolutional neural network proposed by Chao Dong et al. [41], namely FSRCNN-s, which can learn an end-to-end mapping between the original LR and the target HR images with no pre-processing. We give a brief introduction to FSRCNN-s here.

*1) Neural Network Topology:* The same as in [41], we denote a convolution layer as $Conv(c_i, f_i, n_i)$ and a deconvolution layer as $DeConv(c_i, f_i, n_i)$, where the variables $c_i$, $f_i$, and $n_i$ represent the number of channels, the filter size, and the number of filters, respectively. FSRCNN-s can be decomposed into the following five stages (layers).

**Feature Extraction** $Conv(1, 5, 32)$ extracts 32 feature maps from the original LR image using filters of size $5\times5$.

**Shrinking** $Conv(32, 1, 5)$ reduces the LR feature dimension from 32 to 5 using filters of size $1\times1$.

**Mapping** $Conv(5, 3, 5)$ nonlinearly maps LR features onto HR features using filters of size $3\times3$.

**Expanding** $Conv(5, 1, 32)$ expands the HR feature dimension from 5 to 32 using filters of size $1\times1$.

**Deconvolution** $DeConv(32, 9, 1)$ upsamples and aggregates previous features using filters of size $9\times9$.

*2) Activation Function:* FSRCNN-s suggests the use of the Parametric Rectified Linear Unit (PReLU) after each convolution layer. The activation function is defined as

$$f(x_i) := \max(x_i, 0) + a_i \min(x_i, 0), \tag{4}$$

where $x_i$ is the input signal of the activation $f$ on the $i$-th channel, and $a_i$ is the coefficient of the negative part. Unlike for ReLU where the parameter $a_i$ is fixed to be zero, it is learnable for PReLU.

*3) Cost Function:* FSRCNN-s adopts the mean square error (MSE) as the cost function. The optimization objective is represented as

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} ||F(Y_s^i; \theta) - X^i||_2^2, \tag{5}$$

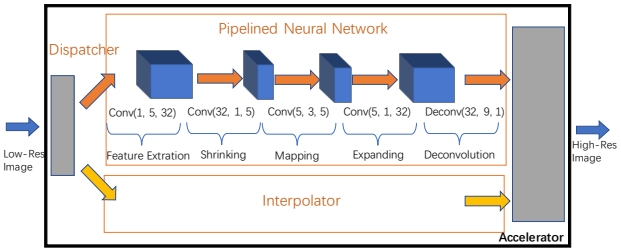


Figure 2. An overview of the proposed system. Grey boxes indicate the input and output frame buffers. The dispatcher is integrated into the input buffer.

where $Y_s^i$ and $X_i$ are the $i$-th LR and HR sub-image pair in the training data, and $F(Y_s^i; \theta)$ is the network output for $Y_s^i$ with parameters $\theta$. All parameters are optimized using stochastic gradient descent with the standard backpropagation.

## IV. Implementation

### A. System Overview

Figure 2 illustrates the overall system design. It consists of three main parts:

- Dispatcher: It calculates the TV value of each block according to the equations mentioned in Section III-B. Then it dispatches the blocks whose TV values are greater than the threshold, which we have decided in advance, to the neural network, while others to the interpolation module.
- Pipelined Neural Network: FSRCNN-s is implemented in a fully pipelined structure where each network layer is a pipeline stage. The number of multipliers in each stage is also configured to achieve well-balanced throughputs of the stages.
- Interpolator: The simple and fast interpolator handles the blocks whose TV values are less than the threshold. The bilinear algorithm, which can upscale images with low costs and fine performance, is selected.

Finally, the output blocks from the network or interpolator are assembled to generate the final output high-resolution image.

### B. Micro-architecture for Stencil Masking

In our TV-based masking, the vertical and horizontal gradients of a pixel rely on the pixel itself, as well as the one below and the one on the right-hand side, respectively. The computation pattern can be considered as stencil computation, i.e., each point is updated (iteratively) as a function of its value and values at its neighboring elements. Figure 3 depicts the access pattern of the computation. To calculate gradients at $x[offset]$, we have to access pixels $x[offset]$, $x[right]$, and $x[down]$, which are colored dark blue in the figure.

