Image Super-Resolution Via Convolutional Neural Networks

Single image super-resolution (SISR) is a problem that aims to obtain a high-resolution (HR) output from its corresponding low-resolution (LR) input image.

Detailed Report on SRCNN for Image Super-Resolution

1. Introduction

Super-Resolution (SR) is a technique in image processing that aims to reconstruct a high-resolution (HR) image from a low-resolution (LR) version. This technique is crucial in fields such as medical imaging, satellite imagery, and digital photography where high-resolution images provide better details for analysis and decision-making. Traditional SR methods like interpolation often result in blurred images due to their inability to recover high-frequency details.

Deep Learning in SR: The advent of deep learning has transformed SR tasks, enabling the reconstruction of high-quality images. Among various models, the Super-Resolution Convolutional Neural Network (SRCNN) stands out for its simplicity and effectiveness.

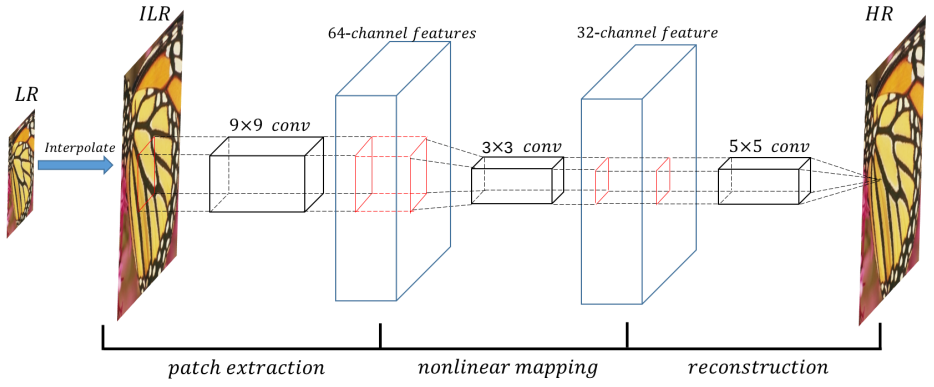
2. SRCNN Overview

SRCNN: SRCNN, introduced by Dong et al. in 2014, was one of the first deep learning models designed for image super-resolution. It demonstrated significant improvements over traditional methods by learning the mapping between LR and HR images through a deep convolutional network.

Historical Context: The development of SRCNN marked a pivotal point in SR research, leading to numerous advancements in the field. It showed that even a simple deep network could surpass complex traditional algorithms.

3. Detailed Architecture of SRCNN

Network Architecture: SRCNN consists of three convolutional layers, each designed to perform specific tasks in the SR process:



1. **Layer 1 (Patch Extraction and Representation)**:
   - **Input**: Low-resolution image.
   - **Operation**: Convolution with 64 filters of size 9×9, followed by ReLU activation.
   - **Purpose**: Extracts feature maps from the input image.
2. **Layer 2 (Non-linear Mapping)**:
   - **Input**: Output of Layer 1.

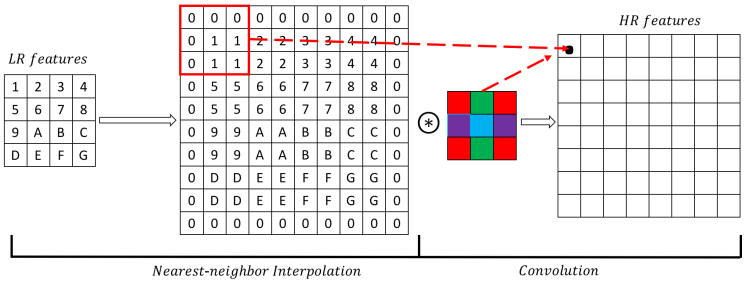
- **Operation**: Convolution with 32 filters of size 3×3, followed by ReLU activation.
- **Purpose**: Maps the extracted features to a high-resolution feature space.

3. **Layer 3 (Reconstruction)**:
   - **Input**: Output of Layer 2.
   - **Operation**: Convolution with 1 filter of size 5×5 to reconstruct the high-resolution image.
   - **Purpose**: Combines the high-resolution features to produce the final HR image.

Pooling and Padding:

- **Padding**: Zero-padding is used to maintain the spatial dimensions of the image after each convolution operation.
- **Pooling**: SRCNN does not use pooling layers to avoid losing spatial information, which is crucial for SR tasks.

