Final Project Report - MLDS 201

Christos Konstantas

September 29, 2023

Contents

1 Super Resolution Convolutional Neural Network (SRCNN)

1.1 Introduction

Super-resolution (SR) ([1], [2]), in general, is a class of techniques that enhance the resolution of an imaging system. In the context of image processing, it's a way to construct high-resolution (HR) images from one or several low-resolution (LR) images. The basic idea behind SR is to combine the non-redundant information contained in multiple LR images to generate an HR image. This is often formulated as an inverse problem:

$$I_{\text{HR}} = D(H(I_{\text{LR}}) + n) \tag{1}$$

where:

- I_{HR} is the high-resolution image we want to obtain.
- I_{LR} is the low-resolution input image.
- H is the blurring function.
- D is the downsampling function.
- n is additive noise term.

Super Resolution Convolutional Neural Networks (SRCNN) are a type of Convolutional Neural Networks (CNN) designed specifically for super resolution. SRCNNs learn an end-to-end mapping of LR to HR images, bypassing the need for traditional methods that involve explicit modeling and estimation of the above functions. Therefore, in Super Resolution Convolutional Neural Networks (SRCNN), learning an end-to-end mapping from low-resolution (LR) images to high-resolution (HR) images, involves implicitly learning a model for the blurring and down-sampling that occurs when going from a high-resolution image to a low-resolution one. However, the goal is not to learn the blurring function per se, but rather to learn how to reverse this process - going from a blurred, down-sampled image back to a sharp, high-resolution one. So, in essence, we're trying to learn an "un-blurring" or "resolution-enhancing" function.

1.2 Background

A common Convolutional Neural Network (CNN) has a convolutional layer, an activation layer (usually ReLU and we add it to the convolutional layer), a pooling layer (average or max pooling), one or more fully connected layers, and an output layer (usually softmax) for a CNN that will work as an image classifier (for example handwritten digit recognition).

1.2.1 Convolutional Layer

The convolutional layer in a neural network is designed to identify various features in the input data ($\boldsymbol{x}^l_{i,j}$ elements of matrix $\boldsymbol{X}^l$ that is usually an image in the l-th layer). It does this by applying a set of learnable filters (or kernels $\boldsymbol{w}^l_k$) to different spatial regions of the input. Each filter is responsible for creating a unique feature map, which represents where in the input data a certain feature is found. The process involves two steps: convolution and activation. In the convolution step, each filter is applied to the input data, resulting in a convolved feature map $\boldsymbol{F}(\boldsymbol{X})$. In the activation step, a non-linear function (like ReLU) is applied to this feature map to introduce non-linearity into the model. The key parameters in this layer are the weights ($\boldsymbol{w}^l_k$) and biases (b^l_k) of the filters, which are learned during the training process. The sharing of filters across all spatial locations in the input ensures that the same features can be detected regardless of their position in the input. Both CNN and SRCNN have convolutional layers that apply a set of learnable filters to the input.

1.2.2 Pooling Layer

This layer reduces the spatial dimensions (width and height) of the input volume. It helps control overfitting and reduces computational cost by downsampling the input. Common methods include Max Pooling and Average Pooling. This is where SRCNN differs. SRCNNs typically do not include pooling layers. The aim of SRCNN is to increase the resolution of the input image, which is the opposite of what pooling layers do (downsampling). We can say in SRCNN this is the deconvolutional layer, which is used by the SRCNN to increase the size of the feature maps and produce the high-resolution output image. The deconvolutional layer can have a high computational cost, depending on the size and number of the filters, the stride and padding of the convolution, and the size and number of the input and output feature maps. The deconvolutional layer can also introduce some artifacts, such as checkerboard patterns or aliasing effects, due to the uneven overlap of the filters.

1.2.3 Fully Connected Layer

Before the output layer, CNNs often have one or more fully connected layers where neurons have full connections to all activations in the previous layer. These layers can help learn non-linear combinations of high-level features as represented by the output of the convolutional layers. Traditional CNNs often have one or more fully connected layers, but these are not typically used in SRCNN. Instead, SRCNN has a reconstruction layer that aggregates the high-dimensional vectors from the previous layer to form the final high-resolution output.

1.2.4 Output Layer

The final fully connected layer is often followed by a softmax activation function instead of a sigmoid, especially for multi-class classification problems like handwritten digit recognition. The softmax function outputs a vector representing the probability distributions of each class. In CNN, the output layer often uses a softmax activation function for multi-class classification problems. In contrast, SRCNN does not have a traditional output layer. The output of SRCNN is the reconstructed high-resolution image from the final reconstruction layer.

1.2.5 Notation

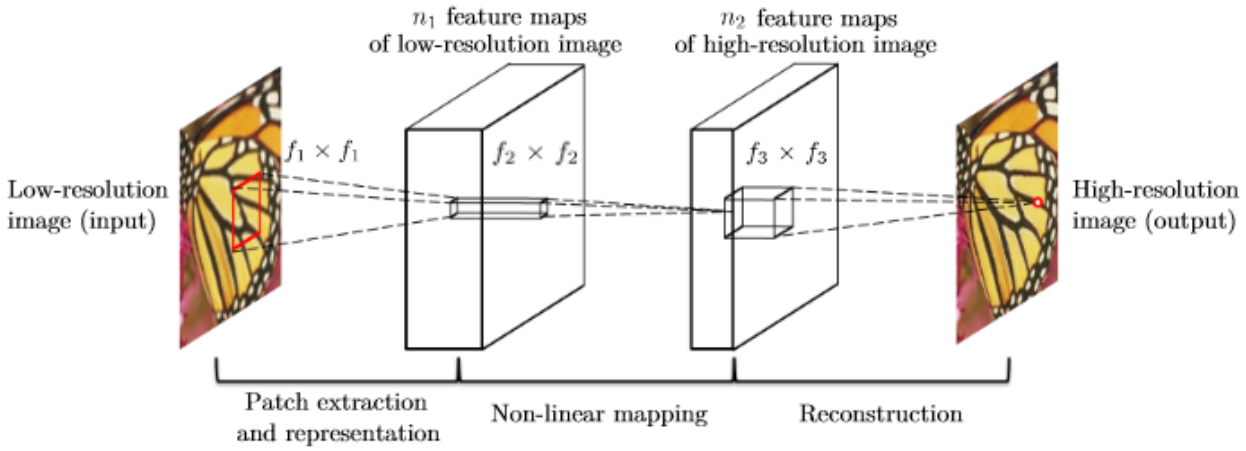


Figure 1: Super resolution process of the SRCNN [3]

Regarding the convolutional layer, the feature value at location-pixel (i, j) in the k-th feature map of the l-th layer is:

$$z_{i,j,k}^{l} = \boldsymbol{w}_{k}^{l^{\boldsymbol{T}}} \cdot \boldsymbol{x}_{i,j}^{l} + b_{k}^{l} \tag{2}$$

The complete feature maps are obtained by using several different learned kernels. The bias term in SRCNN (b_k^l) is a parameter that is added to the output of each convolutional layer. The bias term can help to adjust the output to be closer to the target, or to pass through a certain point, such as the origin and is enabled by default in the PyTorch class SRCNN. This means that each convolutional layer has a bias vector that has the same size as the number of output channels.

