

Assignment 4

Image Filtering and Denoising

Any Filter: **anyf**

The algorithm used by me: Fast Non-Local Means Denoising (Coloured) [[Link to Paper](#)]. I used the library function from OpenCV directly. In short, the algorithm can be described as: "We take a pixel, take a small window around it, search for similar windows in the image, average all the windows and replace the pixel with the result we got. This method is Non-Local Means Denoising."




The algorithm performs extremely well as is clear from the output images. The noise in the sky is completely removed and the blurring is minimal as well.

```
[cv2.fastNlMeansDenoisingColored(image, destination, p1, p2, p3, p4)]
```

The optimal values of parameters:

Parameter Variable	Parameter Name	Optimal Value	Remark
p1	Filter Strength	10	Decides Filter Strength
p2	Filter Strength	10	Same as p1, but for coloured image
p3	templateWindowSize	7	Size of the patch [p3 needs to increase with p4, should be odd]
p4	searchWindowSize	15	Research window size [increase to improve noise removal capacity, should be odd]

Output Images:

Input: rome.jpg	Output: anyf_outputs/rome.jpg
	
Input: iitk.jpg	Output: anyf_outputs/iitk.jpg
	

Bilateral Filter: **mybf**

For improving the speed of “mybf_original”, I have utilised the concept of stride wherein, instead of visiting all pixels one by one, we do the following:

1. Visit the pixel
2. Get the value of that pixel as done in mybf_original
3. Replace a window of size (s,s) around the pixel [(i-s/2, j-s/2, k) to (i+s/2, j+s/2, k) : (i, j, k) are the pixel coordinates] with the new value
4. Skip next “s” pixels, goto 1 if anymore pixels are left

Note: This method reduces the time taken by a factor of s^2 but this has a side-effect of reducing the quality of the image. As this improvement reduces the quality of the image, it is not applied automatically and an additional parameter called “stride” needs to be supplied if a speedup is desired.

There is one more improvement that can be done. This is very similar to what we have done in rtbf. We can assume that there are 256 pixel intensity buckets and do rtbf on it.

Please Note: *The functions my BF and rt BF functions do not change the color space of the input images, if the input image is in BGR (as given in the example), the output image will also be in BGR.*

Analysis: table

Table of Parameters

Type	Number of Intensity Bins
Good Accuracy	32
Good Speed	4
Best Parameter	8

rome.jpg

Algorithm	Time (in sec.)	PSNR (dB)	SSIM
anybf	7.932	32.856	0.941
mybf (original)	233.345	N/A	
mybf (speed)	60.696	32.396	0.758
rtbf (para-1: good accuracy)	27.439	38.477	0.985
rtbf (para-2: good speed)	1.981	32.314	0.958
rtbf (para-3: best para)	3.525	38.671	0.983
rtbf (best para) (JPEG 99)	3.808	38.608	0.982
rtbf (best para) (JPEG 95)	3.638	38.333	0.982
rtbf (best para) (JPEG 90)	3.925	37.524	0.979

iitk.jpg

Algorithm	Time (in sec.)	PSNR (dB)	SSIM
anybf	13.162	35.036	0.828
mybf (original)	1594.501	N/A	
mybf (speed)	448.014	44.059	0.978
rtbf (para-1: good accuracy)	155.154	50.119	0.992
rtbf (para-2: good speed)	12.123	43.418	0.985
rtbf (para-3: best para)	22.883	48.783	0.991
rtbf (best para) (JPEG 99)	23.135	42.092	0.973
rtbf (best para) (JPEG 95)	23.067	42.079	0.973
rtbf (best para) (JPEG 90)	22.964	41.896	0.971

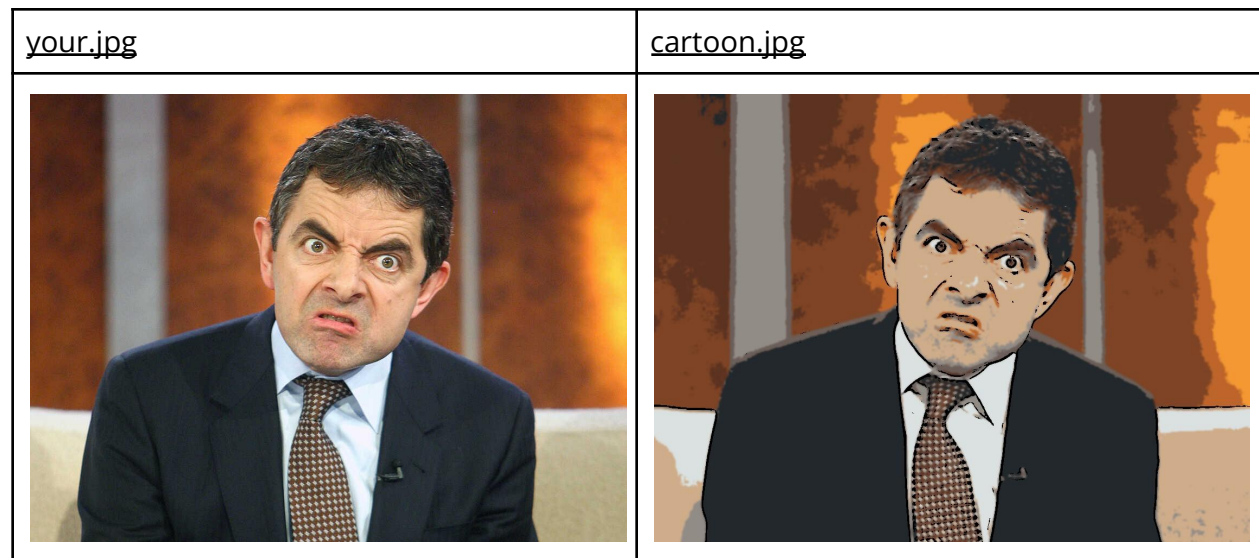
Application: **cartoonizer**

Algorithm:

1. Load the image (your.jpg)
2. Use an edge detector to get a mask that can be used to enhance edges
3. Make N rt_BF passes on the image and store it in img_rt
4. Enhance img_rt using mask (Bitwise_and)
5. Save the image (cartoon.jpg)

Parameters:



1. Kernel_size = 51
2. Sigma_s = 256
3. Sigma_r = 0.025
4. Number of Color Bins = 15
5. Number of rt_BF iterations (N) = 4



Segmentation: **seg**

Algorithm:

1. Filter the image using the filter from “anyf”.
2. Convert the filtered image to HSV color space.
3. Filter out the “green” areas in the image.
4. Save the generated mask as “fordrone.jpg”

Original Image	Filtered Image
	
Mask [<i>fordrone.jpg</i>]	Image after applying the mask