For interpolation, we use Nearest-neighbor interpolation and then convolution



Filter Selection and Number of Layers: The three-layer architecture with specific filter sizes was chosen to balance computational efficiency and performance. The 9×9 filters in the first layer effectively capture local image structures, the 3×3 filters in the second layer map these structures to a higher-dimensional space, and the 5×5 filter in the final layer integrates the high-resolution details.

I will be loading pre-trained weights for the SRCNN. These weights can be found at the following GitHub page: https://github.com/MarkPrecursor/SRCNN-keras

---

4. Mathematical Formulation

Convolution Operation: In a convolution operation, a filter (or kernel) slides over the input image, computing the dot product between the filter weights and the input values. Mathematically, this can be represented as:

$$\text{Output}(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{n} \text{Input}(i+m, j+n) \cdot \text{Filter}(m,n)$$

where (i , j) are the spatial coordinates, and k is the filter size.

ReLU Activation: The Rectified Linear Unit (ReLU) activation function is defined as:
ReLU(x) = max(0, x)
This means that ReLU(x) returns 0 if x is less than 0, otherwise, it returns x. This function introduces non-linearity into the model, enabling it to learn complex mappings between the low-resolution and high-resolution images.

---

5. Implementation

Code Implementation: The implementation of SRCNN can be done using popular deep learning frameworks like TensorFlow or PyTorch. The detailed implementation is available on the GitHub repository:

Google Colab

https://colab.research.google.com/drive/17BQye4YmFG0Sxs6335FsKYrLJPrdE8Rp?usp=sharing

Training and Testing:

- **Dataset**: The training datasets used are Set5 and Set14, which are common benchmarks for SR tasks. These datasets can be accessed and processed using the MATLAB code provided in the GitHub repository: http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html
- **Training Procedure**:
  - **Data Augmentation**: Techniques like rotation, flipping, and scaling are used to increase the diversity of the training data.
  - **Learning Rate**: A learning rate scheduler is often employed to reduce the learning rate as training progresses. Learning rate used is 0.0003
  - **Evaluation Metrics**: The model's performance is evaluated using metrics such as Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM).

A) Mean Squared Error (MSE)

Definition: MSE measures the average of the squares of the errors—that is, the average squared difference between the estimated values (SR image) and the actual value (HR image). It is a straightforward metric used to quantify the difference between the original and the super-resolved image.

Mathematical Equation:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i,j) - K(i,j))^2$$

B) Peak Signal-to-Noise Ratio (PSNR)

Definition: PSNR is a measure of the peak error between the original high-resolution (HR) image and the super-resolved (SR) image. It is widely used because it is simple and has a clear physical meaning.

Mathematical Equation:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right)$$

Where:

- MAXI is the maximum possible pixel value of the image. For an 8-bit image, this value is 255.
- MSE is the Mean Squared Error between the HR image and the SR image.

C) Structural Similarity Index Measure (SSIM)

Definition: SSIM is a perceptual metric that quantifies the similarity between two images. It considers changes in structural information, luminance, and contrast. SSIM values range from -1 to 1, where 1 indicates perfect similarity.

Mathematical Equation:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where:

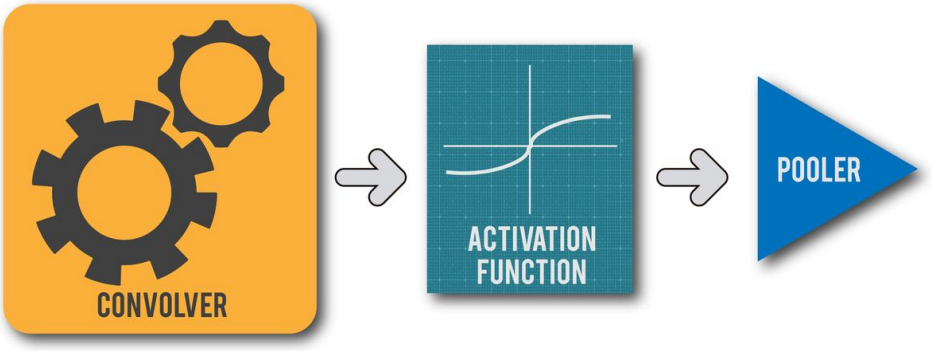
- x and y are the two images being compared.
- μ_x and μ_y are the average of x and y, respectively.
- σ_x^2 and σ_y^2 are the variances of x and y, respectively.
- σ_{xy} is the covariance of x and y.
- C_1 and C_2 are constants to stabilize the division with weak denominator.