When the iterative computation is fully pipelined, the computation kernel needs to load multiple elements from

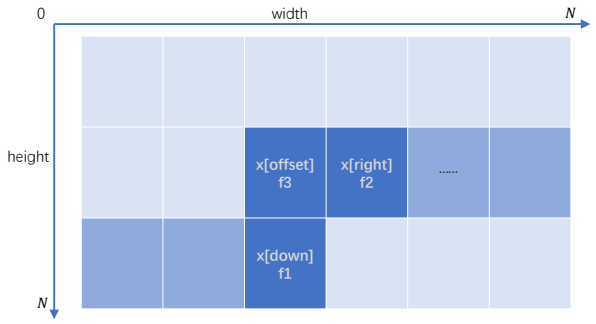


Figure 3. Stencil access pattern of TV computation. Gradients at *offset* rely on the dark blue pixels.

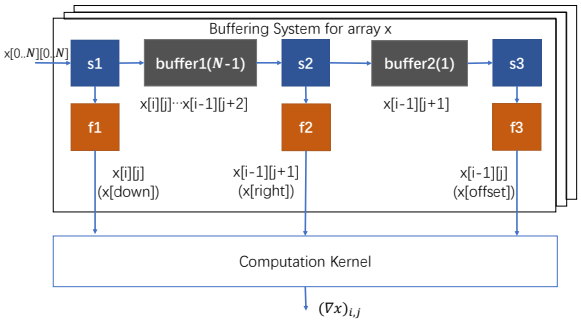


Figure 4. Micro-architecture for TV calculation. Grey, blue, and orange boxes denote buffers, data path splitters and filters, respectively.

Table I
FILLING PROCESS OF BUFFERING SYSTEM. IN FILTER STATUS, D, S, AND F STAND FOR DISCARDING, STALL AND FORWARDING, RESPECTIVELY.

| clock cycle | data in stream | filter status | | | FIFO status (# of data) | |
|---|---|---|---|---|---|---|
| | | filter 1 | filter 2 | filter 3 | FIFO 1 | FIFO 2 |
| 1 | x[0][0] | d | d | f→s | 0 | 0 |
| 2 | x[0][1] | d | f→s | s | 0 | 1 |
| N | x[0][N-1] | d | s | s | N-2 | 1 |
| N+1 | x[1][0] | d→f | s→f | s→f | N-1 | 1 |
| N+2-... | x[1][1]-... | f | f | f | N-1 | 1 |

Table II
NOTATIONS FOR EXPLAINING NEURAL NETWORK IMPLEMENTATION

| Notations | Meanings |
|---|---|
| $c_i$ | Number of input channels of the $i$-th layer |
| $f_i$ | Filter dimension of the $i$-th layer |
| $n_i$ | Number of output channels of the $i$-th layer |
| Conv($c_i$, $f_i$, $n_i$) | The $i$-th convolution layer |
| $N_i$ | Input feature map dimension of the $i$-th layer |
| $k$ | Sub-image stride |
| #Conv | Number of convolution layers |
| $S_{\text{HD}}$ | Size of a Full-HD image |
| $s$ | Upscaling factor |
| $Fr$ | Frame rate (frames/s, fps) |
| $\mathcal{B}$ | I/O bandwidth (bits/s, bps) |
| $\mathcal{C}$ | BRAM capacity (bits) |
| $\mathcal{WL}$ | Word length (bits) |

one array in a single clock cycle, so memory partition is necessary to avoid contention on memory ports. Though uniform memory partition strategies are explored in recent publications, e.g., [42], [43], we adopt the micro-architecture proposed by [44] to decouple the stencil access pattern from the computation.

The micro-architecture, as illustrated in Figure 4, mainly contains buffering systems equipped with memory controllers and data interconnects. There is no data reuse opportunity among different arrays, so the buffering systems are independent of each other. In each buffering system, FIFOs provide storage the same as conventional data reuse buffers do, while data path splitters and filters between FIFOs work as memory controllers and data interconnects. Each buffering system receives a single data stream without additional external memory access. Before the computation starts, the controllers first read-in data and fill up the FIFOs for $N$ cycles. Then in every clock cycle, the filters send the required data to the computation kernel, the kernel consumes all data to generate one output, and the controllers move all the buffered data forward. In this way, the buffering systems keep proceeding until the end of the iteration domain. Table I shows the filling process of the buffering system.

### C. Neural Network Implementation

To increase system throughput, we organize the whole neural network as a pipelined structure, with each network layer as a pipeline stage. All the feature maps and weights, as well as bias vectors and PReLU parameters, are all stored in BRAM. We can keep the data on chip mainly because of 1) the small size neural network and 2) our blocking algorithm which leads to small feature maps. Notations used in the following sections are provided in Table II.