After the convolutions we obtain the new feature maps which will then be used as inputs to the non-linear activation function $a(\cdot)$ which can help to introduce non-linearity to the network, and make it more flexible to learn complex and high-dimensional mapping functions. We can then say that the $a_{i,j,k}^l$ of convolutional feature $z_{i,j,k}^l$ can be computed as:

$$a_{i,j,k}^{l} = a(z_{i,j,k}^{l}) \tag{3}$$

The SRCNN activation function is ReLU (Rectified Linear Unit) which gives $a(x) = \max\{0, x\}$. ReLU can help to avoid the vanishing gradient problem, which is a problem where the gradients of the network become very small or zero during the backpropagation process, making it difficult to update the weights of the network.

If we let $\boldsymbol{\theta}$ denote all the parameters of the SRCNN $(\boldsymbol{w}_k^l, b_k^l)$ then the optimum $\boldsymbol{\theta}$ for a specific task can be obtained by minimizing a suitable loss function defined explicitly for that task. Suppose then that we have N input-output relations $\{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}; n \in \{1, \ldots, N\}\}$ where $\boldsymbol{x}^{(n)}$ denotes the n-th input data (LR), $\boldsymbol{y}^{(n)}$ the corresponding ground truth high resolution target (HR) and $\boldsymbol{o}^{(n)}$ the output- final reconstruction of the SRCNN or the prediction. Then we can say that the loss function we will use is the MSE thus

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} l(\boldsymbol{\theta}; \boldsymbol{y}^{(n)}, \boldsymbol{o}^{(n)}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{\forall i,j}(y_{i,j}^{(n)} - o_{(i,j)}^{(n)})^2 \tag{4}$$

The MSE loss is a convex function, and in the context of this project, the Adam optimizer is used to minimize $\mathcal{L}$ in every epoch of the training loop until convergence to a hopefully global minimum. In Figure 1 patches of size f_1xf_1 are extracted and colvolved with the input image by moving as a sliding window. The representations are learned by a number of weights-filters with size f_1xf_1. This corresponds to one feature map. To obtain n_1 feature maps we need filters of size f_1xf_1xn_1. Then, non-linear mapping is performed using a one-by-one convolution that gives n_2 transformed feature maps. This method produces a non-linear function $o(\boldsymbol{x})$ which aims to be an image as similar as possible to $\boldsymbol{y}$.Before starting the training process, the preprocessing step involves saving a dataset of HR (high-resolution) images. A copy of this dataset is created, and the images in the copy are downscaled by a factor of 2, 3, or 4 using bicubic interpolation. The downscaled images are then upscaled back to their initial size using bicubic interpolation. This process results in a new dataset of LR (low-resolution) images. During the training process of the SRCNN, the optimization step is applied to learn better filter values. The training loss is equal to the MSE (mean squared error) loss defined in equation (4). The MSE loss measures the difference between the output image and the ground truth high-resolution image. The optimization step aims to minimize this loss by adjusting the parameters $\boldsymbol{\theta}$ of the SRCNN. Also we can say that in (4) $\boldsymbol{o}^{(n)} = \boldsymbol{o}(g(\boldsymbol{x}^{(n)})) = \text{pool}^{-1}(\boldsymbol{a}(\boldsymbol{z}^{(n)}))$ where from (2) $\boldsymbol{z}^{(n)}$ contains all $z_{i,j,k}^l$ elements, thus $\boldsymbol{z}^{(n)} = \boldsymbol{W}^{(n)^T} \cdot \boldsymbol{X}^{(n)} + \boldsymbol{b}^{(n)}$. Also $\text{pool}^{-1}(\cdot)$ is used only if the SRCNN is trained for upscale purposes. To clarify all the above we say that $\boldsymbol{z}^{(n)} = g(\boldsymbol{X}^{(n)})$ is a collection of feature values that results when assessing the n-th input/output relationship between the LR and HR target. Also for equations (2), (3) for $l = 1$, $k = 1, \ldots, n_1$ and for $l = 2$, $k = 1, \ldots, n_2$. The filters (weights) have size $f_1 \times f_1$ for $l = 1$ and $f_2 \times f_2$ for $l = 2$. At last N can be expressed as

the number of training samples or the total number of extracted patches from the input image. Also we can see $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$ as a tuple that contains all b_k^l along with their corresponding weights $\boldsymbol{w}_k^l$.

1.3 Process

1.3.1 Step 1

At the very first phase we have to prepare the data and define that the height and width of each image will be equal to the minimum height and width in the **DIV2K** dataset (train folder). In addition the images in the train folder will be turned into YCbCr format and the input to the training model SRCNN, will be only the Y channel which preserves more information compared to Cb and Cr channels.

1.3.2 Step 2

Convolve the inputs $\boldsymbol{X}$ with the filter $\boldsymbol{W}_1$ and then apply the ReLU $a_1(g(\boldsymbol{X})) = \max(0, \boldsymbol{W}_1 * \boldsymbol{X} + \boldsymbol{b}_1)$, where $\boldsymbol{W}_1 \in \mathbb{R}^{c \times f_1 \times f_1 \times n_1}$ that is the collection of all $f_1 \times f_1$ sized filters, for every colorspace ($c = 3$ for RGB, or YCbCr) and for every feature map (n_1 totally in the first layer). Also $\boldsymbol{B}_1 \in \mathbb{R}^{n_1}$ which is the collection of biases for all n_1 feature maps in the first layer. This is the patch extraction and representation step in Figure 1.

