CS543 Assignment 2

Your Name: Karan Pandya
Your NetId: karandp2

Part 1 Fourier-based Alignment:

You will provide the following for each of the six low-resolution and three high-resolution images:

- Final aligned output image.
- Displacements for color channels.
- Inverse Fourier transform output visualization for ***both*** channel alignments ***without*** preprocessing.
- Inverse Fourier transform output visualization for ***both*** channel alignments ***with*** any sharpening or filter-based preprocessing you applied to color channels.

You will provide the following as further discussion overall:

- Discussion of any preprocessing you used on the color channels to improve alignment and how it changed the outputs
- Measurement of Fourier-based alignment runtime for high-resolution images (you can use the python time module again). How does the runtime of the Fourier-based alignment compare to the basic and multiscale alignment you used in Assignment 1?

A: Channel Offsets

Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel. Provide offsets in the **original image coordinates** (after the image has been divided into three equal parts corresponding to each channel) and be sure to account for any cropping or resizing you performed.

Low-resolution images (using channel <C1> as base channel):

Image	<C2> (h,w) offset	<C3> (h,w) offset

00125v.jpg	[4,2]	[8,1]
00149v.jpg	[3,2]	[7,2]
00153v.jpg	[6,3]	[12,5]
00351v.jpg	[3,1]	[11,1]
00398v.jpg	[4,3]	[9,4]
01112v.jpg	[-1,0]	[3,1]

High-resolution images (using channel <C1> as base channel):

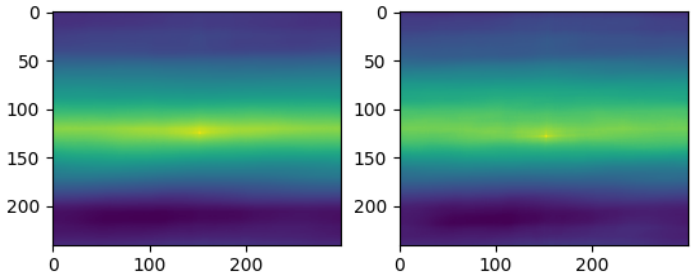
Image	<C2> (h,w) offset	<C3> (h,w) offset
01047u.tif	[23,19]	[69,33]
01657u.tif	[49,9]	[110,12]
01861a.tif	[70,39]	[146,63]

B: Output Visualizations

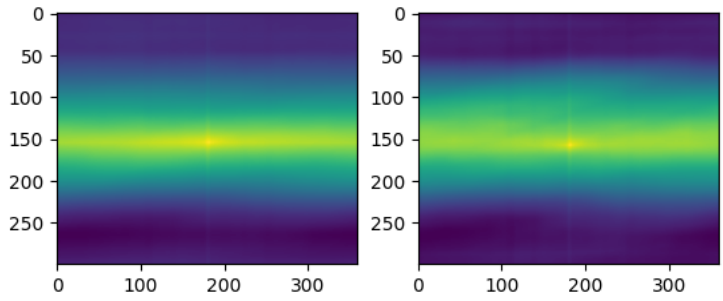
For each image, insert 5 outputs total (aligned image + 4 inverse Fourier transform visualizations) as described above. When you insert these outputs be sure to clearly label the inverse Fourier transform visualizations (e.g. “G to B alignment without preprocessing”).

