Assignment 2 Report
Name: Ramya P Narayanaswamy
Netid : rpn2

Part1

Implementation details : The code is divided into functions for easy readability. The loading, saving, plotting image functions are self-explanatory. Images are converted to greyscale. The filter_lp and filter_hp apply the low pass and high pass filter functions on images. Both the filters used *scipy.ndimage.gaussian_filter* as suggested. The high pass filtered output had negative values initially due to subtraction of low pass filtered image from the original image. The absolute value of minimum value (i.e negative value) is added and the filter result is clipped to be in range [0,1]. Hybrid image is created by averaging the values of low pass and high pass filters and clipping the result to be in the range [0,1].

Choice of Sigma : The value of sigma for high pass filter is always greater than the low pass filter. This is because, sharper details (high frequency components) are obtained by subtracting the original image from low pass filtered image. Having a higher sigma, enables to have smoother low pass, which in turn produces sharp details , for subsequent high pass filtered image.

Visualization of hybrid image: For visualizing the hybrid image, a visualize hybrid image function from https://www.cc.gatech.edu/~hays/compvision/proj1/ was reused. This helper code was useful in visualizing the hybrid image at different resolution, mimicking the viewing distance artifact.

Sigma tuning technique :
For each image and for each filter , sigma is tuned by incrementing in folds of two initially(coarse tuning) and once a satisfactory value if reached, minor fine tuning is performed for both high pass and low pass filters.

Comment on Results : The details are visible at a short distance, which implies high pass filtered image is dominant in original hybrid image. The blurred image is visible at longer viewing distance, which implies low pass filtered image is dominant in resized hybrid images.

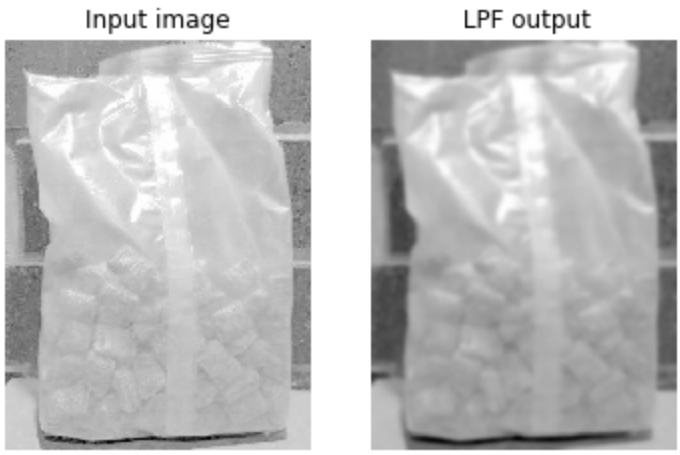
Abbreviations used : :LPF - Low Pass Filter, HPF - High Pass Filter

Example1:

Bag of cereal (for LPF) and Cereal box (for HPF) :

Input images

LPF (Sigma = 2) and HPF images (Sigma = 5)