1.3.3 Step 3

We repeat the same process as in Step 2 but now we have $a_2(g(\boldsymbol{X})) = \max(0, \boldsymbol{W}_2 * a_1(g(\boldsymbol{X})) + \boldsymbol{b}_2)$ where $\boldsymbol{W}_2 \in \mathbb{R}^{n_1 \times 1 \times 1 \times n_2}$ and $\boldsymbol{b}_2 \in \mathbb{R}^{n_2}$. This time we have one-by-one convolutions and that's the non-linear mapping in Figure 1.

1.3.4 Step 4

For the reconstruction there is no non-linearity. What we do can be shown in this formula: $o(\boldsymbol{X}) = \max(0, \boldsymbol{W}_3 * \boldsymbol{X} + \boldsymbol{b}_2)$, where $\boldsymbol{W}_3 \in \mathbb{R}^{n_2 \times f_3 \times f_3 \times c}$ and $\boldsymbol{b}_3 \in \mathbb{R}^c$. Using the above logic we can add extra layers too, to our SRCNN. Our loss function is given then by (4) and the optimizer should find $\arg\min_{\boldsymbol{\theta}} \{\mathcal{L}\}$.

1.4 Optimizer

The optimizer chosen to achieve minimization of the MSE and find the best $\boldsymbol{\theta}$ is the well known Adam optimizer. The Adam (Adaptive Moment Estimation) optimizer is a widely used optimization algorithm in the field of machine learning, particularly in deep learning applications. It combines the strengths of two fundamental optimization techniques: Momentum and Adaptive Gradient Algorithm (AdaGrad). Adam is particularly valuable because it offers adaptive learning rates, robustness to noisy gradients, and is well-suited for addressing sparse gradients. Its effectiveness and versatility have made it a popular choice for training a wide range of machine learning models, including neural networks. At its core, Adam is an iterative optimization algorithm like SGD. Both aim to minimize a loss function by updating model parameters iteratively. Adam maintains a per-parameter learning rate, whereas SGD typically uses a fixed global learning rate. This adaptability allows Adam to converge efficiently, especially when dealing with varying gradient magnitudes across parameters. While SGD updates parameters based on the current gradient, Adam incorporates momentum by considering past gradients. This momentum term helps smooth out parameter updates and speed up convergence, similar to the concept of momentum in SGD. Adam is not limited to convex optimization problems,

Algorithm 1 Adam Optimizer

Initialize model parameters θ, step size α, β_1, β_2, and ϵ
Initialize first moment vector m with zeros, and second moment vector v with zeros
Initialize time step $t \leftarrow 0$
while Not converged **do**
  Sample a mini-batch of training data with inputs X and targets Y
  Compute gradients $\nabla_\theta \mathcal{L}(\theta; X, Y)$ with respect to the loss function
  Increment time step $t \leftarrow t + 1$
  Update first moment estimate: $m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}(\theta; X, Y)$
  Update second moment estimate: $v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot (\nabla_\theta \mathcal{L}(\theta; X, Y))^2$
  Correct bias in first moment: $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$
  Correct bias in second moment: $\hat{v} \leftarrow \frac{v}{1-\beta_2^t}$
  Compute parameter update: $\Delta\theta \leftarrow -\alpha \cdot \frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon}$
  Update model parameters: $\theta \leftarrow \theta + \Delta\theta$
end while

and it is commonly used in non-convex optimization scenarios, such as training neural networks. Adam's adaptability and robustness to noisy gradients make it well-suited for navigating complex, non-convex loss surfaces often encountered in deep learning. In summary, Adam is a versatile optimization algorithm that combines the advantages of SGD, momentum, and adaptive learning rates. While it is effective for non-convex optimization, its ability to adapt learning rates and handle varying gradient magnitudes is valuable in a wide range of machine learning scenarios. The Adam optimizer has several hyperparameters, each of which plays a crucial role in the optimization process:

- α (**Learning Rate**): The learning rate determines the step size for updating model parameters. It is denoted by α and is typically a small positive value, e.g., $\alpha = 0.001$.
- β_1 (**First Moment Decay Rate**): β_1 controls the exponential decay of the moving average of gradients (first moment estimate). It is a value between 0 and 1, e.g., $\beta_1 = 0.9$.
- β_2 (**Second Moment Decay Rate**): β_2 controls the exponential decay of the moving average of squared gradients (second moment estimate). It is a value between 0 and 1, e.g., $\beta_2 = 0.999$.
- ϵ (**Smoothing Term**): ϵ is a small positive constant added to the denominator to prevent division by zero and improve numerical stability. Common values are in the range of 10^{-7} to 10^{-8}, e.g., $\epsilon = 10^{-8}$.
- t (**Time Step**): t represents the current time step or iteration in the optimization process. It starts from 1 and increments with each iteration.

These hyperparameters collectively influence the behavior of the Adam optimizer and its convergence characteristics.

1.5 Training

In our training loop, we follow a structured approach. We start by loading the necessary data: High-Resolution (HR) images from the training folder, their corresponding Low-Resolution (LR) images created through bicubic interpolation, and images from the validation folder. Within this loop, we use mini-batches, which are small subsets of our data, and accumulate gradients over several of these mini-batches. We then proceed with forward and backward passes to calculate the loss and improve the model. We also incorporate a dynamic learning rate strategy. This means that as training

progresses, the learning rate is adjusted systematically after a certain number of steps to help the model converge effectively. At the end of each training epoch, we assess the model's performance on the validation dataset and compute the validation loss. To ensure that we stop training at the right moment, we keep an eye on the validation loss. If it starts increasing, we apply early stopping to prevent overfitting. We save the best-performing model to preserve our progress.

2 Results

In this section, we will present the results of our SRCNN (Super-Resolution Convolutional Neural Network) model implementation and discuss its methodology. The objective of our approach is to enhance the quality of images using SRCNN. To facilitate the operations of the neural network, we initially convert RGB images into the YCbCr color space. This conversion separates the luminance (Y) channel from the chrominance (Cb and Cr) channels, allowing the SRCNN to focus solely on the one-dimensional Y channel. During the reconstruction phase, we combine the reconstructed Y channel with the bicubic interpolated Cb and Cr channels. Subsequently, we convert the YCbCr image back into the RGB format, restoring it to its original appearance. Our approach begins with single-image processing, followed by the evaluation of 19 images from datasets like Set4 and Set15. Finally, we extend our model to handle 1000 images from the complete DIV2K dataset, which comprises 800 images for training and 200 for evaluation. It's worth noting that our SRCNN model primarily focuses on image enhancement rather than upscaling. If upscaling is desired, we recommend implementing a transpose convolution layer in the final stage of the network architecture.