*1) Convolution Layers:* For each convolutional layer $Conv(c_i, f_i, n_i)$, there are $c_i \times n_i$ filters of size $f_i \times f_i$, generating $n_i$ outputs. In our implementation, there will be $c_i \times n_i$ processing elements (PEs) computing in parallel, i.e., one PE per filter. There are three main steps during processing:

**Input** An $f_i \times f_i$ sliding window on each input feature map generates an input vector of $f_i^2$ elements.

**Compute** Corresponding PEs calculate the inner products of the input vector and the filter.

**Output** Partial sums are added up and stored in the target pixels.

The three steps are executed in a pipelining fashion. Figure 5 is a diagram of the Convolution layer architecture.

*2) Deconvolution Layer:* Deconvolution in this neural network can be regarded as a structurally inverse process of convolution. A deconvolution layer $DeConv(c_i, f_i, n_i)$ upsamples and aggregates the previous $c_i$ feature maps with $c_i \times n_i$ deconvolution filters of size $f_i \times f_i$. On account of the memory ports limitation and the reuse of intermediate data, sliding windows are also applied in the deconvolution layer. A sliding window holds on the partial results and updates them lately. This layer also has a three-staged pipeline:

**Input** Input pixels are obtained from the output feature maps of the last convolution layer.

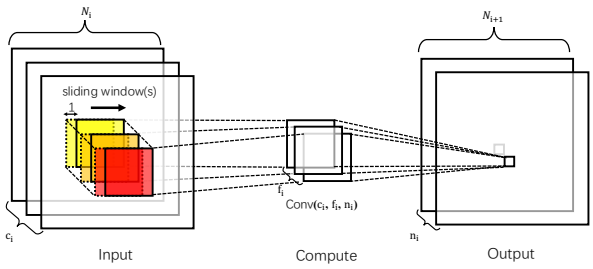


Figure 5. Convolution layer architecture. $f_i \times f_i$ windows are sliding across the input feature maps.

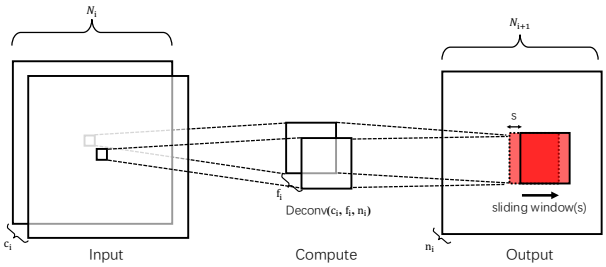


Figure 6. Deconvolution layer architecture. $f_i \times f_i$ windows are sliding across the output feature maps.

**Compute** Output pixels are calculated with input pixels and filters.

**Output** A sliding window updates $s$ columns on the target feature map each time. Note that the rest $f_i - s$ columns are kept in the window for further reuse, and the new $s$ column pixels are initialized to zeroes.

Figure 6 depicts the deconvolution layer architecture.

*3) Pipeline Balancing:* We also balance the whole pipeline through resource allocation. In a convolution stage $Conv(c_i, f_i, n_i)$, there are $c_i \times n_i \times f_i^2 \times N_{i+1}^2$ multiplications, note again $N_{i+1}$ is the dimension of an output feature map of this layer. To balance the throughput of each stage, we should allocate the number of multipliers (DSPs) in each stage proportional to the number of multiplications in the stage, while keeping the overall utilization from exceeding the total amount of available DSPs. Table III shows multiplier allocations of each layer and relevant data. We obtain the ideal number[1] of DSPs (ideal #DSP) of each layer by assigning multipliers proportionally to the number of multiplications (#Mult.) of them. Then the ideal IIs are computed accordingly. We set II of each layer manually (so that the necessary performance is achieved), and the required number of DSPs (#DSP) to achieve such II is obtained.

[1]Rounded down to the nearest integer.