Hybrid Image and Resized Hybrid images


Essential
EVERYDAY
Shredded
Wheat
55g
7g

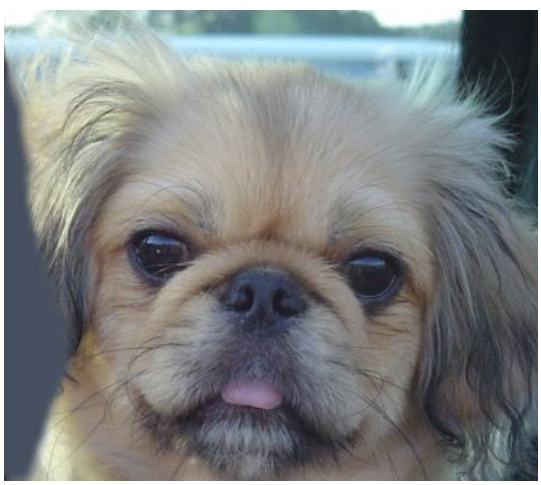

Essential
EVERYDAY
Shredded
Wheat
55g
7g


Shredded
Wheat

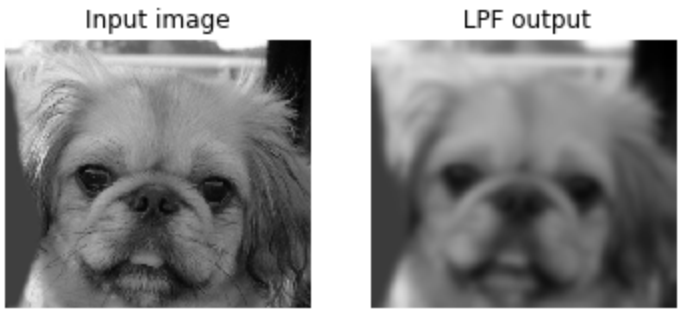
Comments on results : The 4th image on set of resized images looks like bag of cereal. The original sized image looks like cereal box.

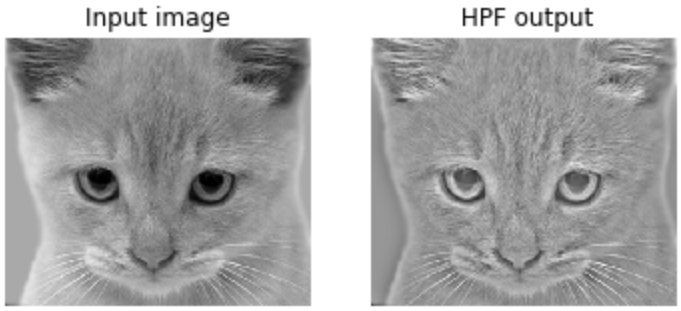
Example2

Dog (for LPF) and Cat (for HPF):




LPF (sigma = 5) and HPF images (sigma = 11)





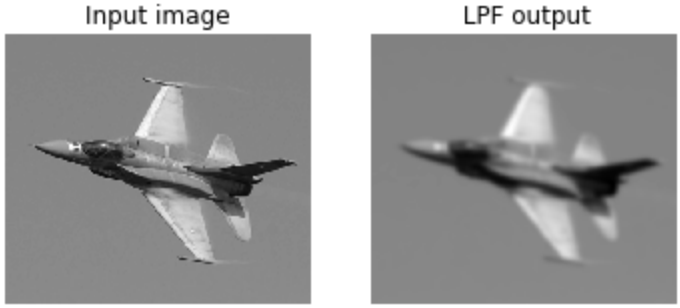
Hybrid image Resized hybrid images

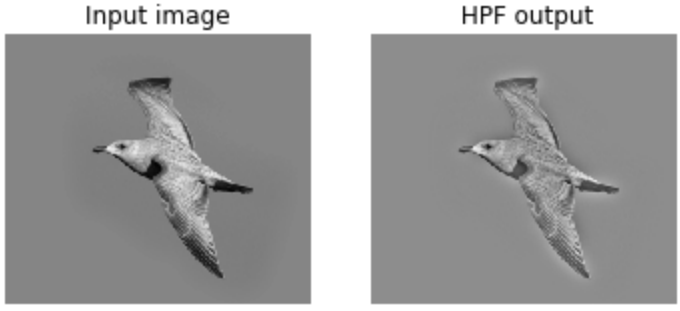
Comments on results : The hybrid image appears like cat in its original dimension and like a dog in the resized dimension. The 5th image on the resized images looks like a dog to a certain degree. The dog image used for LPF is not completely untextured (esp the hair), so the quality of output hybrid image at the lower resolution still has traces of cat. Moreover, in this example, high value of sigma was used for high pass filter. So the appearance of dog happens at a higher viewing distance, resembling the 5th resized image.