2.1 Single image enhancement (no generalization)

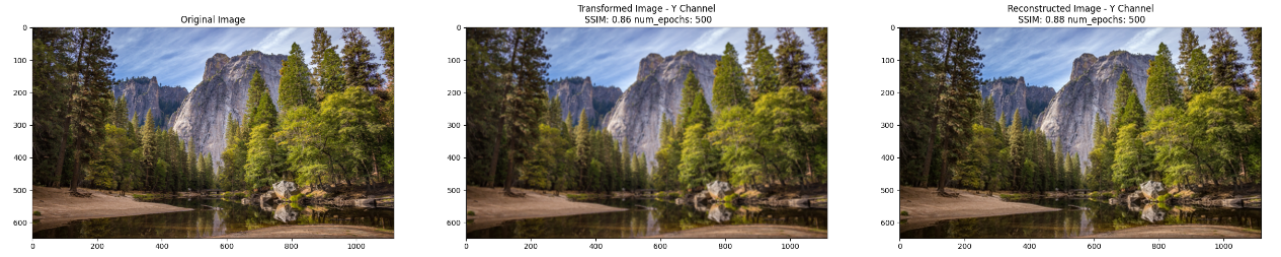
During this phase we focus on single image enhancement (a chosen image from the DIV2K dataset). The results appear in Figure 2. As we can see we have improvements in quality, considering SSIM as an IQA (Image Quality Assesment) metric.

1. For downscale_factor = 2 the PSNR between the original and the bicubic image is 31.89 dB. Also the PSNR between the original and the reconstructed image is 32.08 dB.
2. For downscale_factor = 3 the PSNR between the original and the bicubic image is 30.87 dB. Also the PSNR between the original and the reconstructed image is 31.03 dB.
3. For downscale_factor = 4 the PSNR between the original and the bicubic image is 30.40 dB. Also the PSNR between the original and the reconstructed image is 30.54 dB.

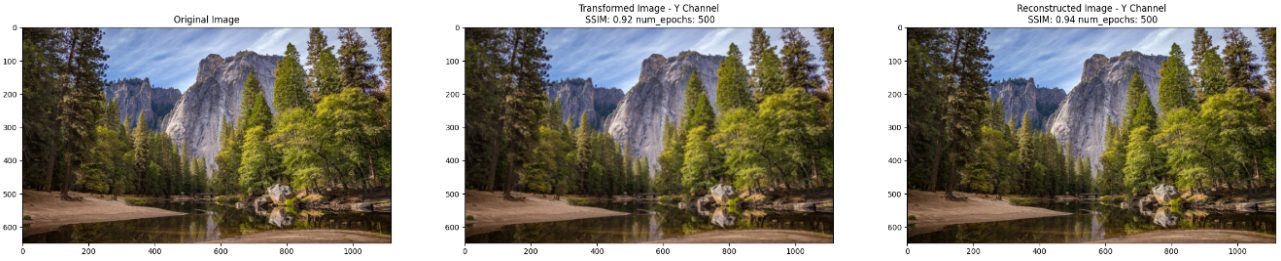
At last the model we have created has the below characteristics:

- The number of feature maps in the first layer is $n_1 = 64$.
- The kernel size in the first layer $(f_1 \times f_1)$ is 9×9.
- The number of feature maps in the second layer is $n_2 = 32$.
- The kernel size in the second layer $(f_2 \times f_2)$ is 5×5.
- The number of color channels is $c = 1$ because we only train the Y channel.

(a) downscale_factor = 4



(b) downscale_factor = 3



(c) downscale_factor = 2

Figure 2: The Structural Similarity Index Measure (SSIM) was computed for various downscale factors (2, 3, 4) in the experimental analysis. A comprehensive assessment determined that a fixed number of 500 epochs yielded optimal results. Additionally, a learning rate of 0.001 was employed in these experiments for consistency and stability.

2.2 Multiple image enhancement (some generalization)

Here the model is trained for downscale factor equal to 2 for the Set14 dataset. The model is the same as in the previous subsection and again we don't use SRCNN for upscaling purposes, only for enhancing the image. Below three different images are illustrated, two that belong in the training dataset, one of them partially (Figure 3) and one totally (Figure 6). When we say partially we mean that Figure 3 is the monarch.png from Set14 but its zoomed version. The third image was unprocessed by the training phase and belongs in Set5 dataset 5. We didn't use any evaluation logic at this point. The Set 14 dataset was trained for 800 epochs. To get an intuition of the whole concept of Figure 1 on how the image of Figure 3 results, you can see Figure 4.

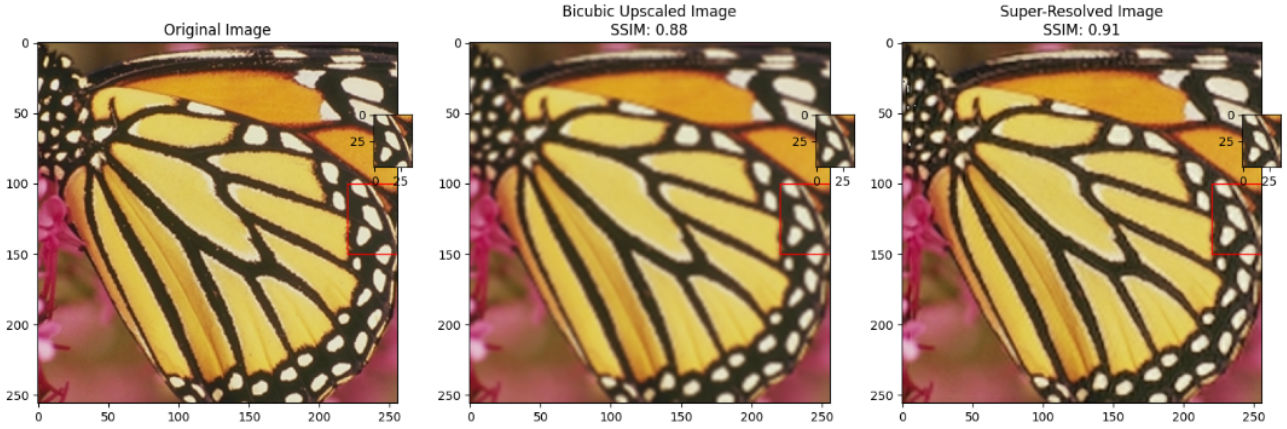


Figure 3: butterfly.png zoomed test image from Set14 dataset which was partially seen in the training phase. We see some generalization here. Also we notice some aliasing effects that is a common problem that may happen in SRCNNs and they can be seen where x-axis value is zero and y-axis value is 50 to 60 as very little white dots.

