CS576 Multimedia System Design

Assignment#1

Submission Date: 09/15/2017

Monika Devanga Ravi

USC ID - 3448881178

Programming Part

Instructions for compiling and running the program

Program Name: imageReader

To compile the program type "*javac imageReader.java*" in command prompt

To run the program type "*java imageReader in_img_name in_img_width in_img_height resampling_method out_size_img*"

Here:

- in_img_name is the name of the input image file
- in_img_width and in_img_height are the width (4000/400) and height (3000/300) of the input image
- resampling method is the type of sampling method and it takes values 1 or 2;

  For downsampling an image: resampling methods are –

  1. Specific sampling.
  2. Gaussian smoothing.

  For upsampling an image: resampling methods are –

  1. Nearest neighbor sampling.
  2. Bilinear interpolation.
- out_img_size is the size of the output image. Sizes are given by O1/O2/O3

  O1: 1920 x 1080

  O2: 1280 x 720

  O3: 640 x 480

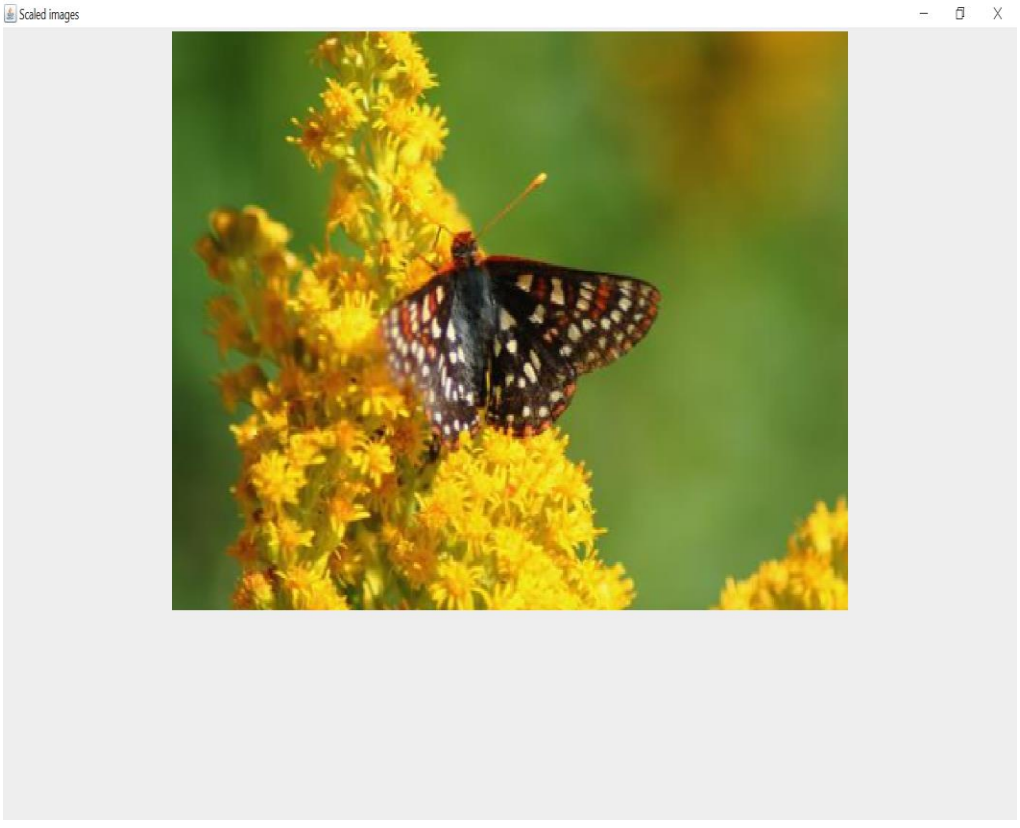
For example: To downsample an input image (hw_1_high_res.rgb) with input size 4000 x 3000 to 1920 x 1080 use the following command:

java imageReader hw_1_high_res 4000 3000 1 O1

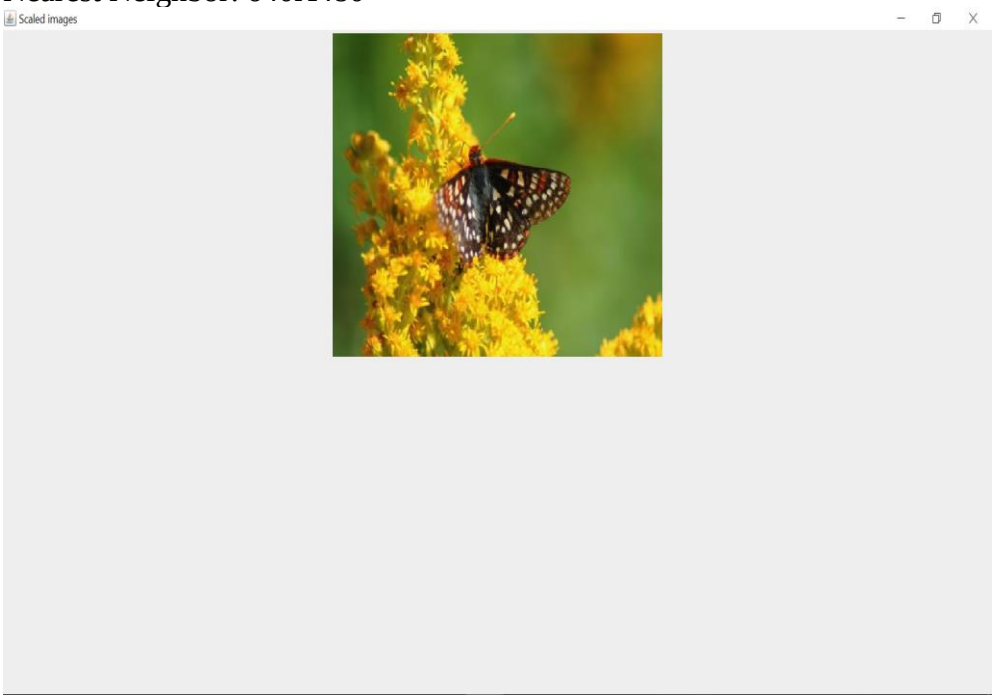
Output:

- Nearest Neighbor: 1920X1080

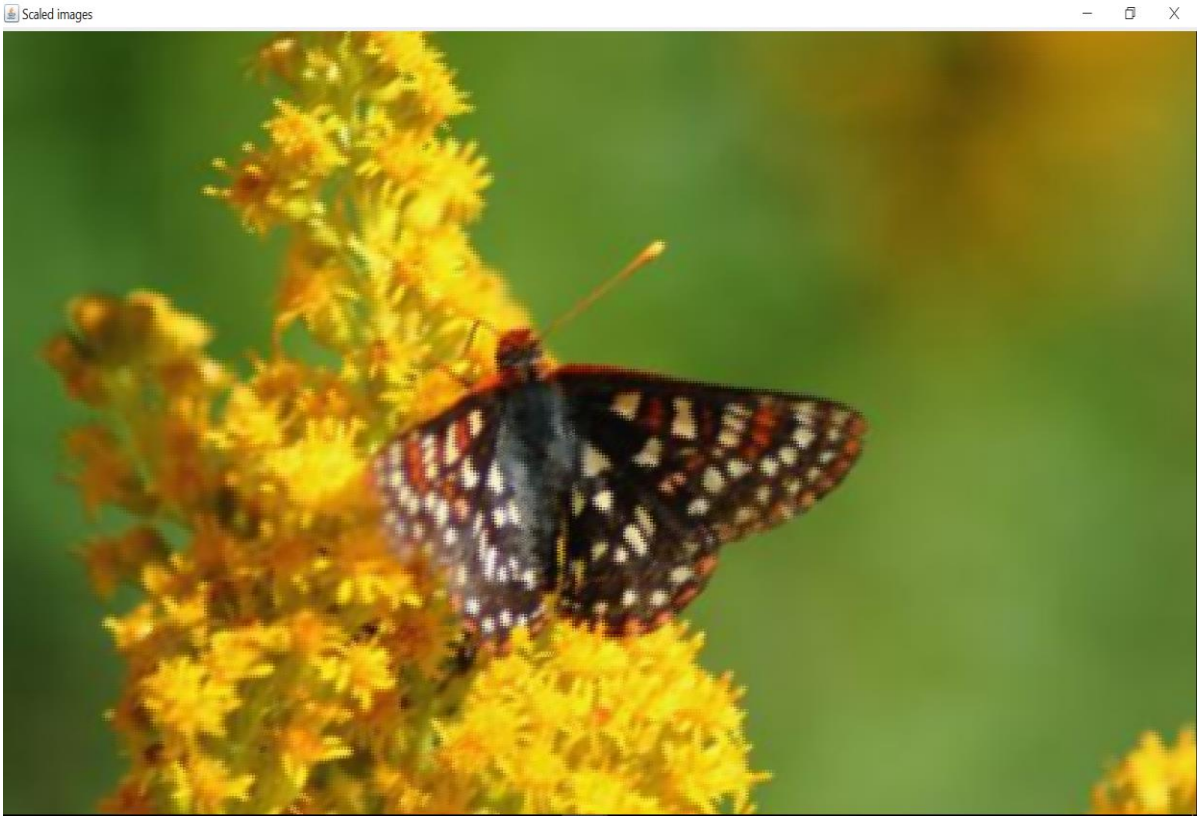
Nearest Neighbor: 1280x720



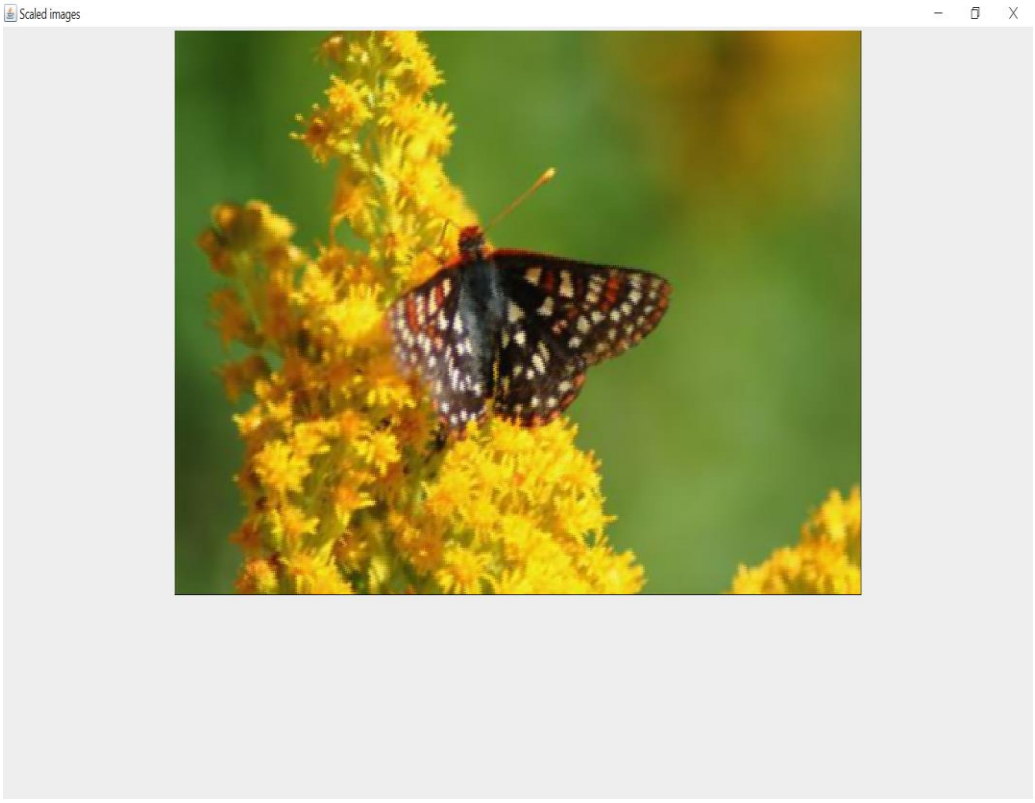
Nearest Neighbor: 640X480



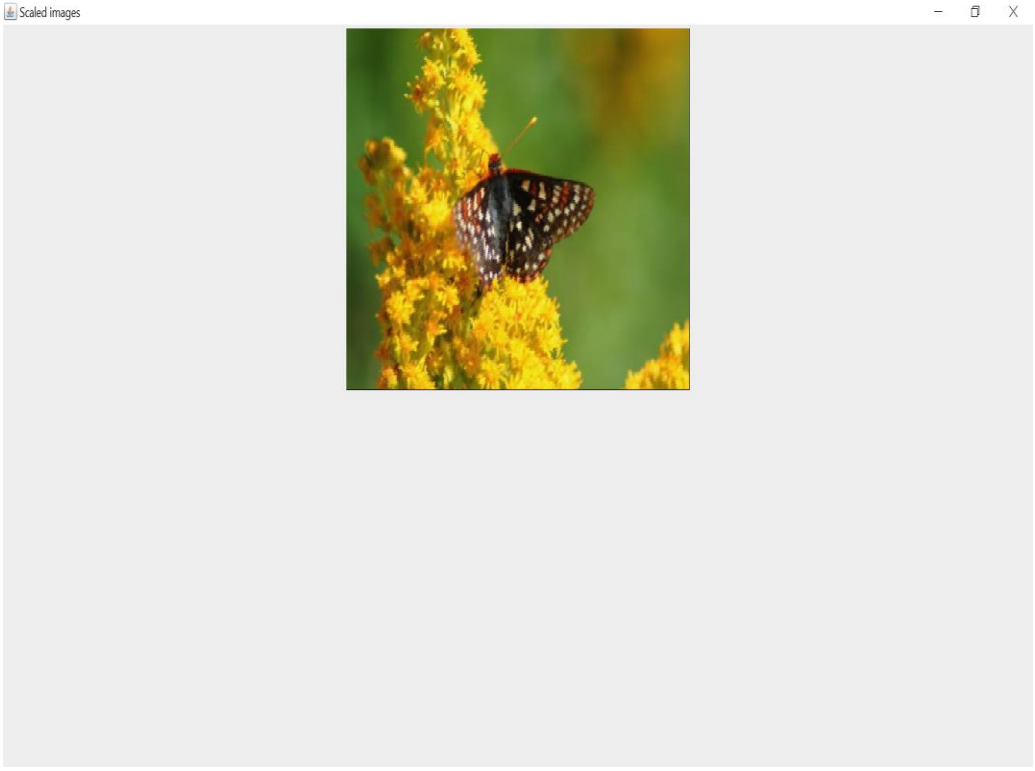
- Bilinear Interpolation:



