LNMIIT
The LNM Institute of
Information Technology

Digital Image Processing

(CSE2012)

Assignment - 1

Submit to: **Dr Anukriti Bansal**

Submitted By

VAIBHAV KHAMESRA - 21UCS224

SHREYASH ACHARYA - 21UCS200

DIVYANSH GARG - 21DCS003

Acknowledgment

It is a matter of great satisfaction and pleasure to present this project on “Image replication and Interpolation and Image Bit Plane Slicing ”, on the very outset of this report, We would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, We would not have made headway in the project.

We take this opportunity to owe our thanks to our faculty member, Dr. Anukriti Bansal, for their encouragement and guidance at every stage of this project.

We express our gratitude to all those who have directly or indirectly helped us to make this project.

Collaborations

We are thankful to:

1. Sparsh Dalal
2. Harshit Chaudhry

For helping us in this assignment

Problem Statement:

Write programs in C/C++ or Python using OpenCV to implement the following tasks:

1. The basic intent of this part of the assignment is to perform the scaling operation to

an image. The scaling up should be done by

    a. Replication

    b. Interpolation

The scaling down should be done with the corresponding operations as would be used for scaling up.

Consider a window (a rectangular area) of a given size for the display within which the zoomed-in or out area is shown when placed on the image.

2. Load the image (8-bit) in grayscale. Perform bit plane slicing. Display all bit planes.

Reconstruct and display the image using following bit planes:

    a. Plane 8, plane 7

    b. Plane 8, plane 7, and plane 6

    c. Plane 8, plane 7, plane 6, and plane 5

Scaling Operation On Images

Scaling an image involves changing its size, either by enlarging or reducing it.

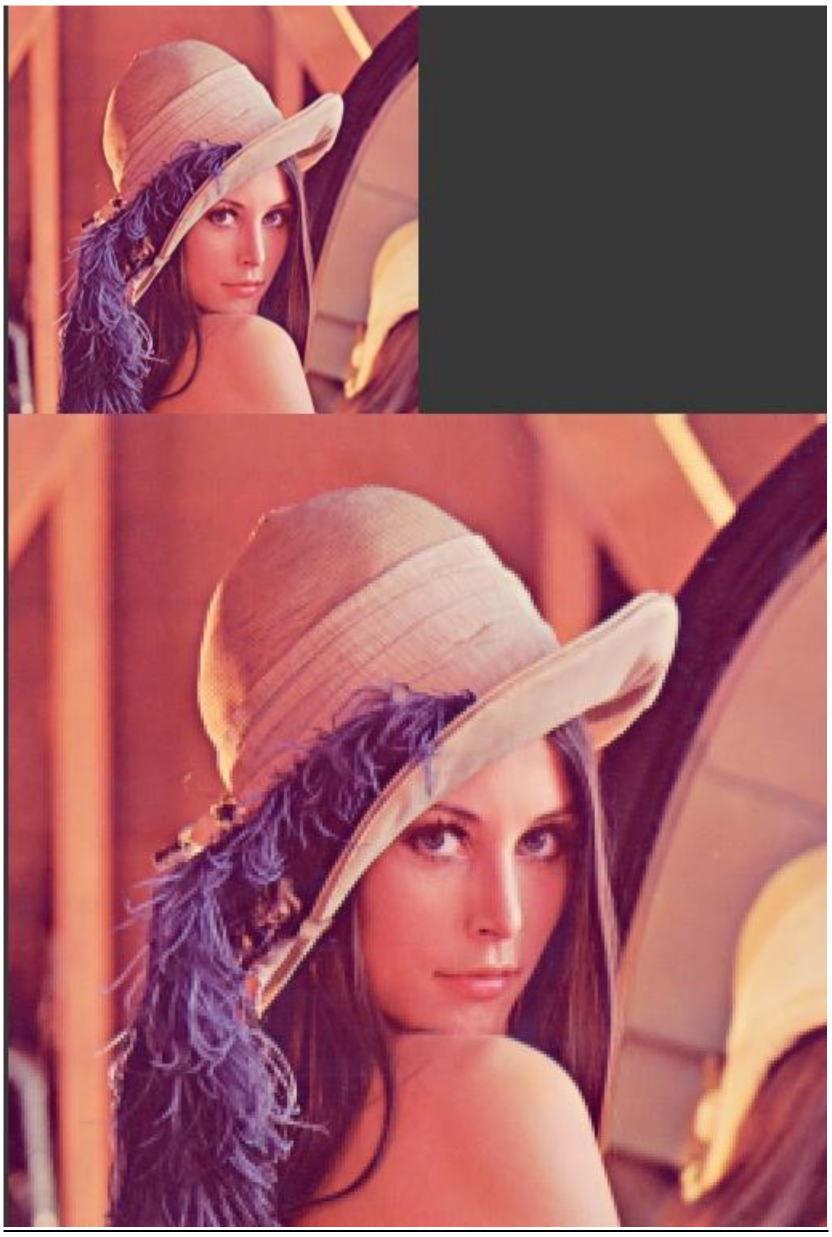
While scaling an image, it is important to maintain the aspect ratio of the image to avoid distortions.