Example 3:

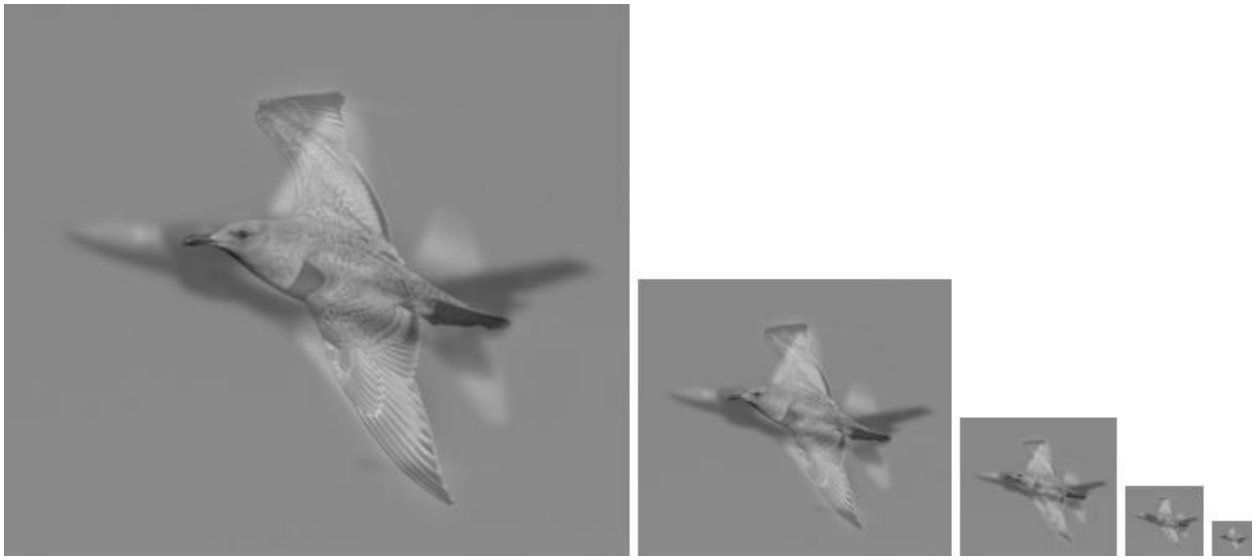
Aeroplane (for LPF) and Bird (for HPF)

LPF (sigma = 3) and HPF images (sigma = 9)





Hybrid image and resized hybrids images



Comments on results : The hybrid image appears like a bird in its original size and like a plane in resized image (esp. the 4th image on resized images). Even after alignment, the length of bird and plane are different.So the original hybrid image has a shadow of plane in the background.

In general, for hybrid image to be of high quality, both the images need to be aligned in shape and size.

References for Part1:

The additional images were obtained from https://www.cc.gatech.edu/~hays/compvision/proj1/. The input images were aligned and cropped to a respectable degree, required for hybrid generation.

Part 2:

Implementation details :

General details : A 3D scale_space array was used as recommended in MP2 spec-sheet to capture the scale space results.

Method 1: Upsize filter function : The recommended library function ndimage.filters.gaussian_laplace was used. The response function was normalized w.r.t sigma and the eventual response was squared.

Method 2: Downsample image : The filter size was kept constant and image was scaled by a factor K for obtaining the scale-space. Since the filter size is constant, it was not necessary to normalize the filter response w.r.t sigma.

3D Non-Maximum Suppression (NMS): This was achieved in 2 stages as recommended. A maximum filter from ndimage with a parameterizable kernel size was used on each level in the scale space. This step replaced every pixel intensity to the maximum value in the kernel window provided. The result from 2D was subject to 3D operation. A broadcast operator np.maximum was used to find the maximum of pixel values among levels (k-1, k, k+1). This operator was applied once per level and it does all pixels in a parallel fashion, speeding things up tremendously. The output from 3D step is compared from the original scale_space to create a boolean image mask - True if the current pixel value is maximum and False if it is not, representing 3D NMS. The boolean image mask is used to retrieve numerical values of scale space after 3D non-maximum suppression, using element-wise multiplication.

Obtaining circles : After 3D non-maximum suppression, a threshold is applied on the values and pixel locations satisfying the threshold are obtained. These pixel locations are centers of circles. The sigma is each level is multiplied by square root of 2 to obtain the radius of circle. The given plot function was used to plot the circles.