Table III
MULTIPLIER ALLOCATIONS AND RELEVANT DATA

| Layer | $c_i$ | $f_i$ | $n_i$ | $N_i$ | #Mult. | ideal | | alloc. | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | #DSP | II | #DSP | II |
| Extraction | 1 | 5 | 32 | 36 | 819200 | 201 | 4076 | 200 | 4096 |
| Shrinking | 32 | 1 | 5 | 32 | 163840 | 40 | 4096 | 32 | 4096 |
| Mapping | 5 | 3 | 5 | 32 | 202500 | 50 | 4050 | 45 | 4500 |
| Expanding | 5 | 1 | 32 | 30 | 144000 | 35 | 4115 | 32 | 4500 |
| Deconvolution | 32 | 9 | 1 | 30 | 2332800 | 573 | 4072 | 519 | 4500 |
| Overall | - | - | - | - | 3662340 | 899 | 4115 | 828 | 4500 |
| Available (ZC706) | - | - | - | - | - | 900 | - | 900 | - |

### D. Interpolator

We use the bilinear interpolation as the alternation of the neural network method (i.e., CheapUpscale in Algorithm 1). We observe that from the output perspective, the bilinear interpolation is very similar to the deconvolution process described in Section IV-C2. For example, in our case of $2\times$ upscaling, an input pixel $\mathbf{X}_{i,j}$ spreads its value to a $3\times3$ window centered at $\mathbf{Y}_{2i,2j}$ of the output with the "deconvolution kernel": $\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$. Similarly, this structure enables the use of sliding window to avoid massive load/store addressing. The observation also accounts for why the deconvolution could be adopted for upscaling instead of pre-enlargement.

### E. Sub-image Stride Selection

Sub-image stride $k$ affects the system performance in both efficiency and quality, and thus should be contemplated carefully. To generate valid convolution results of an $f_i \times f_i$ filter, we should enlarge the input feature map with extra border of size $\frac{f_i-1}{2}$. Therefore, to have a valid $k \times k$ output through all convolution layers, we should have:

$$N_i \equiv k + \sum_{i}^{\#\text{Conv}} (f_i - 1). \tag{6}$$

There are several constraints on the stride $k$ that should be considered:

1) I/O bandwidth constraint. Because each sub-image has to be enlarged with extra pixels for convolution, small stride comes with a large border-to-block ratio, which results in inefficient utilization of I/O bandwidth. To satisfy I/O bandwidth constraint, we have:

$$(\frac{N_1}{k})^2 \times S_{\text{HD}} \times Fr \times \mathcal{WL} < \mathcal{B} \tag{7}$$

2) Storage capacity constraint. Large stride comes with large size of feature maps, which makes storing all feature maps on chip impossible. To satisfy storage

capacity constraint, we have[2]:

$$\left(\sum_{i=1}^{\#\text{Conv}+1}(N_i^2 \times c_i) + (k \times s)^2\right) \times \mathcal{WL} \times 2 < \mathcal{C} \quad (8)$$

3) Upscaling performance constraint. The empirical relation between this constraint and the design parameters will be presented in Section V-B3.

By solving simultaneous equation and inequalities (6)-(8) using corresponding data, we can obtain that $2 \leq k \leq 57$.

## V. Experimental Evaluation

### A. Experimental Setup

*1) Hardware Platform:* We test our system on Xilinx ZC706 Evaluation Board featuring the XC7Z045 FFG900 -2 AP SoC, which has 350 Logic Cells, 19.1Mb Block RAM, 900 DSP Slices, 360 Maximum I/O Pins, and 16 Maximum Transceiver Count. We set its working frequency at 100 MHz and use 16-bit fixed data type.

*2) Software Setup:* The design is implemented by Xilinx SDSoC Development Environment v2016.3.

*3) Dataset:* We use the ultra-high resolution 4K video sequences from SJTU Media Lab [45], which is of YUV $4:2:0$ color sampling, 8 bits per sample, and a frame rate of 30 fps. The original 4K images are used as the ground truth, and the 2K LR images are obtained by down-sampling. Our super-resolution system generates the reconstructed 4K HR images.

*4) Metric:* To evaluate our system performance, we use the Peak Signal-to-Noise Ratios (PSNR) and Structural SIMilarity (SSIM) [46], both of which are widely-used metrics for quantitatively evaluating image resolution quality. These metrics measure the difference between reconstructed HR images and the corresponding ground truth.

The calculation of PSNR using the equation as follows:

$$PSNR = 10\log_{10}\frac{R^2}{MSE}, \quad (9)$$

where $R$ is the maximum fluctuation in the input image data type. For example, our images are encoded with the 8-bit unsigned integer data type, thus the R is 255. MSE represents the mean square error, which is calculated as:

$$MSE = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W}(I_1(i,j) - I_2(i,j))^2, \quad (10)$$

where $H$ and $W$ are the height and the width of the input images, and $I_1(i,j)$ and $I_2(i,j)$ are the corresponding pixel values of the two images.