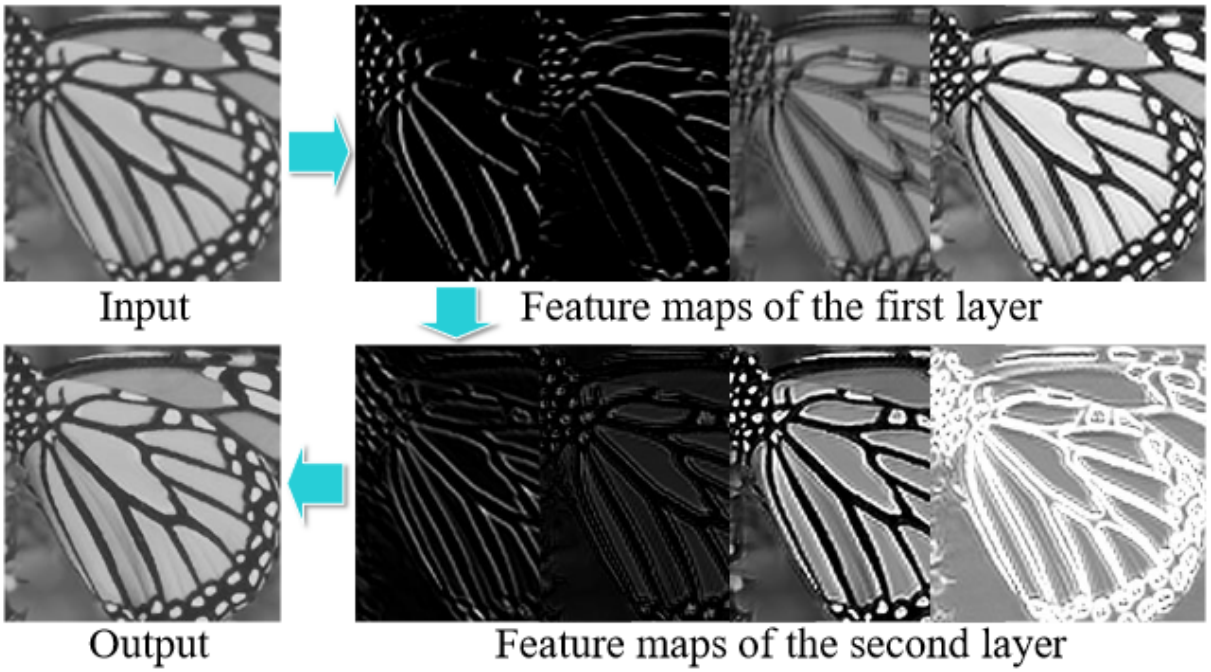


Figure 4: Feature maps of different layers [3]

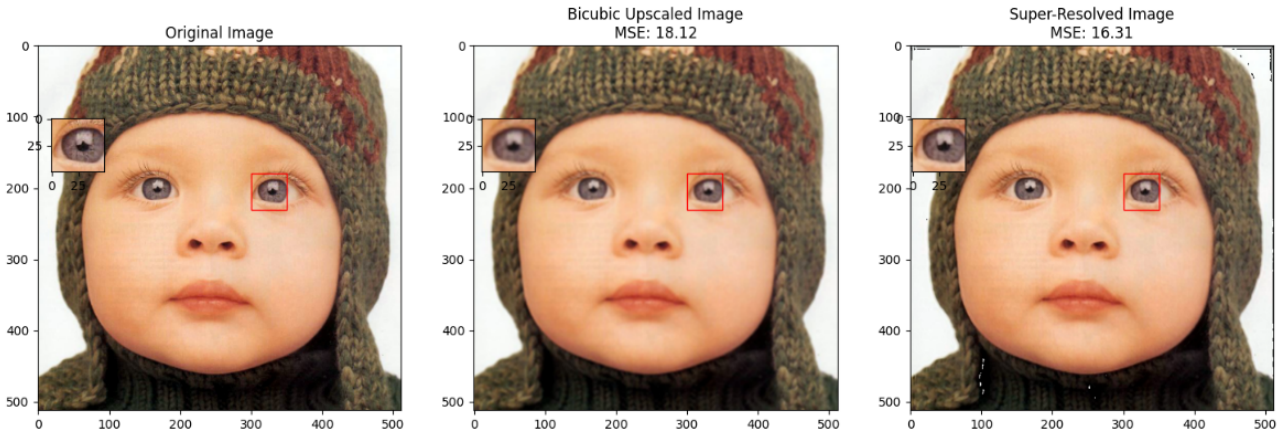
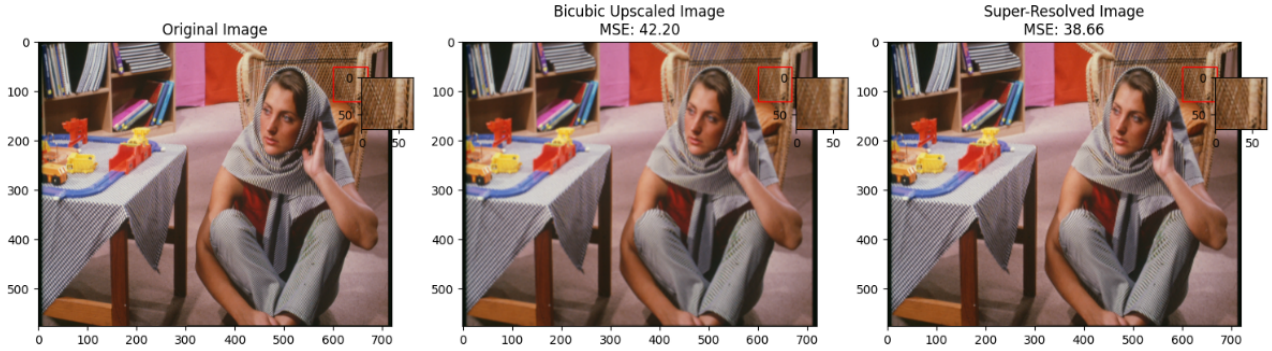
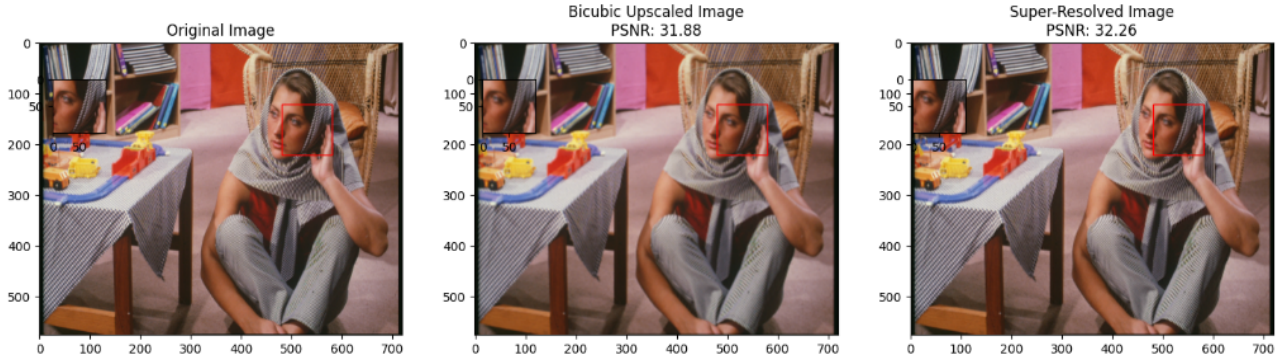