Parameters in code and how they were chosen :

- **Number of levels in scale-space**: For all the test cases, 15 was used.
- **Initial sigma** : The initial starting point of sigma is 2 for all images. This sigma is kept constant for method2.

- **Scaling factor**: The scaling factor range was from 1.16 to 1.28 for different images. A coarse tuning was performed in increments of 0.02 and once a decent value was obtained, minor fine tuning was performed to fix the scaling factor for each test image. Moreover, the circles obtained in resultant images (i.e if they encircle expected regions to a certain degree) serve as a visual clue to increase or decrease the scaling factor.
- **Kernel size for 2D NMS**: Sometimes a lot overlapping circles in adjacent pixels were obtained. The kernel size for 2D maximum filter helped in reducing this artifact. Depending on the image, the size was varied from 3 to 7.
- **Threshold**: Different thresholds were used for upsizing filter and downsizing image methods. The downsize image threshold was ~10x smaller than upsize filter threshold. The threshold was chosen based on the quantity of circles that could be displayed without too many overlaps. It was also ensured that the number of circles displayed from both the methods were almost same to help in comparison of execution times.

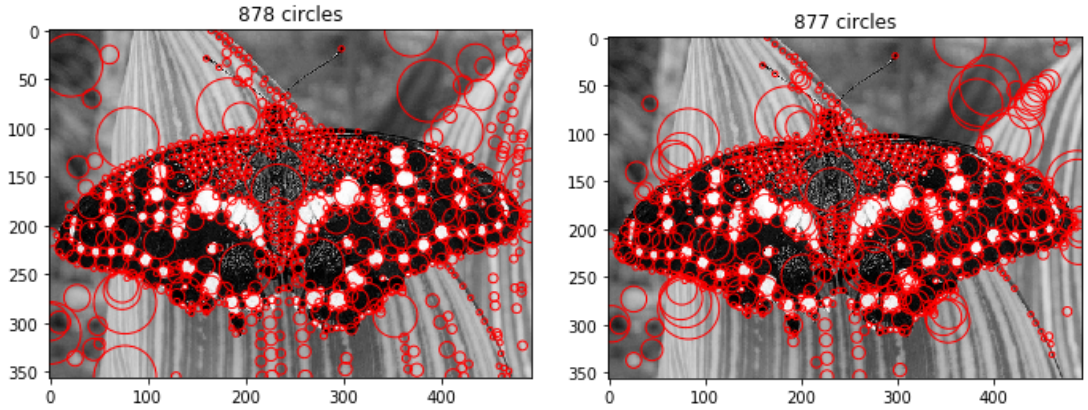
Note on execution time: The execution time was obtained by averaging 10 times.

<u>Butterfly</u>

Execution times in secs:
Non- Efficient method: 3.104
Efficient method : 2.145

Scaling factor: 1.189
2D NMS kernel size : 5
Upsize filter threshold: 0.013
Downsize image threshold : 0.001

Upsize filter on the left vs Downsize image on the right

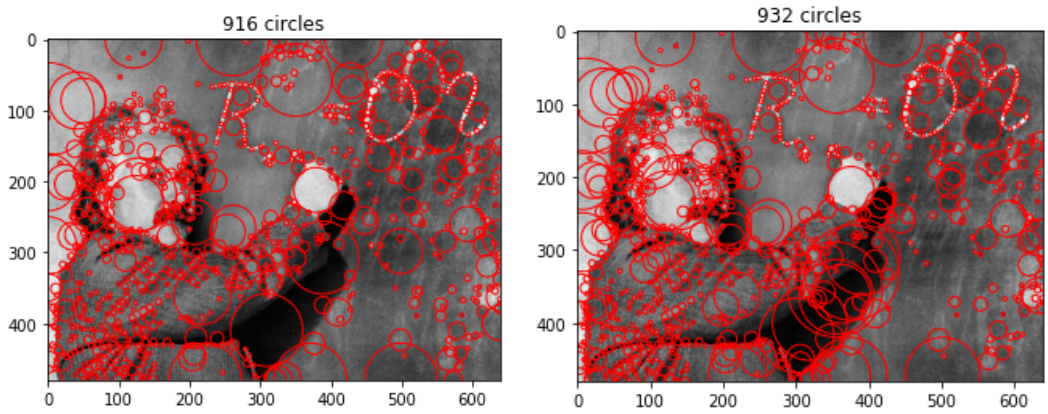


Einstein

Execution times in secs:
Non- Efficient method:5.864
Efficient method : 2.978

Scaling factor: 1.273
2D NMS kernel size : 7
Upsize filter threshold: 0.003
Downsize image threshold : 0.0002