Additionally, scaling an image up too much can result in a loss of image quality and pixelation, while scaling an image down too much can lead to a loss of details.

The various method to perform scaling operations are :

1. Replication
2. Bilinear Interpolation
3. Bicubic Interpolation

Scaling Operation Using Replication



Scaling up – Scaling up an image using replication involves creating a larger version of the image by duplicating each pixel in the original image a certain number of times to fill in the additional space in the new image. The replication process involves copying the color values of the existing pixels and pasting them into the new locations. This technique is also known as nearest-neighbor interpolation.

To scale up an image using replication, you can follow these steps:

1. Determine the size of the new image: Decide how much you want to scale up the original image. For example, if you want to double the size of the image, the new image will have twice as many pixels in both width and height.

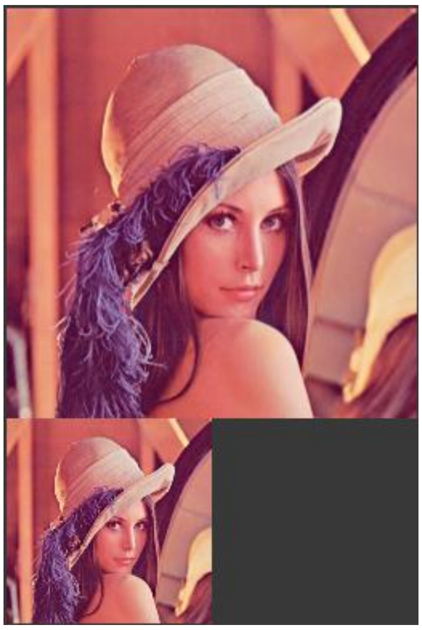
2. Create a new empty image: Use image editing software to create a new image that is the desired size.

3. Replicate the pixels: For each pixel in the original image, replicate its color value to the corresponding position in the new image. For example, if the original image has a pixel at position (2,3) with a color value of red, copy the color value to positions (4,6), (4,7), (5,6), and (5,7) in the new image.

4. Save the new image: Save the new image in a file format of your choice.

Scaling up an image using replication is a simple and fast method, but it may not always produce the best results. The resulting image can appear blocky or pixelated, especially if the scaling factor is large. This is because the algorithm does not take into account the values of neighbouring pixels, resulting in a loss of image quality. Other interpolation techniques, such as bicubic interpolation can produce better results, but they are more computationally intensive.

Scaling down – Scaling down an image using replication involves reducing the size of an image by discarding some of the pixels in the original image and replicating the remaining pixels to fill in the new image. This process is also known as nearest-neighbour interpolation.