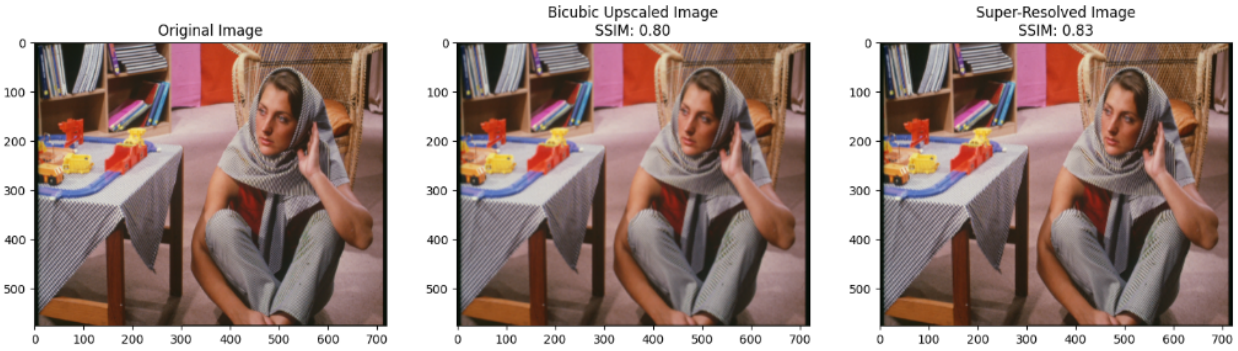
Figure 5: baby.png image from Set5 dataset MSE loss. As in Figure 3 we still notice an aliasing effect.



(a) Image depicting MSE loss



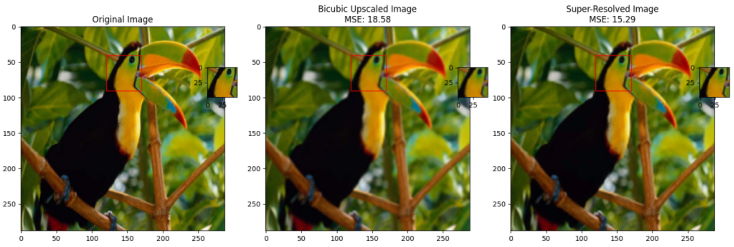
(b) Image depicting PSNR gain



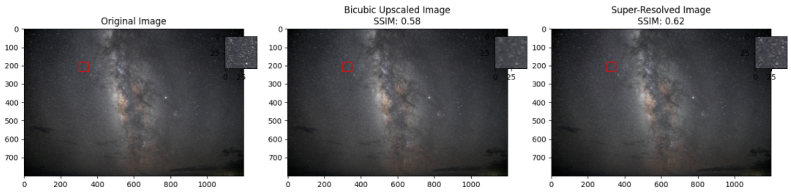
(c) Image depicting SSIM gain

Figure 6: barbara.png image enhancement from Set 14 after downscaling and upscaling by a factor of 2 and after that applying a Gaussian Blur Kernel of size 5×5 and standard deviation of 0 in the image. The bicubic upscaled image that you notice contains the Gaussian blur operation.

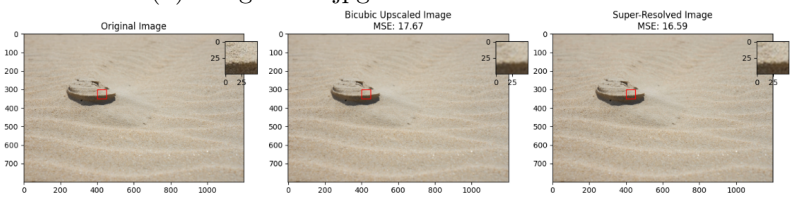
2.3 Multiple image enhancement (enough generalization)



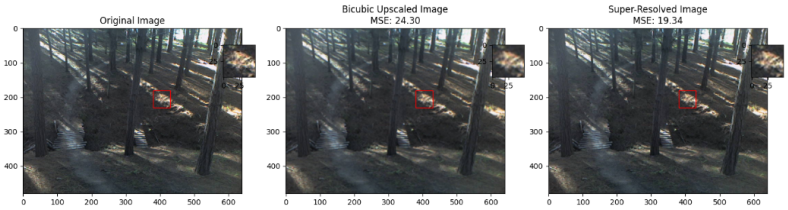
(a) Image bird.png from Set5 dataset.