Upsize filter on the left vs Downsize image on the right

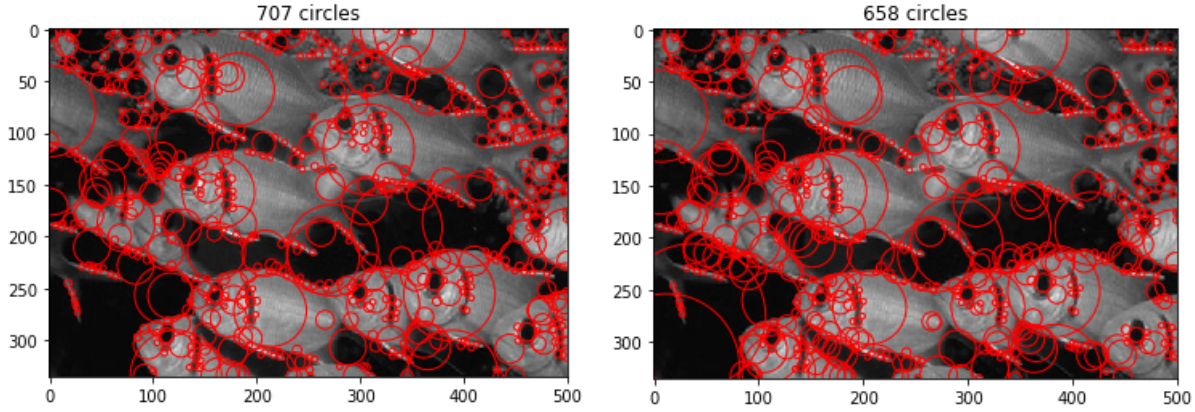


Fishes

Execution times in secs:
Non- Efficient method: 2.971
Efficient method: 1.790
Number of levels: 13

Scaling factor: 1.257
2D NMS kernel size : 5
Upsize filter threshold: 0.005
Downsize image threshold : 0.0004

Upsize filter on the left vs Downsize image on the right

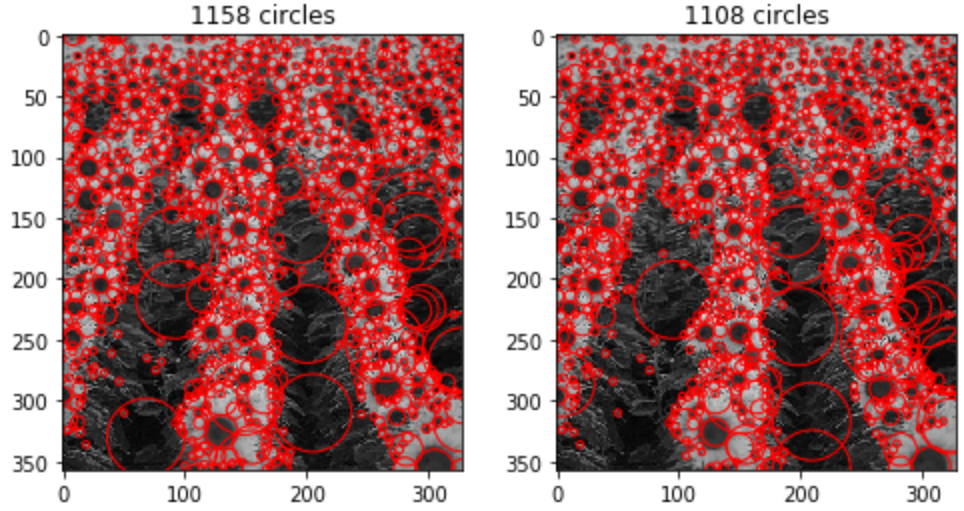


Sunflower:

Execution times in secs:
Non- Efficient method: 3.125
Efficient method: 2.6

Scaling factor: 1.189
2D NMS kernel size : 5
Upsize filter threshold: 0.01
Downsize image threshold : 0.007

Upsize filter on the left vs Downsize image on the right



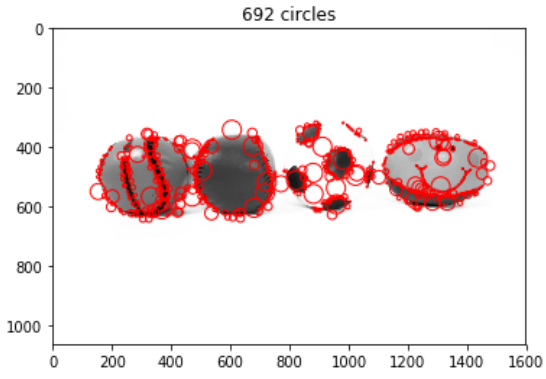
Small Ball:

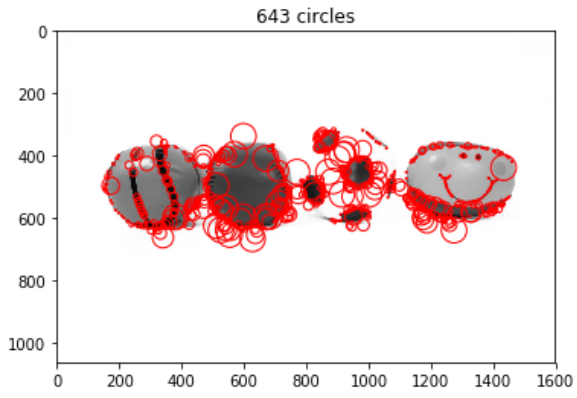
Original Image

Execution times in secs:
Non- Efficient method : 12.515
Efficient method : 7.641

Scaling factor: 1.21
2D NMS kernel size : 7
Upsize filter threshold: 0.005
Downsize image threshold : 0.0009

Upsize filter on the left vs Downsize image on the right





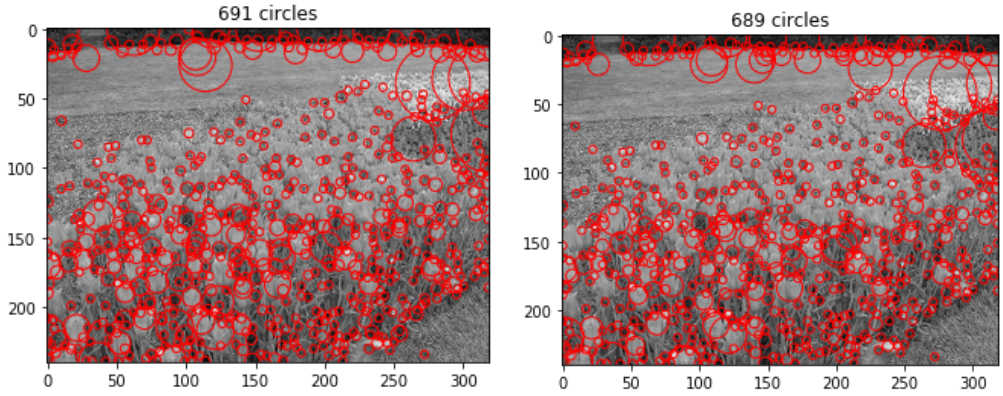
Tulips:

Original Image

Execution times in secs:
Non- Efficient method:1.42
Efficient method: 1.34

Scaling factor: 1.189
2D NMS kernel size : 5
Upsize filter threshold: 0.005
Downsize image threshold : 0.0003