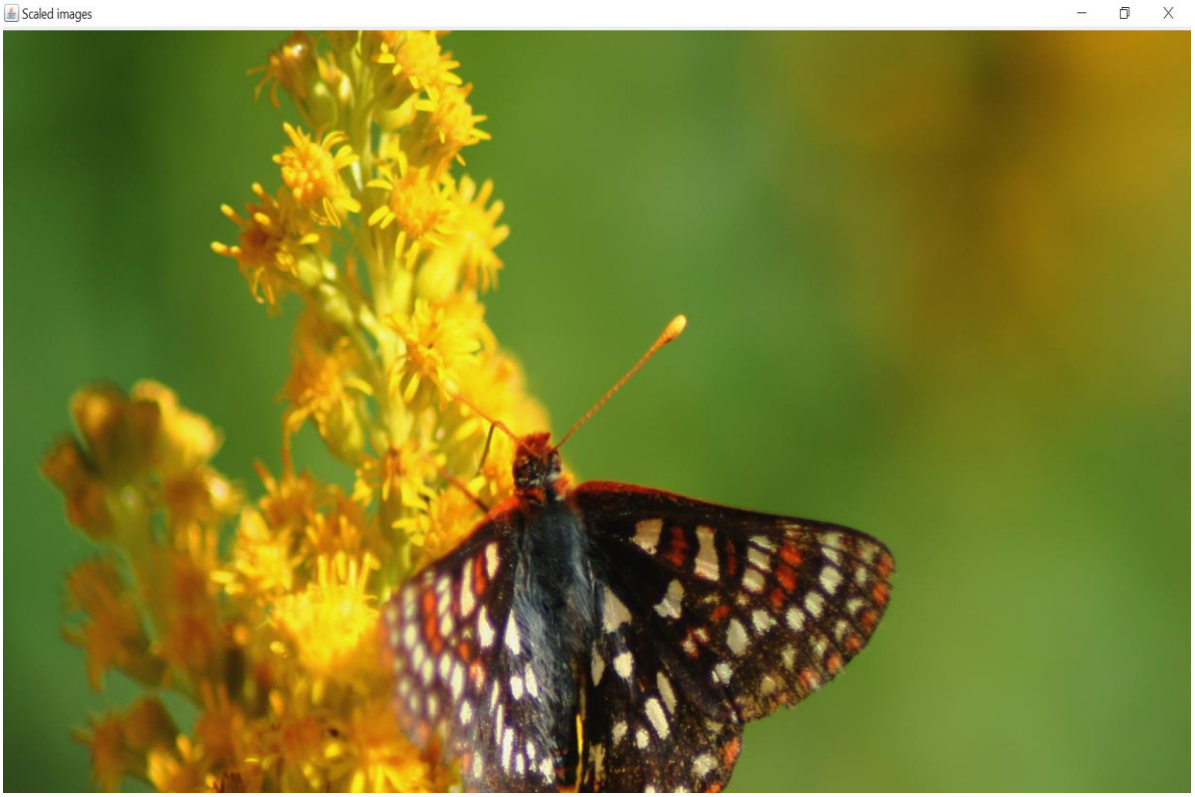
Bilinear Interpolation: 1280 x 720



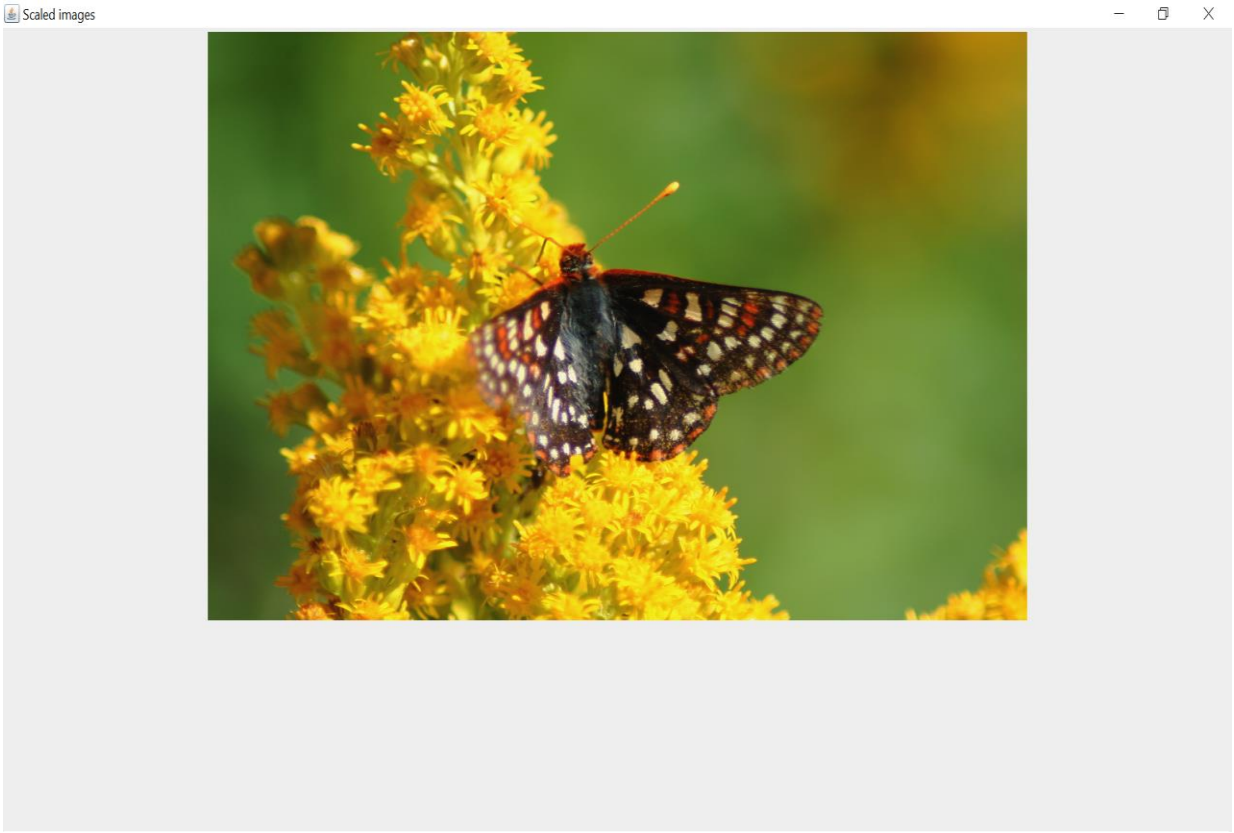
Bilinear Interpolation: 640x480



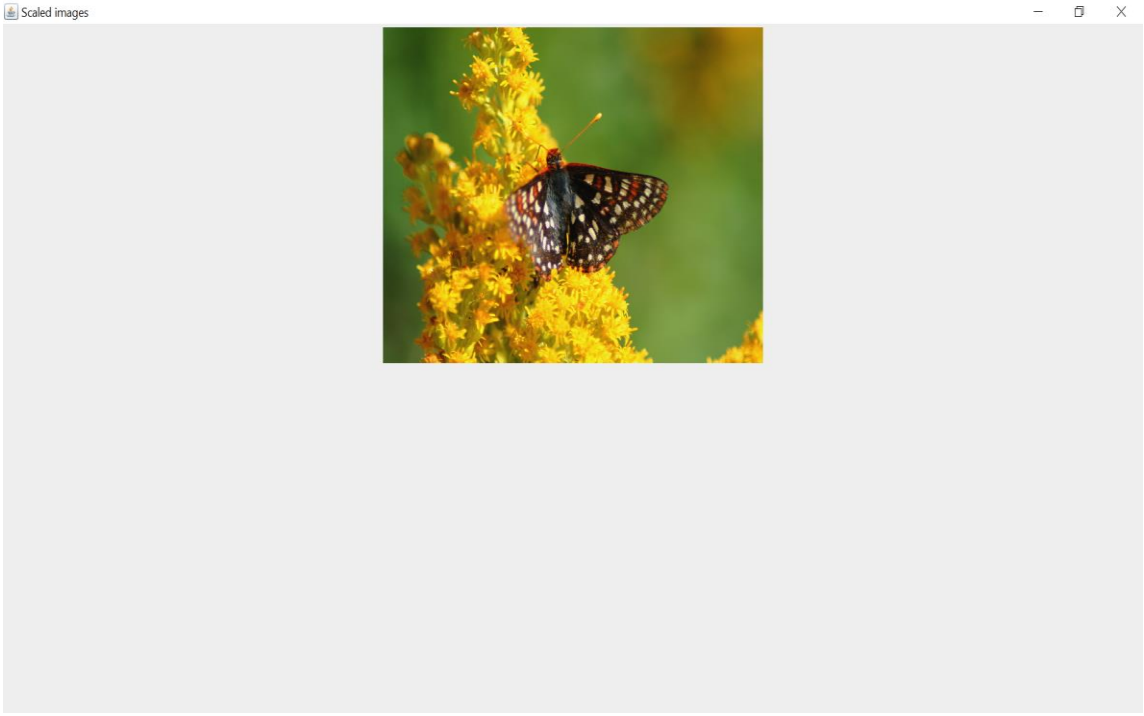
- Specific sampling: 1920x1080



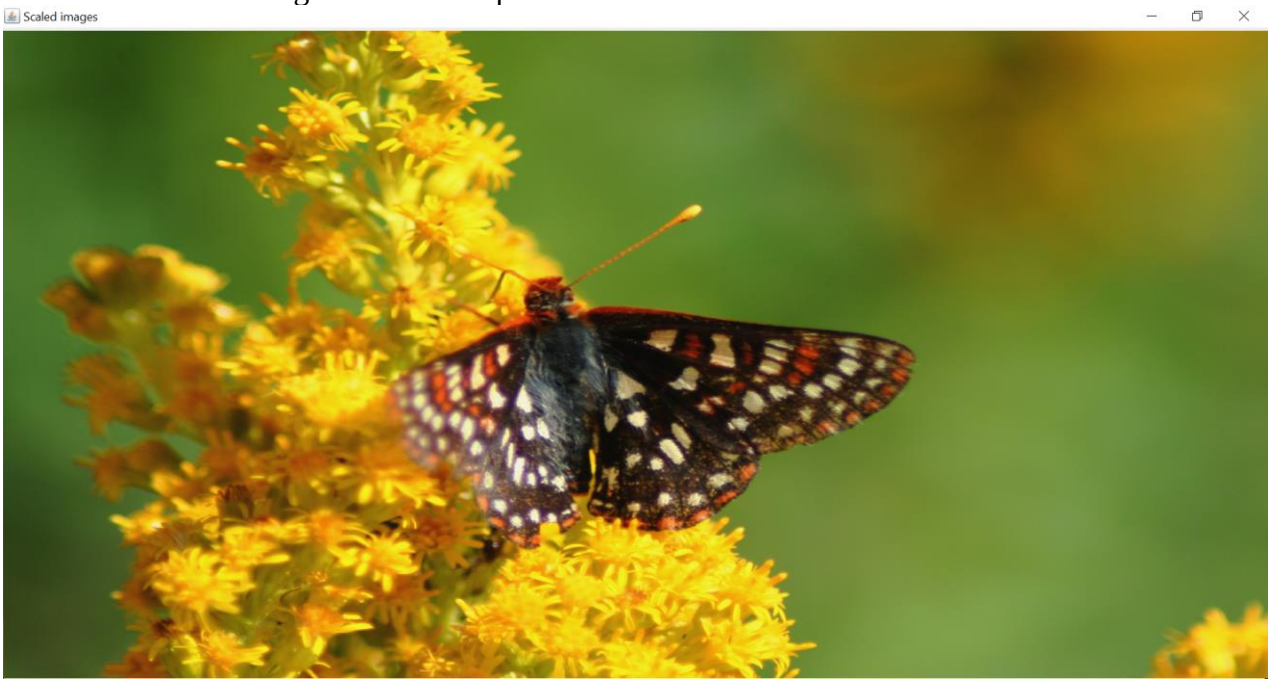
Specific sampling: 1280x720



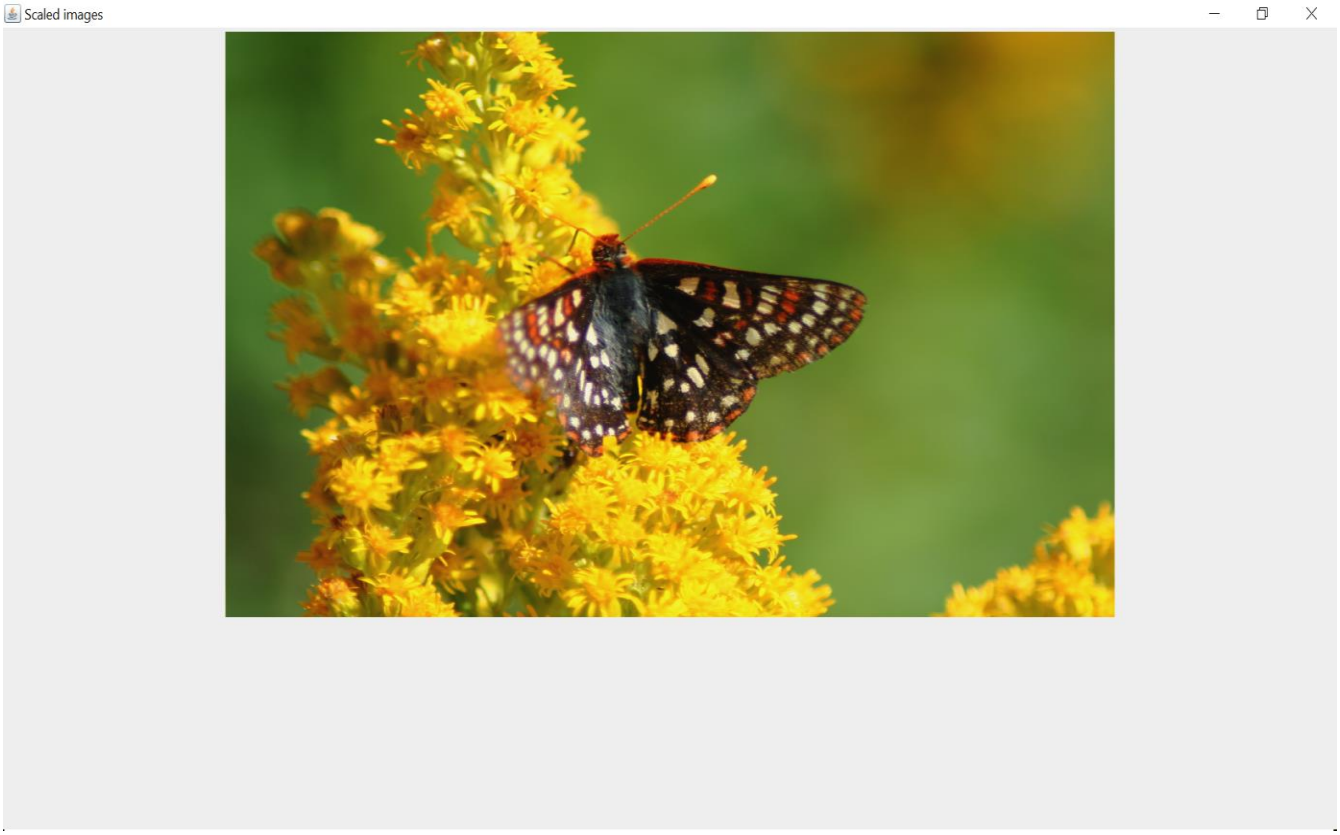
Specific sampling: 640x480



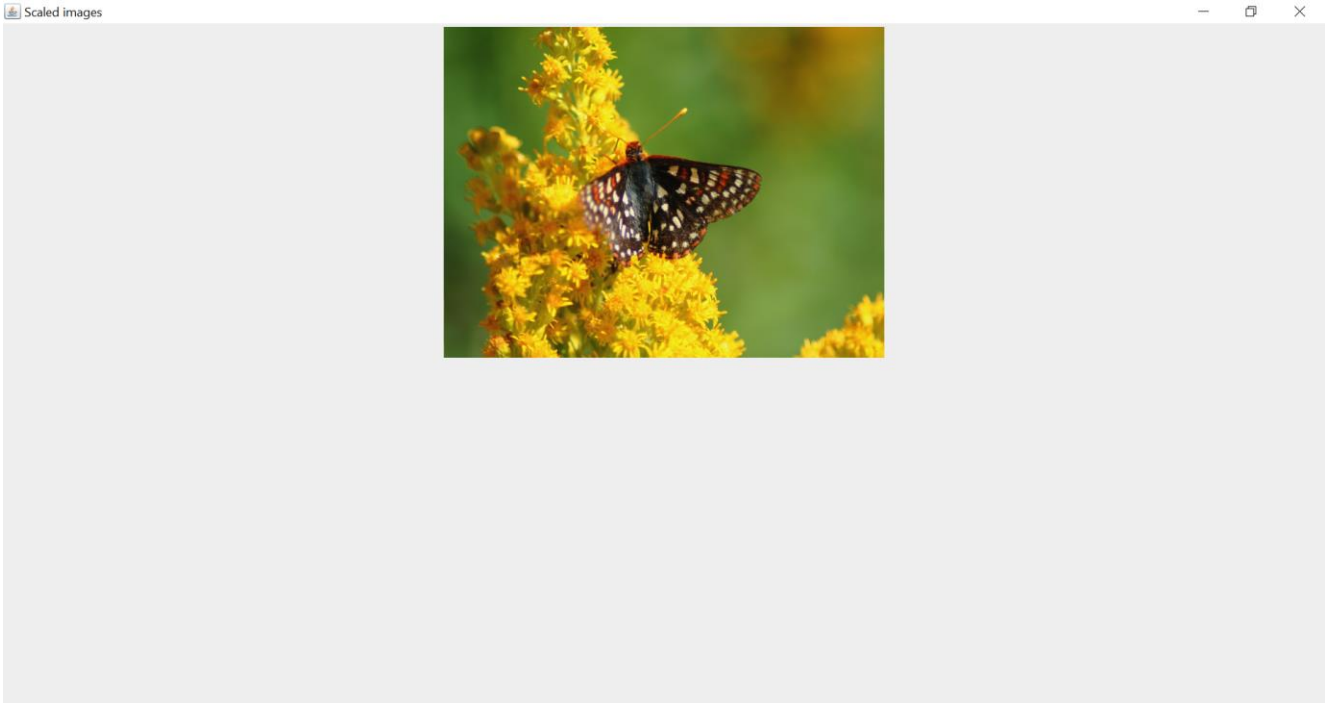
- Gaussian average of set of samples: 1920x1080



Gaussian average of set of samples: 1280x720



Gaussian average of set of samples: 640x480



Discussion Part

1. All the outputs here are different from the 4:3 aspect ratios, resulting in a change in pixel aspect ratios in the output. This will either cause pixel stretching or compression which are undesired effects. Propose and implement at least **two methods** to eliminate, or at least minimize the change in pixel aspect ratio. You do not need to submit your program which does this but please attach outputs of your method(s) on high res and low-res image samples in your submission document.

Here, I describe two methods for maintaining the aspect ratio of the image:

1. Pillarboxing / letterboxing
2. Fixed width

Method 1: Pillarboxing / letterboxing

Pillarboxing/Letterboxing technique is used to display images/movies on a wide screen by adding vertical/horizontal bars on the sides.

It works as follows: first take the ratio of old_image width and new screen width, also take the ratio of old_image height and new screen height. Then, scale the image based on the highest ratio among them. The remaining unfilled blank areas in the new image is filled with some background color.

For example, if we want to resize 4000 * 3000 image to 1920 * 1080 image, the width ratio is 2.083 and the height ratio would be 2.77. Now,

Scaling the image by ratio of 2.77 gives a 1440 * 1080 image which has same aspect ratio as original image(4000x3000), for the remaining pixel i.e., (1920-1440 = 480) fill with background color.

Below are the output images obtained after pillarboxing an high resolution image (4000x3000) to a low-resolution image.