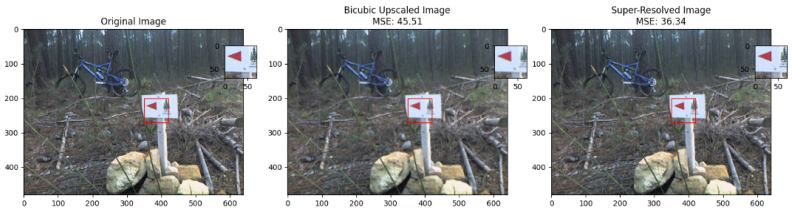
(b) Image 1034.jpg from BSD100 dataset.



(c) Image 1046.jpg from BSD100 dataset.



(d) Random low-quality image in nature.



(e) Random low-quality image in nature.

Figure 7: Test images after training the SRCNN model on the entire DIV2K dataset.

As we can see the training on DIV2K dataset with 800 high resolution images and 200 high resolution validation images is even more generalized. To achieve even greater generalization, one might consider as a future work to train even more high resolution images for even more epochs, utlizing as many GPUs as possible to accelerate training time as long as have an adaptive learning rate scheduler and/or a regularizer so that the model doesn't overfit on the training data.

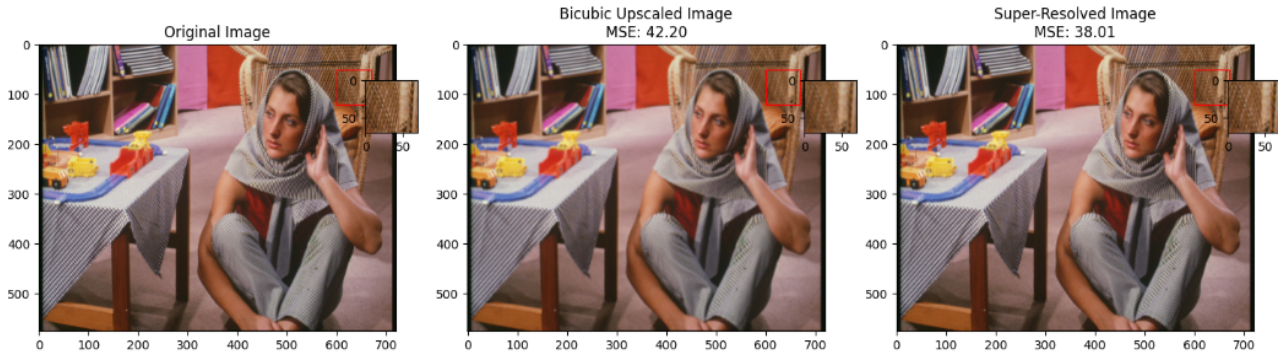


Figure 8: barbara.png test on the DIV2K pre-trained model gives less MSE than test on Set14 pre-trained model in Figure 6a.

3 Advanced Super-Resolution Techniques

1. **Generative Adversarial Networks (GANs):** GANs have made significant contributions to super-resolution. Models like SRGAN (Super-Resolution GAN) use a generator and a discriminator network to not only improve resolution but also enhance the visual quality of images. GANs can produce sharp and highly detailed images but are more complex to train and can be computationally intensive [4].

2. **PixelRL (Pixel Reinforcement Learning):** PixelRL is an interesting approach that applies reinforcement learning techniques to the super-resolution task. It formulates the problem as a sequential decision-making process, where the model learns to generate high-resolution pixels step by step. While it's a novel approach, it may require more complex training and resources [5].

3. **Attention Mechanisms:** Many recent super-resolution models incorporate attention mechanisms, such as self-attention or channel-wise attention, to focus on relevant image regions. These mechanisms help the model capture fine details and long-range dependencies in the images, leading to improved results.

4. **Multi-Scale Architectures:** Some models use multi-scale architectures, where the network processes images at different resolutions simultaneously. This approach allows the model to capture information at various scales and then fuse it together to produce high-quality results.

5. **Real-World Applications:** Super-resolution techniques find applications in various domains, including medical imaging, satellite imaging, surveillance, and more. For instance, in medical imaging, super-resolution can help enhance the clarity of MRI or CT scans, aiding in more accurate diagnoses.

6. **Challenges:** Despite the advancements, super-resolution still faces challenges like dealing with real-world noise, handling low-resolution images with extreme upscaling factors, and making the models more efficient for practical deployment.

In summary, the field of super-resolution is continually evolving, with a wide range of models and techniques being developed to address different aspects of the problem. Depending on the specific application and computational resources available, researchers and practitioners choose the most suitable approach to achieve high-quality, super-resolved images.

A Supplementaries

A.1 YCbCr format

The YCbCr color space is a color representation that separates the luminance (Y) component from the chrominance (Cb and Cr) components of an image. It is widely used in image and video compression because it allows for efficient representation and compression of color information.

The conversion from the RGB color space to the YCbCr color space is typically performed using the following equations:

$$
\begin{aligned}
Y &= 0.299R + 0.587G + 0.114B \\
Cb &= \frac{1}{2}(B - Y) \\
Cr &= \frac{1}{2}(R - Y)
\end{aligned}
$$

Where: - Y represents the luminance component (brightness) of the image. - Cb and Cr represent the chrominance components (color information). - R, G, and B are the red, green, and blue components of the original image.

A.1.1 Training Y Component and Bicubic Interpolation for Cb and Cr

In many image processing tasks, such as super-resolution, it is common to train models to enhance the quality of the Y (luminance) component while leaving the Cb and Cr (chrominance) components untouched. There are two main reasons for this approach:

1. Human Visual Perception: The human visual system is more sensitive to changes in brightness (luminance) than to changes in color (chrominance). Enhancing the Y component can lead to a perceptually improved image.

2. Reduced Complexity: Training models separately for the Y component simplifies the training process. Since color information (Cb and Cr) is less critical for most tasks, it is more efficient to apply bicubic interpolation to upscale or downscale these components.

A.1.2 Reconstruction Phase

In the reconstruction phase, after enhancing the Y component using the trained model, the YCbCr image is converted back to RGB. The reconstructed RGB image is obtained using the following equations:

$$
\begin{aligned}
R' &= Y + 1.13983 \cdot Cr' \\
G' &= Y - 0.39465 \cdot Cb' - 0.58060 \cdot Cr' \\
B' &= Y + 2.03211 \cdot Cb'
\end{aligned}
$$

Where: - R', G', and B' are the red, green, and blue components of the reconstructed RGB image. - Y, Cb', and Cr' are the enhanced Y component and the original Cb and Cr components.