The SSIM quality assessment index is based on the computation of three terms, namely the luminance term, the contrast term, and the structural term. The overall index is a multiplicative combination of these three terms:

$$SSIM(\mathbf{X},\mathbf{Y}) = [l(\mathbf{X},\mathbf{Y})]^\alpha \cdot [c(\mathbf{X},\mathbf{Y})]^\beta \cdot [s(\mathbf{X},\mathbf{Y})]^\gamma \quad (11)$$

[2]The storage requirement has to be doubled when using ping-pong buffers in the pipeline.

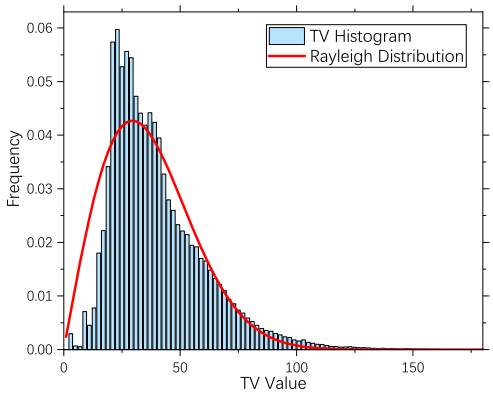


Figure 7. TV Distribution of the SJTU 4K Video Sequence Dataset. The distribution follows the Raylaigh distribution (the red curve). High TV value indicates more high-frequent information, which should be reconstructed carefully.

where

$$\begin{aligned} l(\mathbf{X},\mathbf{Y}) &= \frac{2\mu_\mathbf{X}\mu_\mathbf{Y} + C_1}{\mu_\mathbf{X}^2 + \mu_\mathbf{Y}^2 + C_1} \\ c(\mathbf{X},\mathbf{Y}) &= \frac{2\sigma_\mathbf{X}\sigma_\mathbf{Y} + C_2}{\sigma_\mathbf{X}^2 + \sigma_\mathbf{Y}^2 + C_2} \\ s(\mathbf{X},\mathbf{Y}) &= \frac{\sigma_{\mathbf{XY}} + C_3}{\sigma_\mathbf{X}\sigma_\mathbf{Y} + C_3} \end{aligned} \quad (12)$$

where $\mu_\mathbf{X}$, $\mu_\mathbf{Y}$, $\sigma_\mathbf{X}$, $\sigma_\mathbf{Y}$, and $\sigma_{\mathbf{XY}}$ are the local means, standard deviations, and cross-covariance for images $\mathbf{X}$ and $\mathbf{Y}$. For the other constants, we often set $\alpha = \beta = \gamma = 1$ for the exponents, and $C_1 = (K_1 \times L)^2$, $C_2 = (K_2 \times L)^2$, $C_3 = C_2/2$ with $K_1 = 0.01$, $K_2 = 0.03$, and $L = 255$.

It is worth noticing that the human eye is most sensitive to luma information, and thus we only separately process and measure the intensity channel in our YCbCr images.

### B. Analysis of Design Options

We carry out multiple experiments to explore the relationship between the performance with varying TV threshold values and block sizes. The two factors are critical since different TV thresholds change the workload of each processing module and thus, influence the performance (in both speed and quality). At the same time, block sizes determine resource utilization of the implementation. These experiments help us determine design parameters in further system implementation on FPGA.

*1) TV Statistics:* TV values of sub-image blocks vary from one to another and could relate to the visual properties of the original image. Statistics on TV values of our dataset is illustrated in Figure 7. We observe that the TV distribution follows the Raylaigh distribution. In our implementation, we choose 50 as the base value, above which the proportion is 25.3% according to the statistics.

*2) Different TV Thresholds with the Same Block Size:* In this group of experiments, we choose 30 as the block size and set TV value threshold from 30 to 70 with a stride of

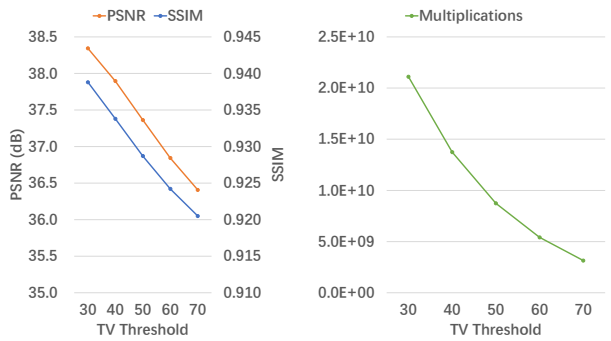


Figure 8. Evaluation of different TV thresholds with the same block size. Performance and the number of multiplications drop as TV threshold increases.