Image 1:

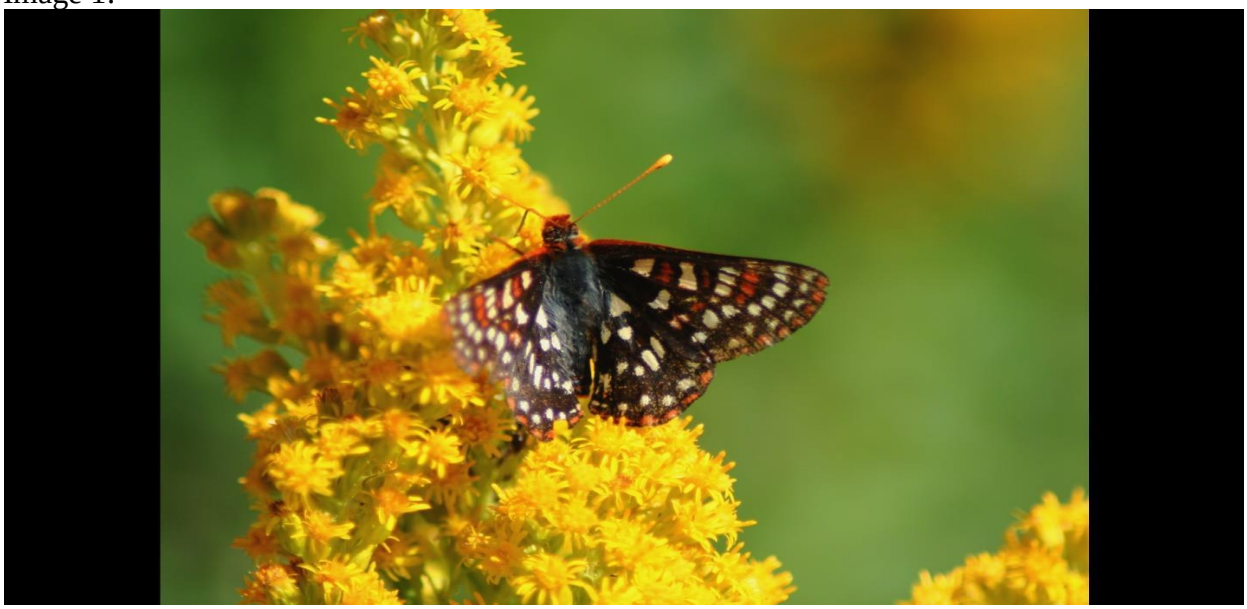


Image 1.1 high_low_1920x1080

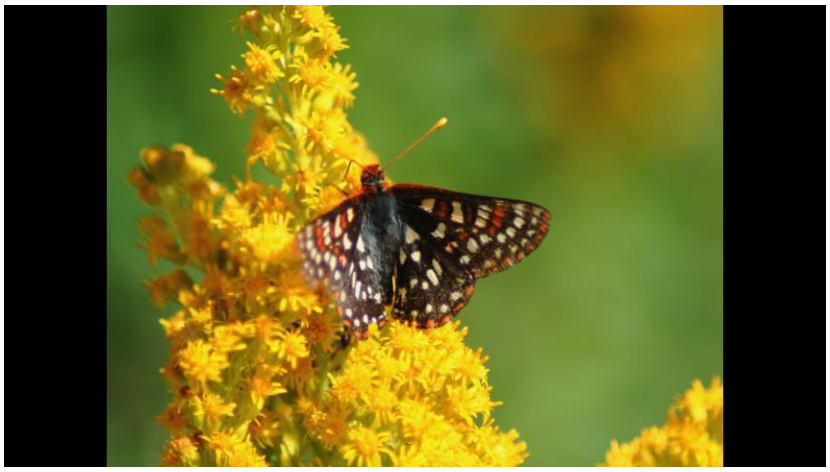


Image 1.2 high_low_1280x720

Image 1.3 high_low_640x480

Image 1.4 high_low_1920x1080

Image 1.5 high_low_1280x720

Image 1.6 high_low_640x480

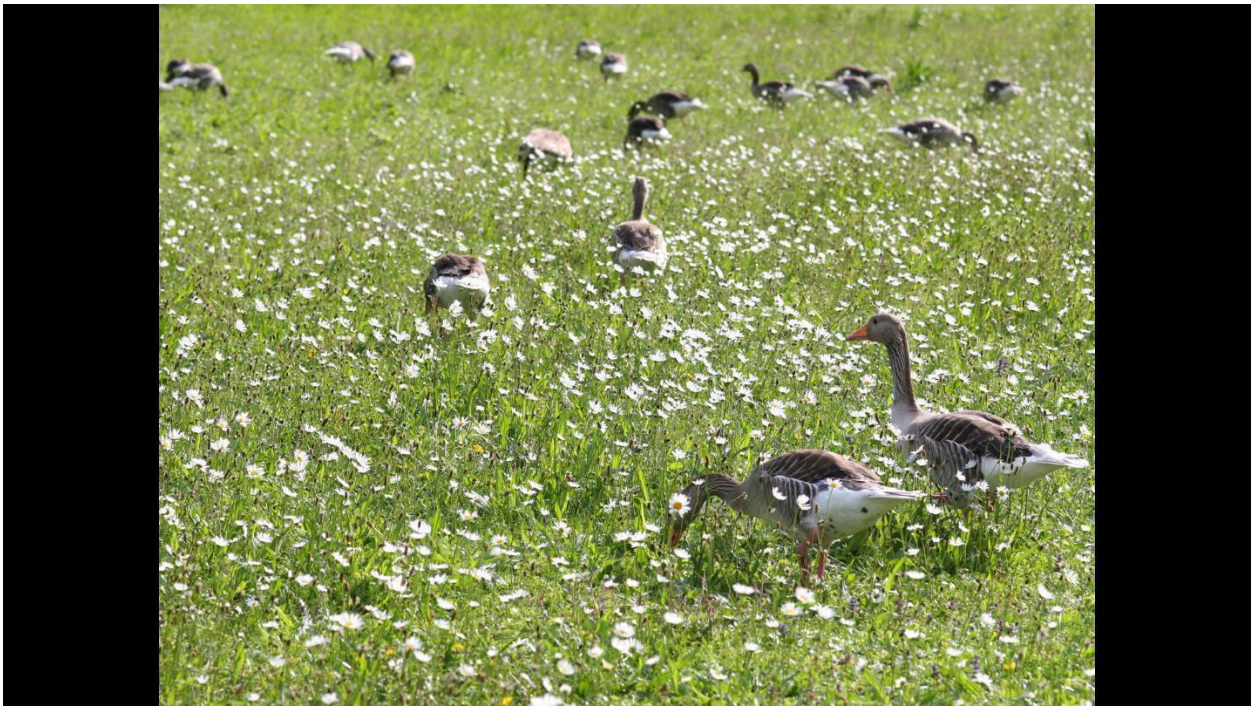


Image 1.7 high_low_1920x1080

Image 1.8 high_low_1280x720

Image 1.9 high_low_640x480

Image 1.10 high_low_1920_x_1080

Image 1.11 high_low_1280x720

Image 1.12 high_low_640x480

Below are images for scaling low resolution image (400x300) to high-resolution image