To scale down an image using replication, you can follow these steps:

1. Determine the size of the new image: Decide how much you want to scale down the original image. For example, if you want to reduce the size of the image by half, the new image will have half as many pixels in both width and height.

2. Create a new empty image: Use image editing software to create a new image that is the desired size.

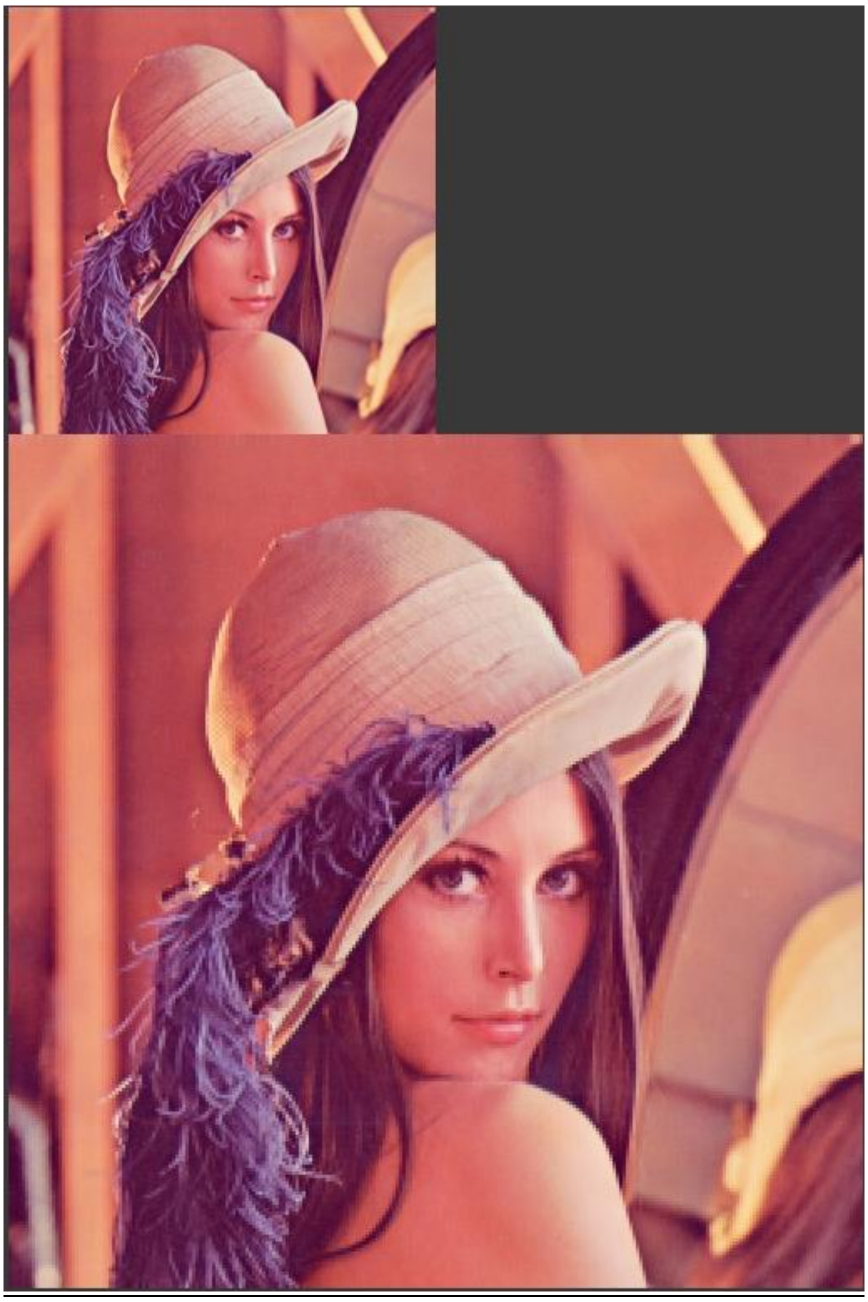
3. Replicate the pixels: For each pixel in the new image, find the corresponding pixel in the original image by multiplying the x and y coordinates by the scaling factor. Then, replicate the color value

of the original pixel to the corresponding position in the new image.