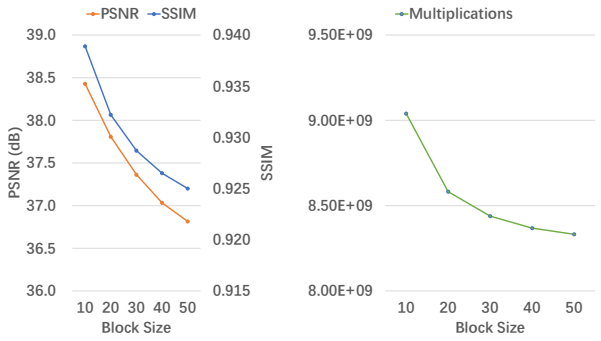


Figure 9. Evaluation of different block sizes with corresponding TV value thresholds. Performance and the number of multiplications drop as block size increases.

10. We test the average values of each block to evaluate the performance, as shown in Figure 8. We can obtain that the higher the threshold value, the higher performance. Evidently, when a higher threshold value is chosen, more blocks will be processed with the neural network, which often leads to better results.

*3) Different Block Sizes with Corresponding TV Value Thresholds:* In this group of experiments, we increase block size from 10 to 50 with a stride of 10 and set corresponding thresholds according to block areas. We use the block size of 30 and the TV threshold of 50 as the control group. The results are shown in Figure 9. From the figure we can see that selecting blocks in a finer-grained gains higher reconstruction quality. However, this is at the cost of higher computation complexity.

*4) Overall Comparisons:* In this group of experiments, we compare six solutions with different configurations in Table IV. Considering both preprocessing method (blocking/none) and upscaling method (neural network/interpolation), we test all four possible combinations. The fifth and the sixth solutions both use blocking and mixed upscaling methods, where 25.3% blocks are up-scaled by the neural network according to the analysis in Section V-B1, and the other are up-scaled by interpolation. And the difference is that the fifth solution selects upscaling method for each block randomly, while the sixth solution uses total variation threshold for dispatching. Figure 10 shows the example

Table IV
OVERALL COMPARISONS OF DIFFERENT CONFIGURATIONS

| No. | Preprocessing | Upscaling | #Mult. | PSNR (dB) | SSIM |
|---|---|---|---|---|---|
| 1 | None | Interpolation | $6.6\times10^7$ | 35.51 | 0.9138 |
| 2 | None | Neural Network | $8.2\times10^9$ | 38.55 | 0.9421 |
| 3 | Blocking | Interpolation | $6.6\times10^7$ | 35.51 | 0.9138 |
| 4 | Blocking | Neural Network | $8.4\times10^9$ | 38.55 | 0.9420 |
| 5 | Blocking | Mixed-Random | $2.2\times10^9$ | 36.10 | 0.9211 |
| 6 | Blocking | Mixed-TV | $2.2\times10^9$ | 37.36 | 0.9287 |

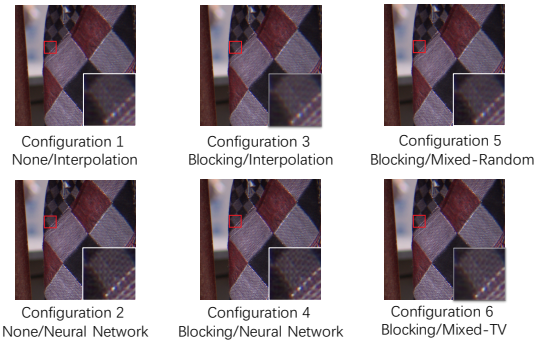


Figure 10. Example outputs of different configurations. Configuration 6, which is adopted in our system, shows better reconstruction quality than interpolations and mixed-random method (configuration 1, 3, and 5), and costs only $1/4$ multiplications compared with neural network methods (configuration 2, 4).

outputs of the six configurations.