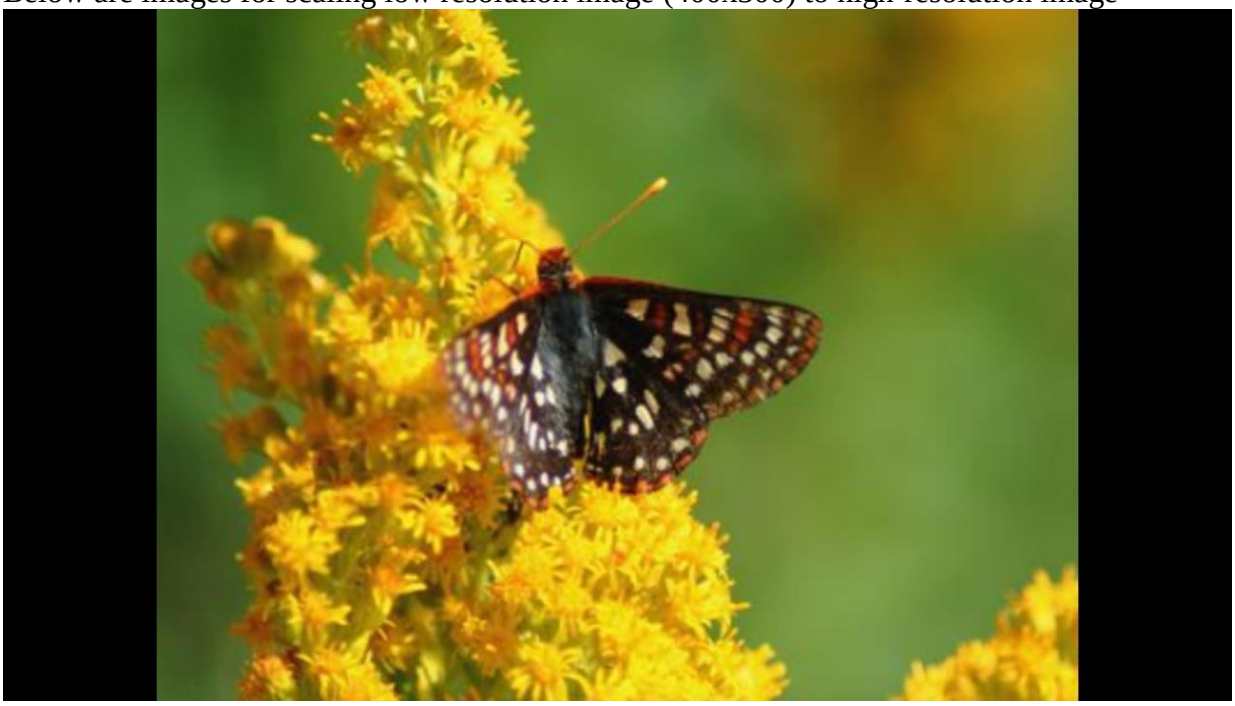


Image 1.13 low_high_1920x1080

Image 1.14 low_high_1280x720

Image 1.15 low_high_640x480

Image 1.16 low_high_1920x1080

Image 1.17 low_high_1280x720

Image 1.18 ow_high_640x480

Image 1.19 low_high_1920x1080

Image 1.20 low_high_1280x720

Image 1.21 low_high_640x480

Image 1.22 low_high_1920x1080

Image 1.23 low_high_1280x720

Image 1.24 low_high_640x480

Method 2: Fixed width / fixed height

To get the image with fixed width:

Fix the width of the image then, calculate the height of the image based on the new_width/old_width ratio to maintain the aspect ratio.

For example:

Input image size is 400x300, Aspect ratio – 4:3

New image size is 1920x1080, Aspect ratio – 1.777:1

Here new width needed is 1920px then,

get the ratio of new_width/old_width = 1920 / 400 = 4.8

multiple the old height with ratio calculated above = 300 x 4.8 = 1440

This will increase the height of the image more than the required size but aspect ratio is maintained.

Below is the output of images from Low-res(400x300) to high-res:

Required size: 1920x1080 - Image is stretched to 1920x1440

Required size: 1280x720 - Image is stretched to 1280x960

Required size: 640x480 - Image is 640x480

Required size: 1920x1080 - Image is stretched to 1920x1440

Required size: 1280x720 - Image is stretched to 1280x960

Required size: 640x480 - Image size is 640x480

Required size: 1920x1080 - Image is stretched to 1920x1440

Required size: 1280x720 - Image is stretched to 1280x960

Required size: 640x480 - Image size is 640x480

Required size: 1920x1080 - Image is stretched to 1920x1440

Required size: 1280x720 - Image is stretched to 1280x960

Required size: 640x480 - Image size is 640x480

---

2. As shown in class, seam carving is another smarter way to resize your image that takes the content of the image into account. Read the paper available here (http://perso.crans.org/frenoy/matlab2012/seamcarving.pdf). Download the code to perform scene carving (http://code.google.com/p/seam-carving-gui/), compile it and run it on the images given to you and attach them in your submission. Comment on the results – the pros and especially the cons. Where do you think the method performs well or does not perform well?

   Ans: First, we report the resizing of images using seam carving method, and then later the pros and cons of the method is discussed in detail.

   Below are the results of applying seam carving method to down sample a high-resolution image 4000 x 3000 to low-resolution images.

Image 2.1 high_low_1920x1080

Image 2.2 high_low_1280x720

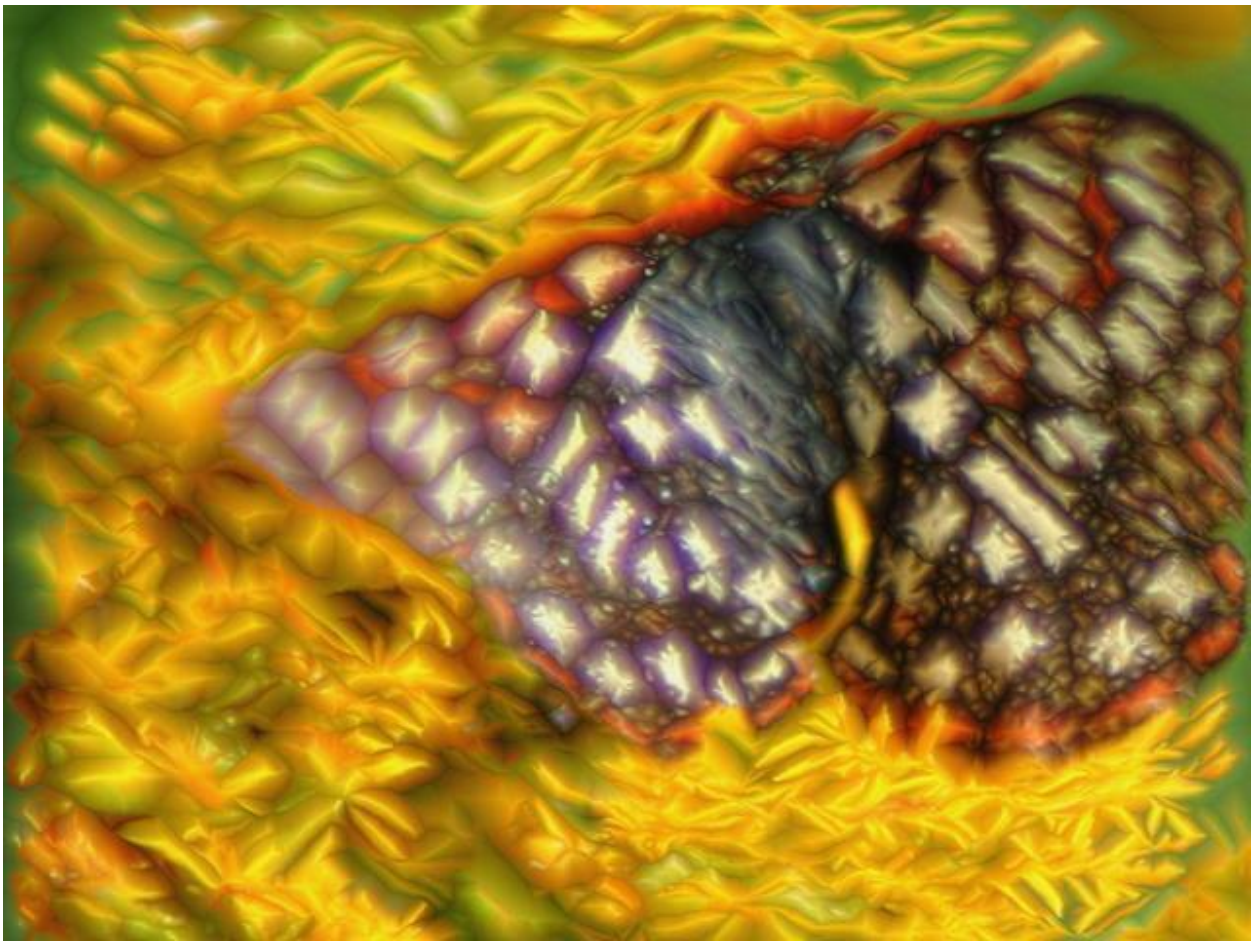


Image 2.3 high_low_640x480

Image 2 :

Image 2.4 high_low_1920x1080

Image 2.5 high_low_1280x720

Image 2.6 high_low_640x480

Image 3:

Image 2.7 high_low_1920x1080

Image 2.8 high_low 1280x720

Image 2.9 high_low_640x480

Image 4

Image 2.10 high_low_1920x1080

Image 2.11 high_low_1280x720

Image 2.12 high_low_640x480

Below are the results of applying seam carving method to up-sample a low-resolution image to a high-resolution image.
Image 1:

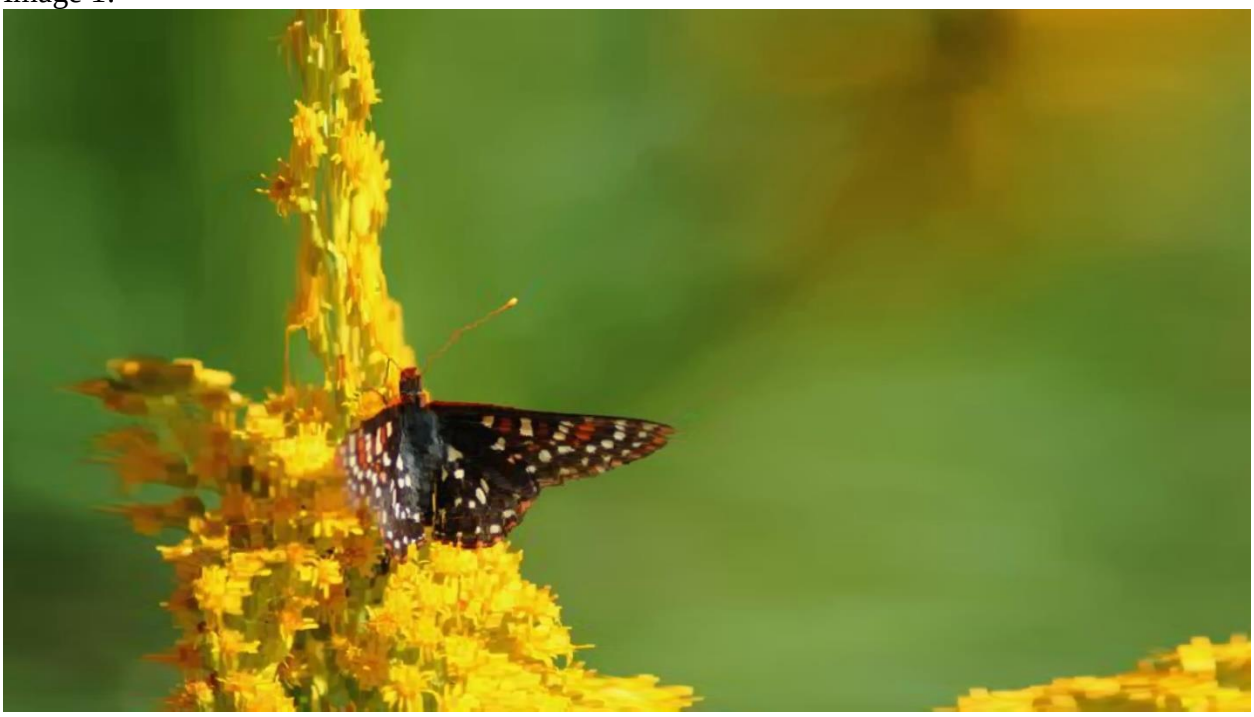


Image 2.13 low_high_1920x1080

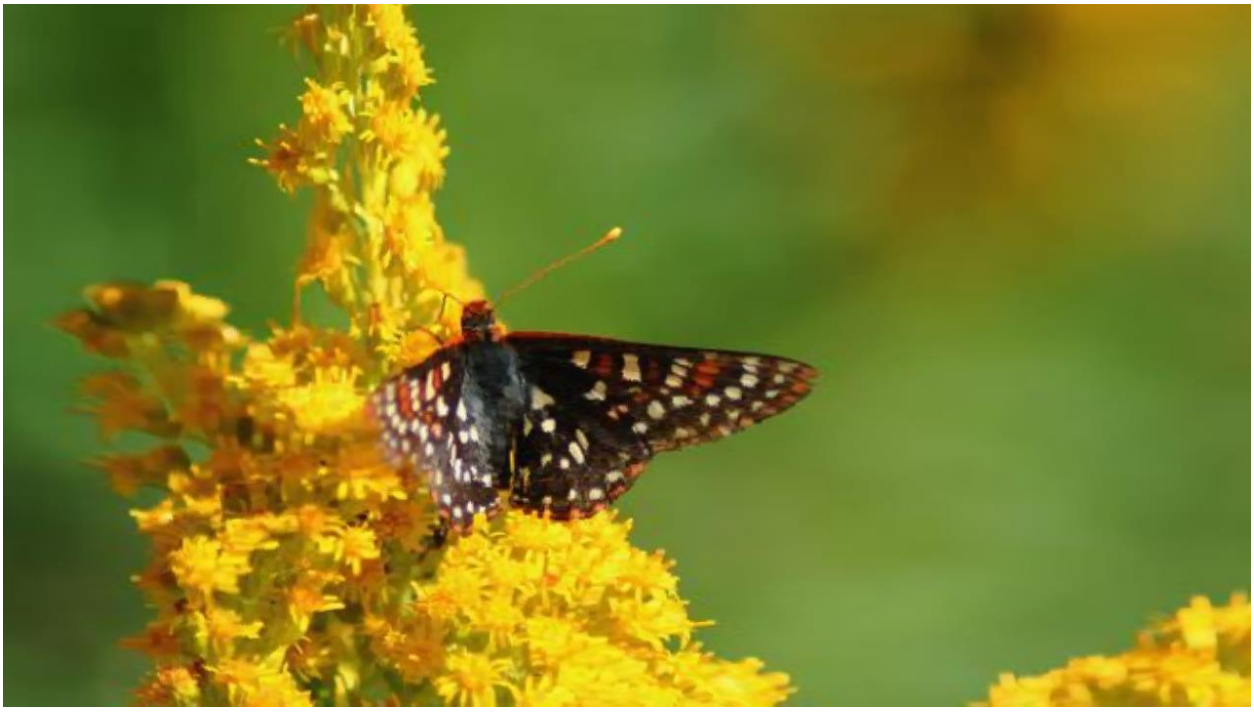


Image 2.14 low_high_1280x720

Image 2.15 low_high_640x480

Image 2.16 low_high_1920x1080

Image 2.17 low_high_1280x720

Image 2.18 low_high_640x480

Image 2.19 low_high_1920x1080

Image 2.20 low_high_1280x720

Image 2.21 low_high_640x480

Image 2.22 low_high_1920x1080

Image 2.23 low_high_1280x720

Image 2.24 low_high_640x480

Advantages:

- The main advantage of resizing with seam carving is that it preserves the main features in the image and only removes the unimportant pixels in the image. From the downscaling examples above, image in 2.1 shows that most of the pixels of the butterfly are retained and removes the surrounding green background of the image, in 2.4 sky part is removed and train is retained and in 2.10 huts are retained and sky part is removed.
- Another advantage of seam carving is seen during up-sampling an image. It inserts the unimportant pixels in the image instead of just stretching the whole image uniformly. From the above pictures, all the images from 2.13 – 2.24 shows up-sampling of 400x300 images to different sizes. In picture 2.14 butterfly is the main feature of the image which is kept without any distortion, surrounding green part is added while up-scaling an image. Similarly, for 2.16, pixels of the sky region is inserted and the landscape are preserved.

Disadvantages:

- Seam carving doesn't work well all the time. While reducing the size of image, it introduces some deformation of the main feature itself. For example: in the above down-sampling images (2.1-2.12), the flower in the 2.3 image is completely out of shape, 'antenna' of the butterfly doesn't look like antenna. We can see deformation of the main feature in 2.6 clearly, the train is completely out-of-shape and grass surrounding the train become more dominant. We observe that as we down-sample the image more, the deformation of the image increases.
- Seam carving method is not able to distinguish between main feature of the image and its surroundings. For example, as shown in the image 2.12 – hut which is the main feature of the image is completely deformed and grass becomes more dominant. In image 2.9, the bird shapes are not maintained, and while we see that bird and grass look similar.
- While enlarging an image, seam-craving may introduce artifacts. If the original image has smooth background (without much changes both in shapes and colors), the result of enlarged image shows obvious blurriness in the background. In image 2.13, the flower in the background is enlarged and blurred. We can see some block artifacts in 2.22 image in clouds. In the image 2.16, the train engine appears stretched.

From the above seam-craving results, we observe that the up-sampling of images works better than down-sampling.

I also tried to get better results by specifying the parameters such as HD Quality and Forward energy in the seam-carving GUI. Choosing these settings did not result in better outcomes as shown in the image 2.25. Moreover, this setting takes a lot of time (~25-30 mins) to resize a single image.

Here is one of the output after resizing the image to 640x480:
Disadvantage here is: Very time consuming and Butterfly looks as they are stretched and flower is completely distorted. High energy pixels are maintained but they are not smooth.

Image 2.25 Downsampling with HD quality and forward energy features.

---

3. You will notice that the outputs of up sampling to a specific resolution are worst off in quality compared to down sampling to the same resolution. This should be no surprise because in the latter case you have samples to decide and choose from where as in the former case you are interpolating and "imagining" what a best output might be. Research into improving this and propose an intelligent way to up sample and get better results. Hint - you may find it useful pointers by googling "Super - resolution from a single image".

   Answer: Upsampling to a specific resolution leads to poor quality images when compared to downsampling to the same resolution. Below, we discuss some Super-resolution ideas to improve the upsampling of images.

   Image super-resolution is the task of inferring a high-resolution image from one or many low-resolution images. In Super-resolution (denoted by SR), the goal is to recover new missing high-resolution details that are not explicitly found in any individual low-resolution images. There are many ways to get super resolution images. Some of them are:

a. Classical multi-image SR- the high-resolution pixel information needed for output image is assumed to be split across multiple low-resolution images.
b. Example-based SR- the high resolution pixel information needed for output image is assumed to be available in the high-resolution database patches, and learned from the low-resolution/high-resolution pairs of examples in the database

Borrowing the idea from [1], we can obtain a super resolution image by combining the power of the classical and example-based SR techniques.