4. Save the new image: Save the new image in a file format of your choice.

Scaling down an image using replication is a simple and fast method, but it can result in a loss of image quality, especially if the scaling factor is large. The algorithm does not take into account the values of neighboring pixels, resulting in a loss of detail and image sharpness. Other interpolation techniques, such as bilinear interpolation or bicubic interpolation, can produce better results, but they are more computationally intensive.

Scaling Operation Using Interpolation



Scaling up – Scaling up an image using bilinear interpolation involves creating a larger version of the image by calculating the color values of the new pixels based on the values of the neighboring pixels in the original image. This interpolation method uses a weighted average of the four nearest pixels to determine the color of each new pixel, resulting in smoother and more accurate results than replication.

To scale up an image using bilinear interpolation, you can follow these steps:

1. Determine the size of the new image: Decide how much you want to scale up the original image. For example, if you want to double the size of the image, the new image will have twice as many pixels in both width and height.

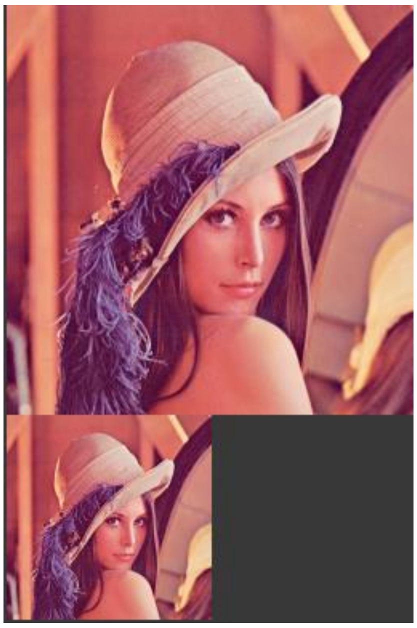
2. Create a new empty image: Use image editing software to create a new image that is the desired size.

3. Calculate the color values of the new pixels: For each pixel in the new image, find the corresponding position in the original image by dividing the x and y coordinates by the scaling factor. Then, calculate the color value of the new pixel by taking a weighted average of the four nearest pixels in the original image.

4. Save the new image: Save the new image in a file format of your choice.

Bilinear interpolation considers the values of neighboring pixels and produces smoother and more accurate results than replication, especially when scaling up an image by a large factor. However, it can still result in some loss of image quality, especially if the scaling factor is large. Other interpolation techniques, such as bicubic interpolation can produce even better results, but they are more computationally intensive.

Scaling down – Scaling down an image using bilinear interpolation involves reducing the size of an image by calculating the color values of the new pixels based on the values of the neighboring pixels in the original image. This interpolation method uses a average of the four nearest pixels to determine the color of each new pixel, resulting in smoother and more accurate results than replication.



To scale down an image using bilinear interpolation, you can follow these steps:

1. Determine the size of the new image: Decide how much you want to scale down the original image. For example, if you want to reduce the size of the image by half, the new image will have half as many pixels in both width and height.

2. Create a new empty image: Use image editing software to create a new image that is the desired size.

3. Calculate the color values of the new pixels: For each pixel in the new image, find the corresponding position in the original image by multiplying the x and y coordinates by the scaling factor. Then, calculate the color value of the new

pixel by taking a weighted average of the four nearest pixels in the original image.

4. Save the new image: Save the new image in a file format of your choice.

Bilinear interpolation produces smoother and more accurate results than replication when scaling down an image. It takes into account the values of neighboring pixels and results in a loss of image quality that is generally less severe than with replication. However, it is important to note that reducing the size of an image can still result in a loss of detail and image sharpness. Other interpolation techniques, such as bicubic interpolation can produce even better results, but they are more computationally intensive.