We can easily conclude that:

1) The neural network method shows significantly better quality ($+3.04$ dB) than the interpolation algorithm, at the cost of two orders of magnitude more multiplications.
2) Cropping image into small blocks with proper padding achieves nearly the same quality as un-cropping.
3) Dispatching blocks according to the TV threshold works better ($+1.26$ dB) than random dispatching.
4) Mixed-TV method saves about 75% cost of multiplications with acceptable quality degradation ($-1.19$ dB) compared with the neural network method.

### C. Overall System Performance

For the super-resolution from Full-HD $1920\times1080$ inputs to Ultra-HD $3940\times2160$ outputs, our system can achieve average frame rates of 23.9fps, 29.3fps, and 31.7fps with 1, 2, and 3 interpolators, respectively. Resource utilization of each component is listed in Table V.

## VI. CONCLUSION AND FUTURE WORK

Inspired by the existing super-resolution works and techniques, we proposed a real-time UHD super-resolution solution based on FPGA accelerator. In our solution, each input frame is cropped into blocks. Then each block is dispatched according to its total variation value and finally up-scaled utilizing either a neural network or an interpolation module. We carry out some pre-experiments to find a proper block

Table V
RESOURCE UTILIZATION OF SYSTEM COMPONENTS ON A XILINX ZC706 EVALUATION BOARD

| Component | BRAM | DSP | FF | LUT |
|---|---|---|---|---|
| Dispatcher | 1 | 2 | 618 | 1138 |
| Neural Network | 178 | 844 | 63149 | 98439 |
| Interpolator | 0 | 10 | 1414 | 3076 |
| Total | 327 | 858 | 66261 | 103714 |
| Available | 1090 | 900 | 437200 | 218600 |
| Utilization (%) | 30 | 95 | 15 | 47 |

size and total variation threshold. Our solution is a trade-off between reconstruction quality and run-time efficiency to achieve satisfying performance.

It is possible to further accelerate our design using other techniques, e.g., to accelerate convolutions with Winograds minimal filtering theory [47]. Resolving the computational challenges of a generalized super-resolution system that generates more frames (e.g., from 60fps to 120fps) and more colors (e.g., from 8-bit RGB pixels to 10-bit ones) is another appealing research direction.

## ACKNOWLEDGMENT

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61520106004.