Natural images contain repetitive visual contents. For example, small (5x5) patches in a natural image redundantly recur many times inside the image, within the same scale as well as across different scale images as shown in below image (Fig1). From this observation, we can obtain a super-resolution image using the patch redundancy ideas discussed below.

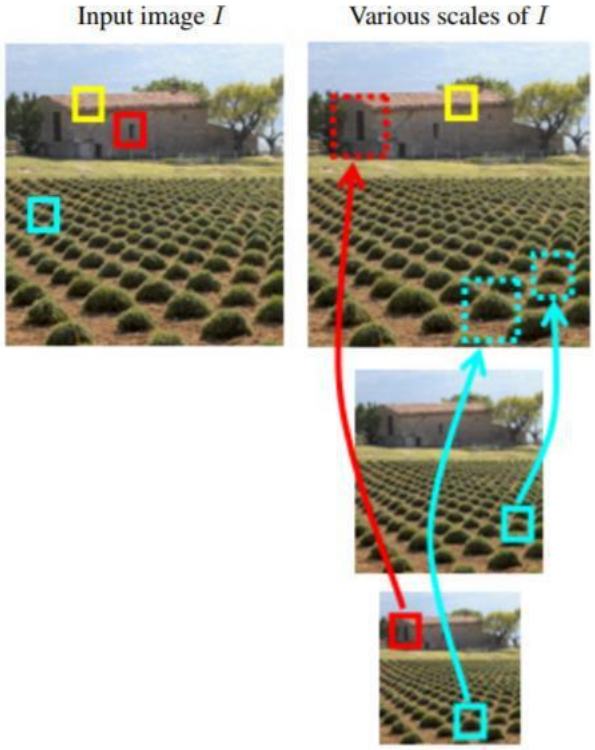


Fig1 - Patch recurrence within and across scales of a single image

Employing in-scale patch redundancy:

Recurrence of patches within the same image scale is a basis for applying the Classical SR constraints to information from a single image. When there is only a single low-resolution image, the problem of recovering high-resolution image becomes under-determined, as the number of constraints induced by low-resolution image is smaller than the number of unknowns in high-resolution image. Nevertheless, as we know there are plenty of patch redundancies within the same low-resolution image, these patches can be considered as if taken from k different low-res images of the same high-resolution scene, thus inducing k times more linear constraints. For increased numerical stability, each equation induced by a patch is globally scaled by the degree of similarity to its source patch. Thus, patches of higher similarity to patch will have stronger influence on the recovered high-res pixel values than patches of lower similarity.

This idea can be translated to the following simple algorithm:
For each pixel in image L, find its nearest patch neighbors in the same image L (using Nearest neighbor algorithm) and compute their subpixel alignment. Assuming sufficient neighbors are found, this process results in a determined set of linear equations on the unknown pixel values in output image. Globally scale each equation by its reliability and solve the linear set of equations to obtain high-res/super resolution image (as shown in Fig2).

The above procedure allows one to extend the applicability of the classical SR to a single image. However, this process still suffers from the same inherent limitations ([2,3]) of the classical multi-image SR. To overcome this, we can leverage the ideas of Example-based SR by learning the correspondences between low and high resolution image patches across different image scales, and applying the cross-scale patch redundancy techniques explained below.

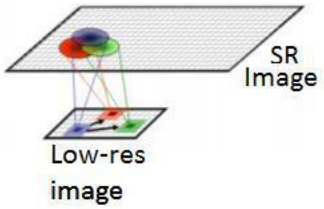


Fig2 – SR image from in-scale patch redundancy

Employing cross-scale patch redundancy:
In example-based SR, correspondences between low and high-resolution image patches are learned from a database of low and high-resolution image pairs, and then applied to a new low-resolution image to recover its most likely high-resolution version. Traditionally, for Example-based SR, we need multiple images with different scales.

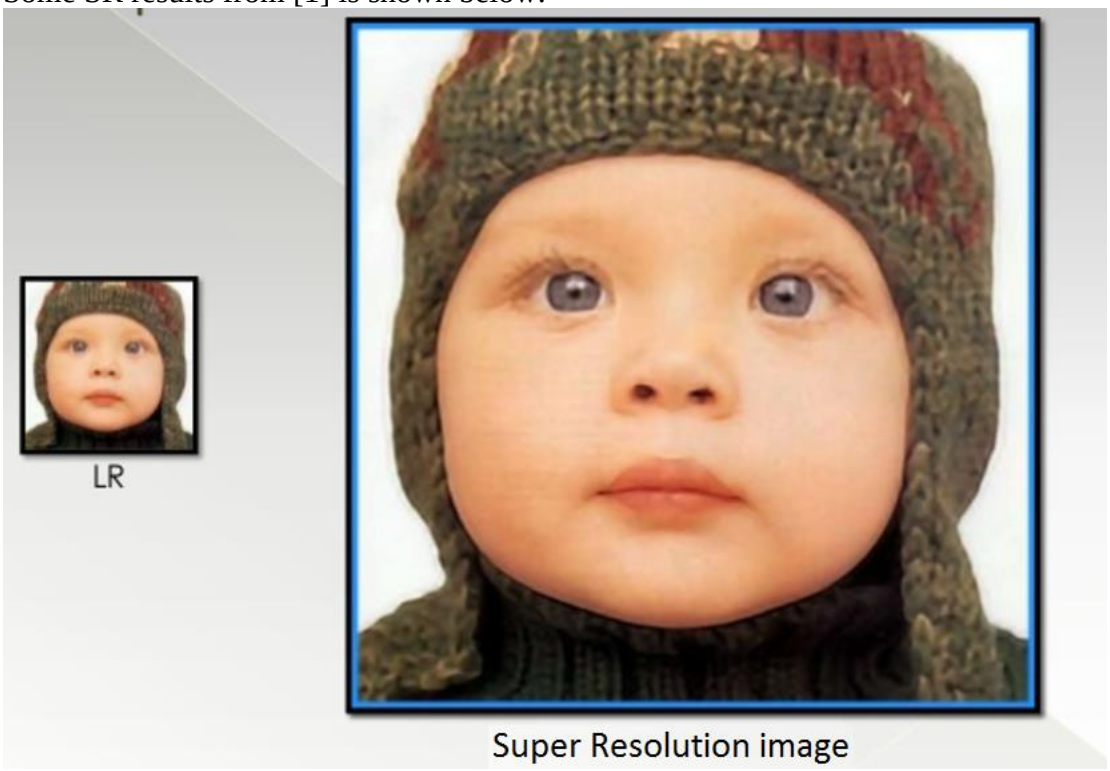
Example-based SR technique can be applied for a single image, by producing multiple high and low-resolution images at different scales. Thus, the low-res/high-res patch correspondences can be learned directly from the image itself, by employing patch repetitions across multiple image scales. The procedure is as follows: First, produce the cascade of low-resolution and high-resolution images using a set of blur functions (eg. Gaussian blur). Next, search a patch P needed for output image given this cascade of low-resolution and high-resolution images. Then, copy that patch from the high-resolution image using the high-resolution/low-resolution patch pair.

Combining above mentioned two patch redundancy techniques into an unified framework, we can obtain super-resolution image from a single image, as discussed below.

Combining classical and example based SR:
Repeating cross-scale patch redundancy process for all pixels in a low-resolution image will yield a large collection of suggested high-resolution patches that range between low-resolution and high-resolution images. Each such learned high-resolution patch induces linear constraints on the unknown target high- resolution image. These constraints are in the form of linear classical SR constraints and each such linear constraint is globally scaled by its reliability. The closer the learned patches are to the target high-resolution image, the better it is the conditioned of resulting set of equations. By solving the resulting set of equations, we can get SR image.
Note that, for a pixel, if the only similar patches are found within the input low-resolution image, then this scheme reduces to the classical single-image SR at that pixel; and if no similar patches

are found, this scheme reduces to simple deblurring Example based SR at that pixel. Thus, the above scheme guarantees to provide the best possible resolution image.

Some SR results from [1] is shown below:



(a) Input image (scaled for display). (b) Bicubic interpolation (×2). (c) Within image repetitions (×2). (d) Unified single-image SR (×2).

Input image, scaled for display (overlay top left), bicubic interpolation ×3 (left) and SR ×3 result (right).

References:

[1] Glasner, Daniel, Shai Bagon, and Michal Irani. "Super-resolution from a single image." Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009.

[2] Lin, Zhouchen, and Heung-Yeung Shum. "Fundamental limits of reconstruction-based superresolution algorithms under local translation." IEEE Transactions on Pattern Analysis and Machine Intelligence 26.1 (2004): 83-97.

[3] Baker, Simon, and Takeo Kanade. "Limits on super-resolution and how to break them." IEEE Transactions on Pattern Analysis and Machine Intelligence 24.9 (2002): 1167-1183.