Upsize filter on the left vs Downsize image on the right



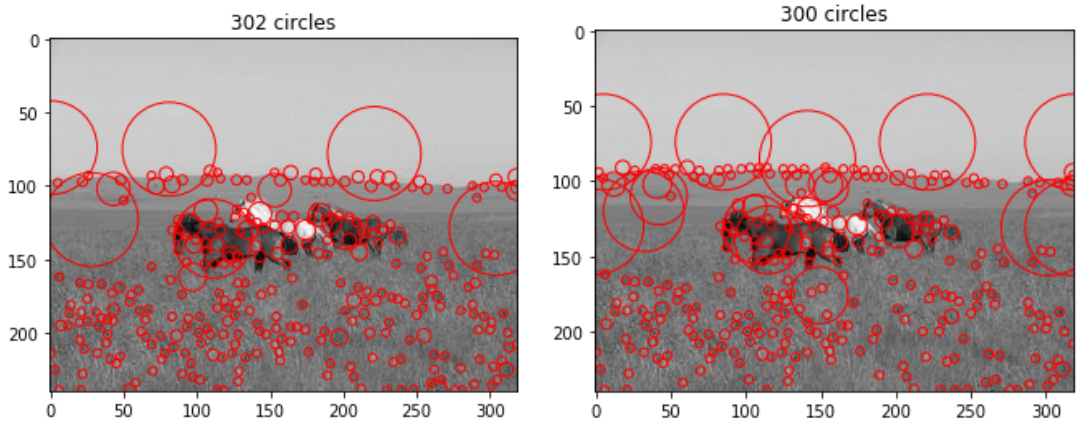
Horses:

Original Image

Execution times in secs:
Non- Efficient method: 0.9768
Efficient method: 0.7717

Scaling factor: 1.189
2D NMS kernel size : 7
Upsize filter threshold: 0.001
Downsize image threshold : 0.00007

Upsize filter on the left vs Downsize image on the right



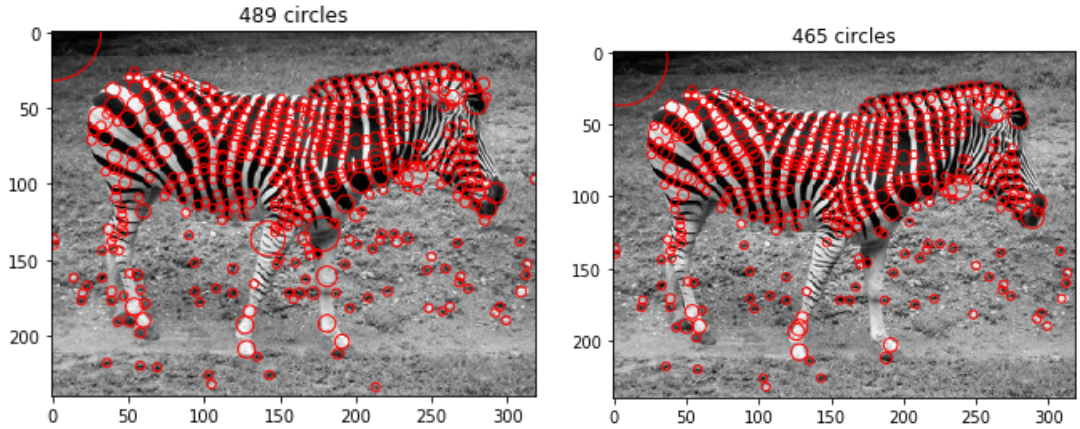
Zebra

Original image

Execution times in secs:
Non- Efficient method: 1.068
Efficient method: 1.018

Scaling factor: 1.189
2D NMS kernel size : 5
Upsize filter threshold: 0.02
Downsize image threshold : 0.0015

Upsize filter on the left vs Downsize image on the right



General comment on all results: There are spurious circles on edges as expected. A filter using Harris edge detector would resolve the is ues.

References:
Last 4 test images are obtained from
https://www.freeimages.com/photo/ball-1418260
https://www.freeimages.com/search/tulip-fa
https://www.freeimages.com/search/zebrarm
https://www.freeimages.com/search/horse