## REFERENCES

[1] K. Turkowski, "Filters for common resampling-tasks," *Graphics Gems*, pp. 147–165, 1990.
[2] Q. Shan, Z. Li, J. Jia *et al.*, "Fast image/video upsampling," *ACM Trans. Graphics*, vol. 27, no. 5, p. 153, 2008.
[3] C.-T. Shen, H.-H. Liu, M.-H. Yang *et al.*, "Viewing-distance aware super-resolution for high-definition display," *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 403–418, 2015.
[4] Y. Bai, H. Jia, X. Xie *et al.*, "A fast super-resolution method based on sparsity properties," *Visual Communications and Image Processing (VCIP)*, 2015.
[5] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *Int'l Jrnl. of Computer Vision*, vol. 40, no. 1, pp. 25–47, 2000.
[6] D. Glasner, S. Bagon, and M. Irani, "Super-resolution from a single image," *Int'l Conf. on Computer Vision (ICCV)*, 2009.
[7] J. Yang, Z. Lin, and S. Cohen, "Fast image super-resolution based on in-place example regression," *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.
[8] C. Dong, C. C. Loy, K. He *et al.*, "Learning a deep convolutional network for image super-resolution," *European Conf. on Computer Vision (ECCV)*, 2014.
[9] W. Shi, J. Caballero, F. Huszár *et al.*, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
[10] C. Ledig, L. Theis, F. Huszár *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," *arXiv preprint arXiv:1609.04802*, 2016.
[11] Y. Romano, J. Isidoro, and P. Milanfar, "RAISR: rapid and accurate image super resolution," *IEEE Trans. Comput. Imag.*, vol. 3, no. 1, pp. 110–125, 2017.
[12] R. Y. Tsai and T. S. Huang, "Multiframe image restoration and registration," *Adv. Comput. Vis. Image Process.*, vol. 1, no. 2, pp. 317–339, 1984.
[13] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: a technical overview," *IEEE Signal Process. Mag.*, vol. 20, no. 3, pp. 21–36, 2003.
[14] C. Liu and D. Sun, "A Bayesian approach to adaptive video super resolution," *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.
[15] C. Zhang, P. Li, G. Sun *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2015.
[16] N. Suda, V. Chandra, G. Dasika *et al.*, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016.
[17] J. Qiu, J. Wang, S. Yao *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016.
[18] Y. Shen, M. Ferdman, and P. Milder, "Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer," *Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2017.
[19] Y. Ma, Y. Cao, S. Vrudhula *et al.*, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.
[20] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.
[21] L. Lu, Y. Liang, Q. Xiao *et al.*, "Evaluating fast algorithms for convolutional neural networks on FPGAs," *Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2017.
[22] R. Zhao, W. Song, W. Zhang *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs." *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.
[23] Y. Umuroglu, N. J. Fraser, G. Gambardella *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.
[24] Y. Li, Z. Liu, K. Xu *et al.*, "A 7.663-TOPS 8.2-W energy-efficient FPGA accelerator for binary convolutional neural networks," *arXiv preprint arXiv:1702.06392*, 2017.
[25] S. Han, J. Kang, H. Mao *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA." *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.
[26] C. Zhang, Z. Fang, P. Zhou *et al.*, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2016.
[27] H. Sharma, J. Park, D. Mahajan *et al.*, "From high-level deep neural models to FPGAs," *Int'l Symp. on Microarchitecture (MICRO)*, 2016.
[28] X. Wei, C. H. Yu, P. Zhang *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," *Design Automation Conf. (DAC)*, 2017.
[29] Y. Guan, H. Liang, N. Xu *et al.*, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," *Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2017.
[30] O. Bowen and C.-S. Bouganis, "Real-time image super resolution using an FPGA," *Int'l Conf. on Field-Programmable Logic and Applications (FPL)*, 2008.
[31] M. Angelopoulou, C.-S. Bouganis, P. Cheung *et al.*, "FPGA-based real-time super-resolution on an adaptive image sensor," *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 125–136, 2008.
[32] M. Elad and Y. Hel-Or, "A fast super-resolution reconstruction algorithm for pure translational motion and common space-invariant blur," *IEEE Trans. Image Process.*, vol. 10, no. 8, pp. 1187–1193, 2001.
[33] S. Farsiu, M. D. Robinson, M. Elad *et al.*, "Fast and robust multiframe super resolution," *IEEE Trans. Image Process.*, vol. 13, no. 10, pp. 1327–1344, 2004.
[34] T. Szydzik, G. M. Callico, and A. Nunez, "Efficient FPGA implementation of a high-quality super-resolution algorithm with real-time performance," *IEEE Trans. Consum. Electron.*, vol. 57, no. 2, 2011.
[35] M.-C. Yang, K.-L. Liu, and S.-Y. Chien, "A real-time FHD learning-based super-resolution system without a frame buffer," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 12, pp. 1407–1411, 2017.
[36] R. Timofte, V. De Smet, and L. Van Gool, "A+: Adjusted anchored neighborhood regression for fast super-resolution," *Asian Conf. on Computer Vision (ACCV)*, 2014.
[37] T. Manabe, Y. Shibata, and K. Oguri, "FPGA implementation of a real-time super-resolution system using a convolutional neural network," *Int'l Conf. on Field-Programmable Technology (FPT)*, 2016.
[38] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.
[39] A. Chambolle, "An algorithm for total variation minimization and applications," *Jrnl. of Math. Imag. Vis.*, vol. 20, no. 1, pp. 89–97, 2004.
[40] D. L. Ruderman, "The statistics of natural images," *Network: Computation in Neural Systems*, vol. 5, no. 4, pp. 517–548, 1994.
[41] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," *European Conf. on Computer Vision (ECCV)*, 2016.
[42] P. Li, Y. Wang, P. Zhang *et al.*, "Memory partitioning and scheduling co-optimization in behavioral synthesis," *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2012.
[43] Y. Wang, P. Li, P. Zhang *et al.*, "Memory partitioning for multidimensional arrays in high-level synthesis," *Design Automation Conf. (DAC)*, 2013.
[44] J. Cong, P. Li, B. Xiao *et al.*, "An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers," *Design Automation Conf. (DAC)*, 2014.
[45] L. Song, X. Tang, W. Zhang *et al.*, "The SJTU 4K video sequence dataset," in *Int'l Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, 2013, pp. 34–35.
[46] Z. Wang, A. C. Bovik, H. R. Sheikh *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.
[47] S. Winograd, "Arithmetic complexity of computations," *Society for Industrial & Applied Mathematics*, vol. 43, no. 2, pp. 625–633, 1980.