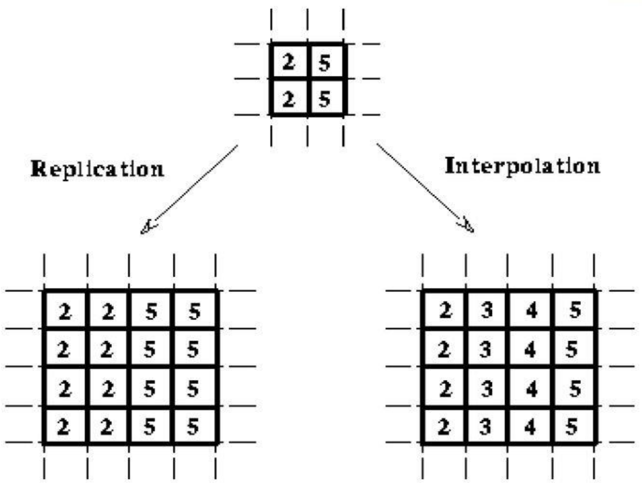


Fig 1. Pixel replication vs. interpolation

Bit Plane Slicing on Images

Bit plane slicing is a technique used in image processing to extract the bit planes of an image. Each pixel in a digital image is represented by a fixed number of bits, usually 8 bits for grayscale images and 24 bits for color images. Each bit plane corresponds to a particular bit position of the binary representation of the pixel values.

To perform bit plane slicing on an image, we extract each bit plane separately, starting from the most significant bit (MSB) to the least significant bit (LSB). This can be done using bitwise operators such as AND and SHIFT.

For example, to extract the MSB bit plane from an 8-bit grayscale image, we perform the following steps:

1. Load the image into memory and obtain its pixel values.

2. Create a new binary image of the same size as the original image.

3. For each pixel in the original image, extract the MSB by performing a bitwise AND operation with

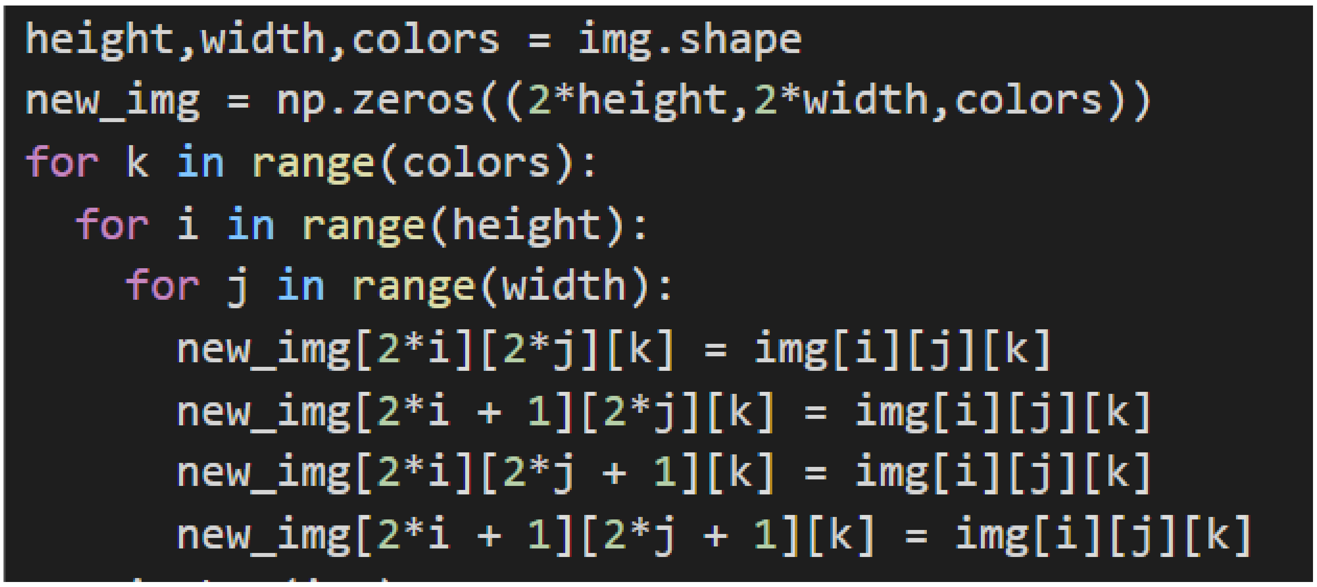
the value 0x80 (which has a binary representation of 10000000).