6. Implementation on FPGA

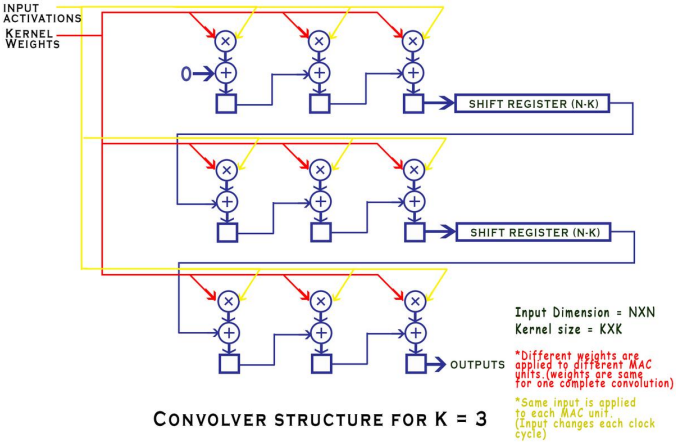
Overview of FPGA: Field-Programmable Gate Arrays (FPGAs) are reconfigurable hardware devices that can be programmed to perform specific computations. They offer high parallelism and can be optimized for specific tasks, making them suitable for real-time applications.

FPGA Architecture for SRCNN:

- **Design Considerations**: Implementing SRCNN on FPGA involves optimizing the hardware for parallel computation and efficient memory access.



- **Hardware Modules**:
  - **Convolution Modules**: Convolution is a fundamental operation in CNNs, which involves sliding a filter (kernel) over an input image to produce a feature map. Implementing the convolution operations in parallel to speed up computation.



CONVOLVER STRUCTURE FOR K = 3

Detailed Workflow

▼ **Data Fetching**:

- The input image is read from the external memory into the input buffers.
- The filter weights are also loaded into dedicated buffers.

▼ **Sliding Window Operation**:

- The line buffers hold consecutive rows of the input image, allowing the filter to slide over the image.
- At each position, a patch of the input image is fed into the multiplication units.

▼ **Element-wise Multiplication**:

- Each element of the input patch is multiplied by the corresponding filter weight.
- This operation is performed in parallel for all elements in the patch.

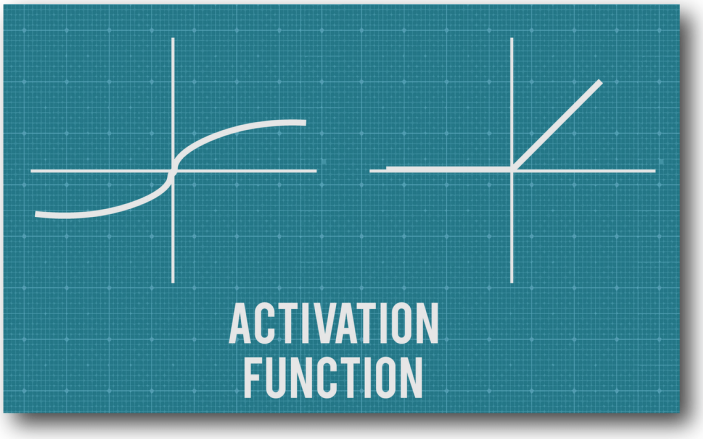
▼ **Accumulation**:

- The results of the multiplications are accumulated to produce a single value for the output feature map.
- This value is passed through the activation function to introduce non-linearity.

▼ **Output Storage**:

- The activated output value is stored in the output buffer.
- Once the entire feature map is computed, it can be written back to the external memory or used for further processing.

◦ **Activation Modules**: Activation functions are applied to the output of convolutional layers to introduce non-linearity into the model, which is essential for learning complex data patterns. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh. Implementing the ReLU function efficiently on hardware.



Detailed Workflow

▼ **Data Fetching**:

- The output feature map from the convolution module is read into the input buffers of the activation module.