In this way, the enhanced Y component is combined with the original Cb and Cr components to produce the final RGB image, preserving color information while benefiting from the improved luminance.

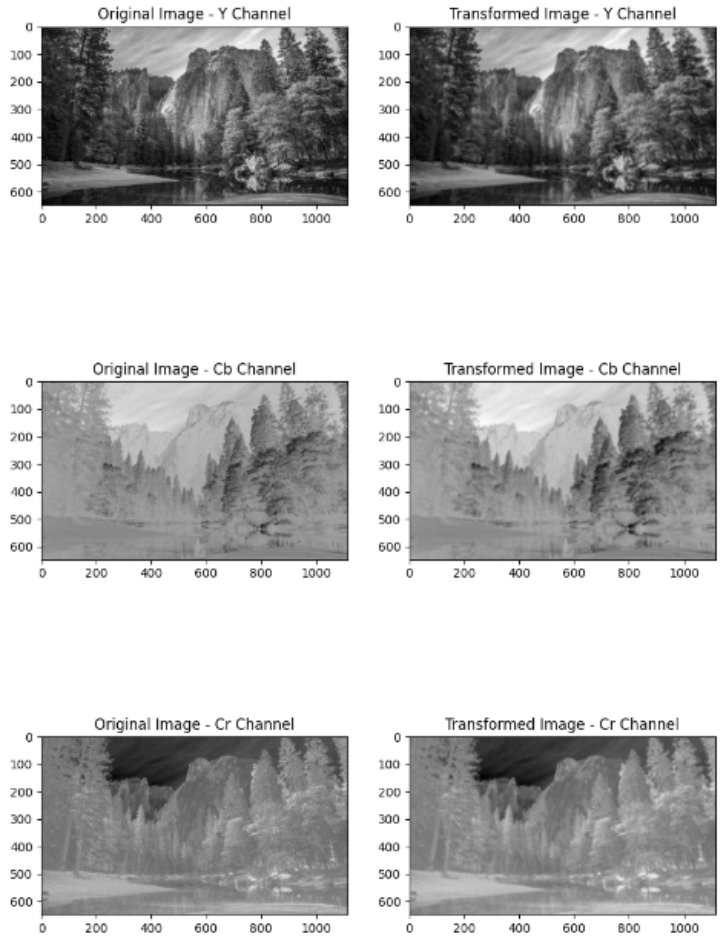


Figure 9: YCbCr illustration. Notice that Y channel contains more critical information about the image. We use grayscale mapping to simplify visualization while preserving luminance information.

B Image Quality Assesment (IQA) metrics

B.1 MSE

MSE, or Mean Square Error, is a commonly used metric in image processing to measure the average squared difference between the values of the original image and the values of a processed or reconstructed image. It is calculated using the following formula:

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

Where: - MSE represents the Mean Square Error. - $I(i,j)$ is the pixel value of the original image at position (i,j). - $K(i,j)$ is the pixel value of the processed or reconstructed image at position (i,j). - m and n are the dimensions of the image.

A lower MSE indicates a smaller difference between the original and processed images, implying better image quality.

B.2 PSNR (Peak Signal-to-Noise Ratio)

PSNR, or Peak Signal-to-Noise Ratio, is a metric used to measure the quality of a processed image compared to the original image. It is calculated using the MSE and has the following formula:

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

Where: - PSNR represents the Peak Signal-to-Noise Ratio. - MAX is the maximum possible pixel value (e.g., 255 for an 8-bit image).

A higher PSNR value indicates better image quality and less distortion.

B.3 SSIM (Structural Similarity Index)

SSIM, or Structural Similarity Index, is a metric that assesses the structural similarity between two images, including luminance, contrast, and structure. It is computed using the following formula:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Where: - $\text{SSIM}(x, y)$ is the SSIM between images x and y. - μ_x and μ_y are the means of images x and y. - σ_x and σ_y are the standard deviations of images x and y. - σ_{xy} is the covariance between images x and y. - c_1 and c_2 are constants to stabilize the division.

The SSIM value ranges from -1 to 1, where 1 indicates perfect similarity, and higher values imply better image quality and similarity. In the context of the project SSIM is normalized and ranges from 0 to 1.

C Bicubic Interpolation

Bicubic interpolation is a widely used image resizing technique that aims to enhance the quality of an image when it is scaled up or down. It achieves this by estimating pixel values that lie between the original pixel positions in a smoother and more accurate manner than nearest-neighbor or bilinear interpolation methods.

The basic idea behind bicubic interpolation is to fit a cubic polynomial to the pixel values in a local 4x4 neighborhood and use this polynomial to compute the interpolated pixel values. The interpolated value at any point within this neighborhood is given by the following formula:

$$I(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij} \cdot (x - x_0)^i \cdot (y - y_0)^j$$

Where: - $I(x, y)$ is the interpolated pixel value at position (x, y). - P_{ij} are the pixel values in the 4x4 neighborhood. - x_0 and y_0 are the coordinates of the center pixel in the neighborhood.

Bicubic interpolation offers several advantages:

1. Smoother Results: It produces smoother and less pixelated images compared to simpler interpolation methods.

2. Preservation of Edges: It helps preserve edges and fine details when upscaling images.

3. High-Quality Resizing: Bicubic interpolation is commonly used in image scaling algorithms to improve image quality.

However, it is computationally more intensive than nearest-neighbor or bilinear interpolation due to the cubic polynomial fitting.

In image processing, bicubic interpolation is often used for upscaling images, especially when maintaining image quality is crucial.

References

[1] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao, "Deep learning for single image super-resolution: A brief review," *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[2] E. J. Candès and C. Fernandez-Granda, "Towards a mathematical theory of super-resolution," *Communications on pure and applied Mathematics*, vol. 67, no. 6, pp. 906–956, 2014.

[3] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[4] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4681–4690, 2017.

[5] R. Furuta, N. Inoue, and T. Yamasaki, "Pixelrl: Fully convolutional network with reinforcement learning for image processing," *IEEE Transactions on Multimedia*, vol. 22, no. 7, pp. 1704–1719, 2019.