4. Shift the resulting bit to the least significant position (by 7 bits) to obtain a binary value of 0 or 1.

5. Set the corresponding pixel in the binary image to the binary value obtained in step 4.

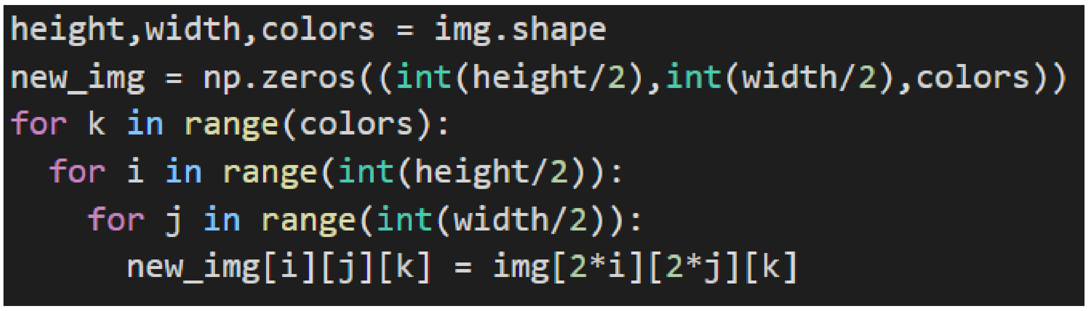
This process is repeated for each bit plane, with the bit position of the AND operation being shifted from the one-bit position to the right for each subsequent bit plane. The result is a set of binary images, each representing a specific bit plane of the original image.

Algorithm for Scaling up using Replication:



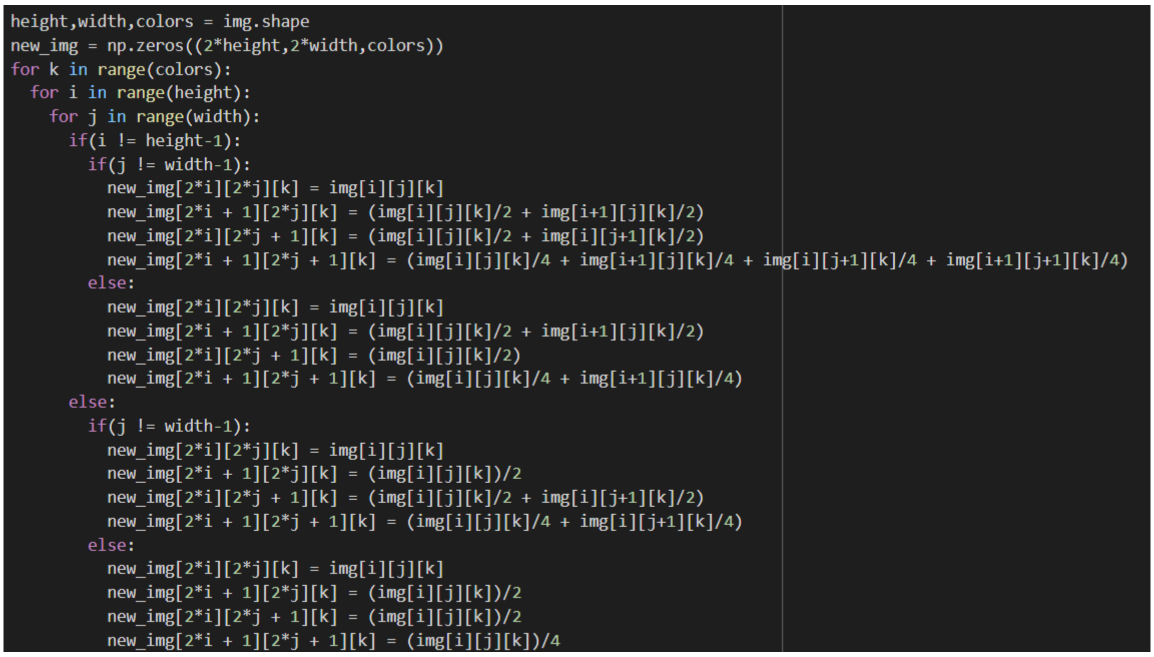
```
height,width,colors = img.shape
new_img = np.zeros((2*height,2*width,colors))
for k in range(colors):
  for i in range(height):
    for j in range(width):
      new_img[2*i][2*j][k] = img[i][j][k]
      new_img[2*i + 1][2*j][k] = img[i][j][k]
      new_img[2*i][2*j + 1][k] = img[i][j][k]
      new_img[2*i + 1][2*j + 1][k] = img[i][j][k]
```