▼ **Activation Processing**:

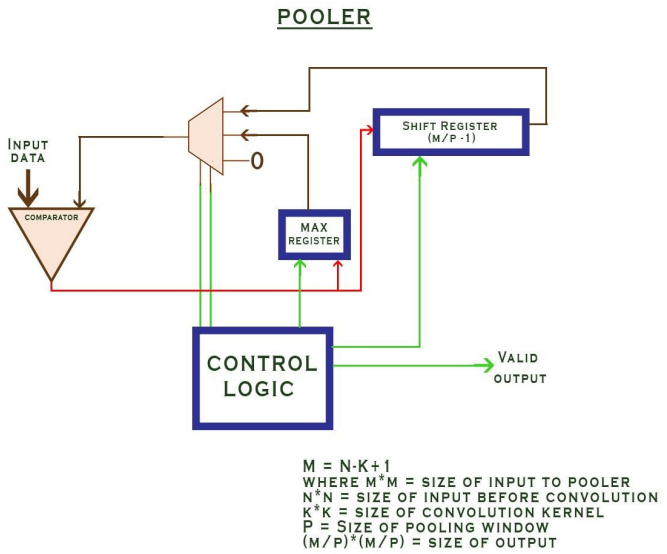
- Each element of the feature map is processed through the activation function. For ReLU, this involves a simple comparison operation. For Sigmoid and Tanh, LUTs or approximate calculations are used.

▼ **Output Storage**:

- The activated output values are stored in the output buffers.

- These values are then sent to the next layer of the network or used for further processing.

- **Pooler Module:** Pooling layers perform down-sampling operations to reduce the size of feature maps and retain essential features. Common types of pooling include Max Pooling and Average Pooling. These operations are beneficial in making the model invariant to small translations, distortions, and slight changes in scale.



Detailed Workflow

▼ **Data Fetching**:

- The input feature map is read from the external memory into the input buffers.

▼ **Sliding Window Operation**:

- The window buffer holds the current patch of the feature map.
- The window slides over the feature map with a specified stride.

▼ **Pooling Computation**:

- For Max Pooling, the maximum value within the window buffer is computed.
- For Average Pooling, the average value within the window buffer is computed.

▼ **Output Storage**:

- The pooled output value is stored in the output buffer.
- The process is repeated until the entire feature map is processed.

- **CNN Accelerator**: We have all the building blocks needed to build this accelerator (or at least a single layer of it). we need to integrate all these modules together.

Though we didn't require pooling, we still implement it as it can be used in future.

Training on GPU/CPU, Testing on FPGA:

- **Workflow**: The model is trained on a GPU/CPU, where flexibility in terms of software libraries and computational power is advantageous. The trained model parameters can then be transferred to the FPGA for real-time inference.
- **Data Transfer**: Parameters such as filter weights and biases are transferred to the FPGA using a suitable interface (e.g., PCIe).

Implementation Details:

- **Convolution Acceleration**: Detailed information about implementing CNNs on FPGA, including design flow and optimization strategies, can be found in the article: The Databus on FPGA for Convolutional Neural

Networks.

HDL Code for FPGA Implementation

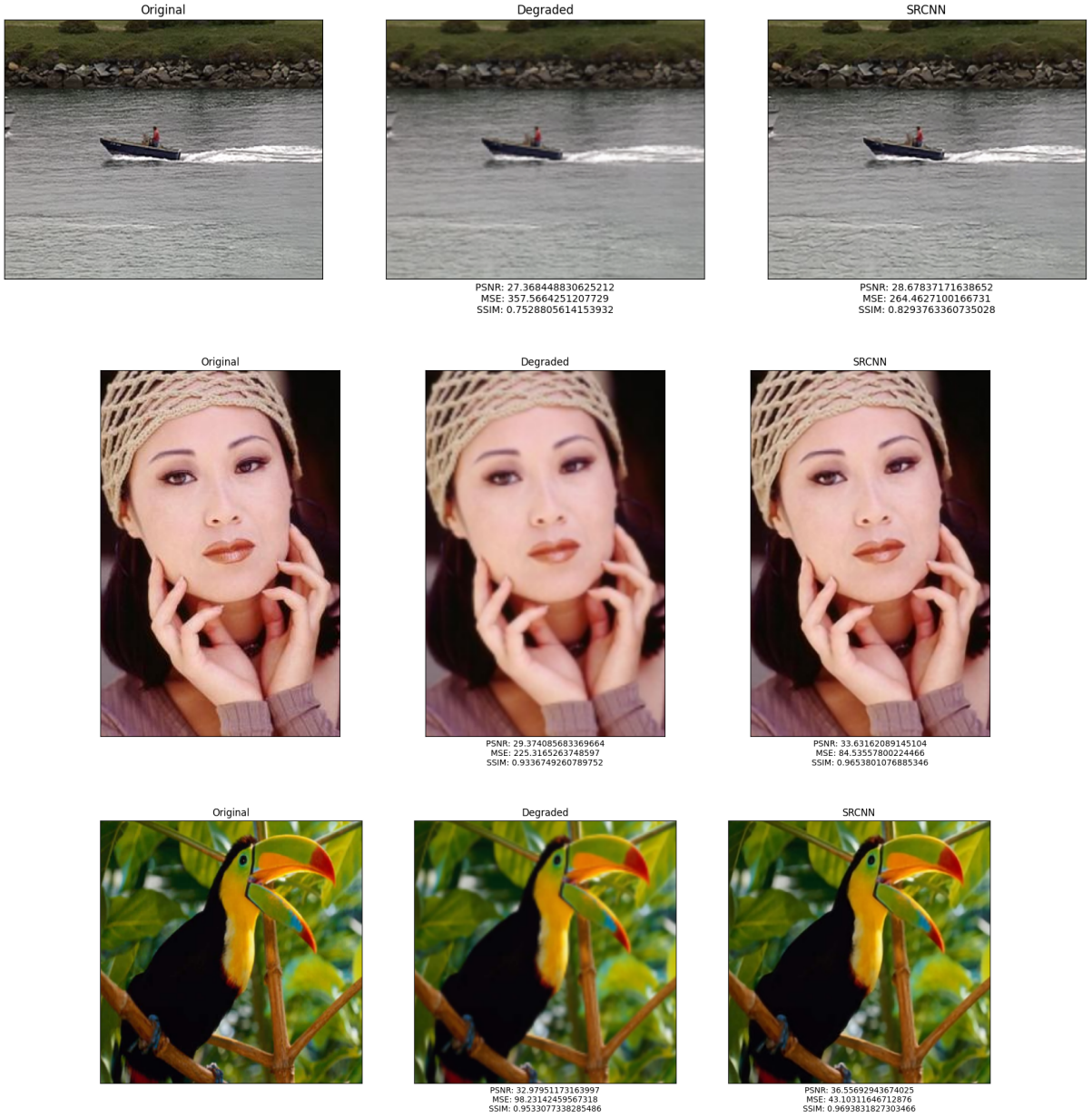
The HDL code for the FPGA implementation is available in the https://drive.google.com/drive/folders/1y1rKcvnJMw9-BSo5ni6H0iXurX216jM4?usp=drive_link, containing all necessary modules for convolution, activation, pooling, and the CNN accelerator.

We couldn't implement more high degree convolution than 2D therefore we didn't integrate FPGA with our trained model. We have implemented all the modules required. We hope in future that we will be able to implement more high degree convolution then we will integrate both the modules.

---

7. Results

Visual Results:



8. Conclusion

This report discusses the implementation and significance of the Super-Resolution Convolutional Neural Network (SRCNN) for single-image super-resolution. SRCNN improves the quality of low-resolution images through a deep learning approach, using three convolutional layers to extract and map features for reconstructing high-resolution images.

The implementation process involves training on GPUs/CPUs, employing data augmentation and learning rate scheduling, and evaluating performance with metrics such as MSE, PSNR, and SSIM. While the FPGA implementation achieved several milestones, integrating higher-dimensional convolutions remains a future goal. The report highlights the potential for implementing SRCNN on hardware like FPGAs for real time applications.

9. Future Prospects

In SRCNN (Super-Resolution Convolutional Neural Network), refining learning rates and optimizing hyperparameters will be pivotal for advancing image resolution. As we explore future directions, integrating higher-dimensional convolutions on FPGA aims to enhance real-time performance, aiming to augment SRCNN's effectiveness across various domains.

10. References

1. Dong, C., Loy, C. C., He, K., & Tang, X. (2016). Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(2), 295-307.
2. The Databus on FPGA for Convolutional Neural Networks: https://thedatabus.io/conv_acc
3. Set5 and Set14 Datasets: Learning a Deep Convolutional Network for Image Super-Resolution (cuhk.edu.hk)
4. For pretrained weights: https://github.com/MaokeAI/SRCNN-keras