Algorithm for Scaling down using Replication:



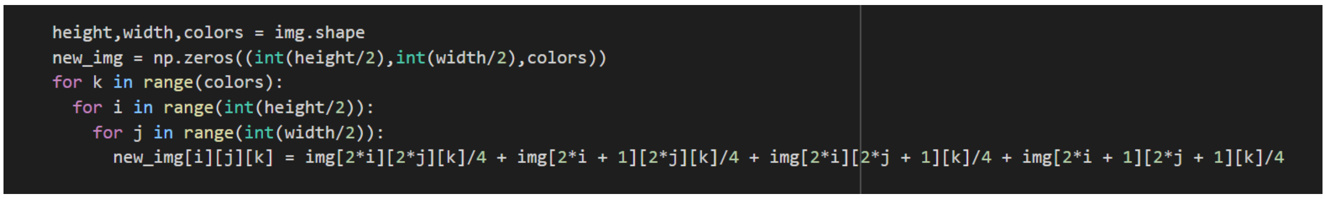
```
height,width,colors = img.shape
new_img = np.zeros((int(height/2),int(width/2),colors))
for k in range(colors):
  for i in range(int(height/2)):
    for j in range(int(width/2)):
      new_img[i][j][k] = img[2*i][2*j][k]
```

Algorithm for Scaling up Using Interpolation



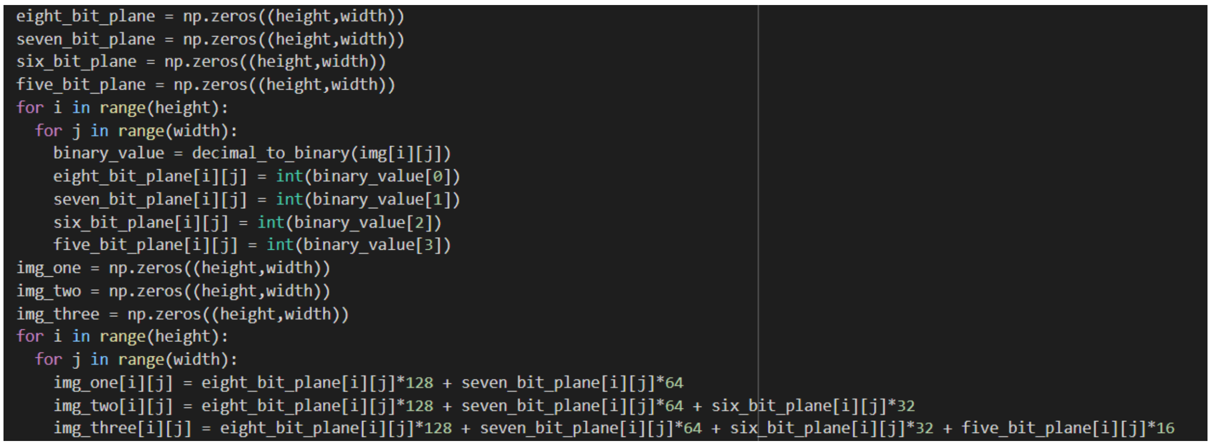
```
height,width,colors = img.shape
new_img = np.zeros((2*height,2*width,colors))
for k in range(colors):
  for i in range(height):
    for j in range(width):
      if(i != height-1):
        if(j != width-1):
          new_img[2*i][2*j][k] = img[i][j][k]
          new_img[2*i + 1][2*j][k] = (img[i][j][k]/2 + img[i+1][j][k]/2)
          new_img[2*i][2*j + 1][k] = (img[i][j][k]/2 + img[i][j+1][k]/2)
          new_img[2*i + 1][2*j + 1][k] = (img[i][j][k]/4 + img[i+1][j][k]/4 + img[i][j+1][k]/4 + img[i+1][j+1][k]/4)
        else:
          new_img[2*i][2*j][k] = img[i][j][k]
          new_img[2*i + 1][2*j][k] = (img[i][j][k]/2 + img[i+1][j][k]/2)
          new_img[2*i][2*j + 1][k] = (img[i][j][k]/2)
          new_img[2*i + 1][2*j + 1][k] = (img[i][j][k]/4 + img[i+1][j][k]/4)
      else:
        if(j != width-1):
          new_img[2*i][2*j][k] = img[i][j][k]
          new_img[2*i + 1][2*j][k] = (img[i][j][k])/2
          new_img[2*i][2*j + 1][k] = (img[i][j][k]/2 + img[i][j+1][k]/2)
          new_img[2*i + 1][2*j + 1][k] = (img[i][j][k]/4 + img[i][j+1][k]/4)
        else:
          new_img[2*i][2*j][k] = img[i][j][k]
          new_img[2*i + 1][2*j][k] = (img[i][j][k])/2
          new_img[2*i][2*j + 1][k] = (img[i][j][k])/2
          new_img[2*i + 1][2*j + 1][k] = (img[i][j][k])/4
```

Algorithm for Scaling down Using Interpolation



```
height,width,colors = img.shape
new_img = np.zeros((int(height/2),int(width/2),colors))
for k in range(colors):
  for i in range(int(height/2)):
    for j in range(int(width/2)):
      new_img[i][j][k] = img[2*i][2*j][k]/4 + img[2*i + 1][2*j][k]/4 + img[2*i][2*j + 1][k]/4 + img[2*i + 1][2*j + 1][k]/4
```

Algorithm for Bit-Plane Slicing



```
eight_bit_plane = np.zeros((height,width))
seven_bit_plane = np.zeros((height,width))
six_bit_plane = np.zeros((height,width))
five_bit_plane = np.zeros((height,width))
for i in range(height):
  for j in range(width):
    binary_value = decimal_to_binary(img[i][j])
    eight_bit_plane[i][j] = int(binary_value[0])
    seven_bit_plane[i][j] = int(binary_value[1])
    six_bit_plane[i][j] = int(binary_value[2])
    five_bit_plane[i][j] = int(binary_value[3])
img_one = np.zeros((height,width))
img_two = np.zeros((height,width))
img_three = np.zeros((height,width))
for i in range(height):
  for j in range(width):
    img_one[i][j] = eight_bit_plane[i][j]*128 + seven_bit_plane[i][j]*64
    img_two[i][j] = eight_bit_plane[i][j]*128 + seven_bit_plane[i][j]*64 + six_bit_plane[i][j]*32
    img_three[i][j] = eight_bit_plane[i][j]*128 + seven_bit_plane[i][j]*64 + six_bit_plane[i][j]*32 + five_bit_plane[i][j]*16
```