00125v.jpg

G to B and G to R alignment with preprocessing

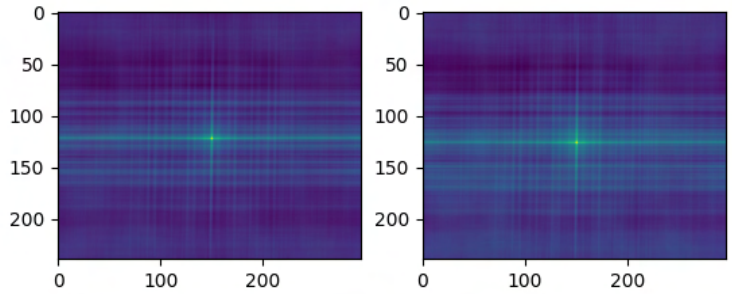


G to B and G to R alignment without preprocessing

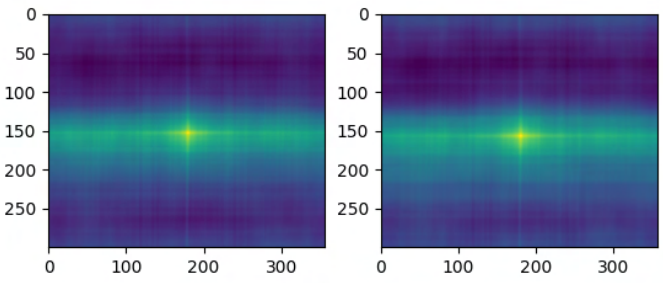


00149v.jpg

G to B and G to R alignment with preprocessing

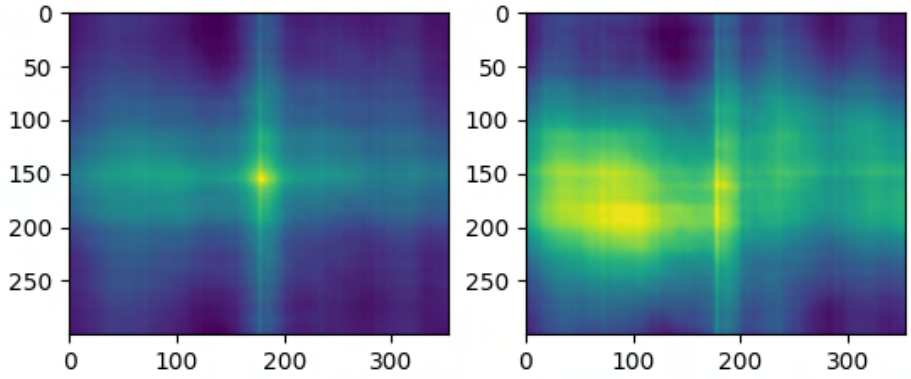


G to B and G to R alignment without preprocessing

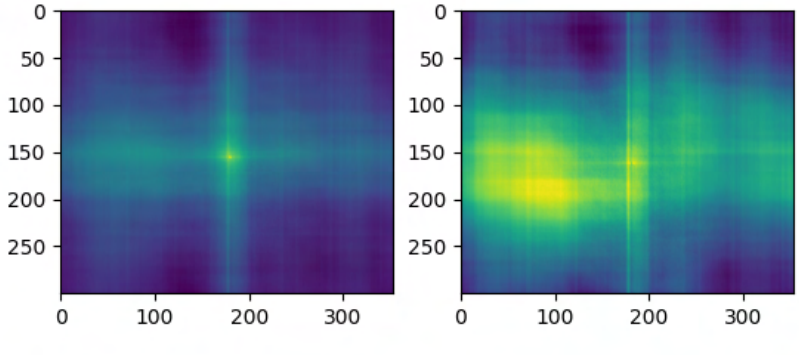


00153v.jpg

G to B and G to R alignment with preprocessing

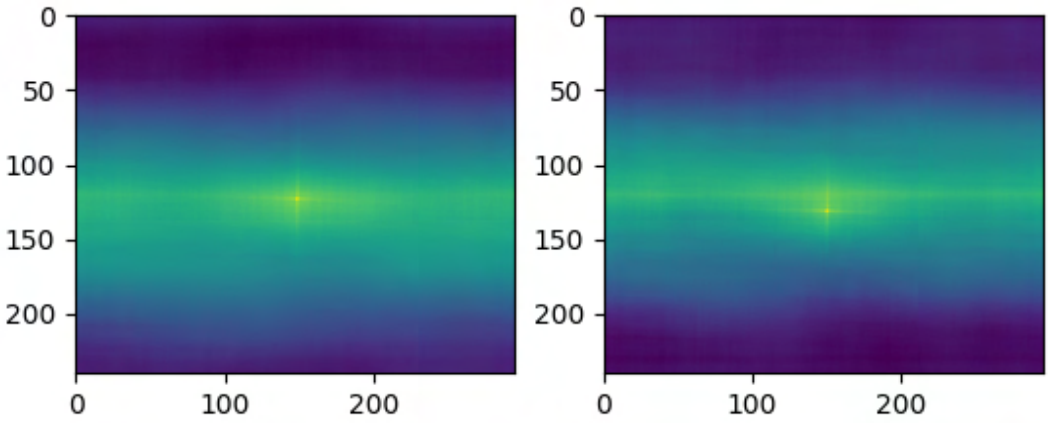


G to B and G to R alignment without preprocessing

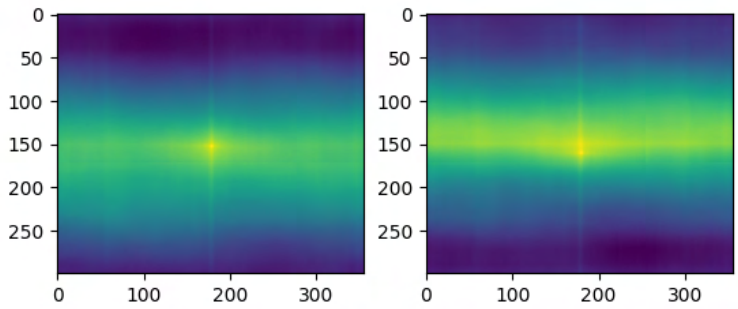


00351v.jpg

G to B and G to R alignment with preprocessing

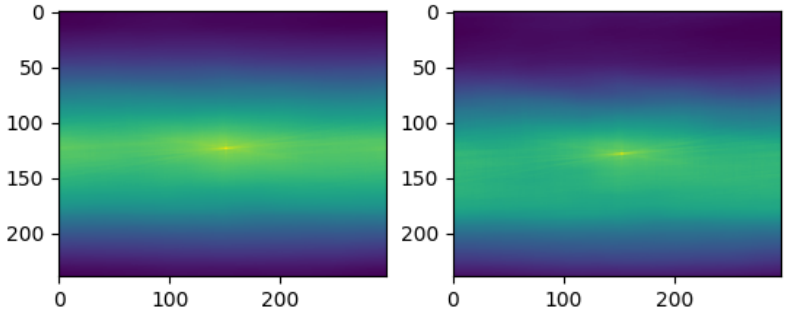


G to B and G to R alignment without preprocessing

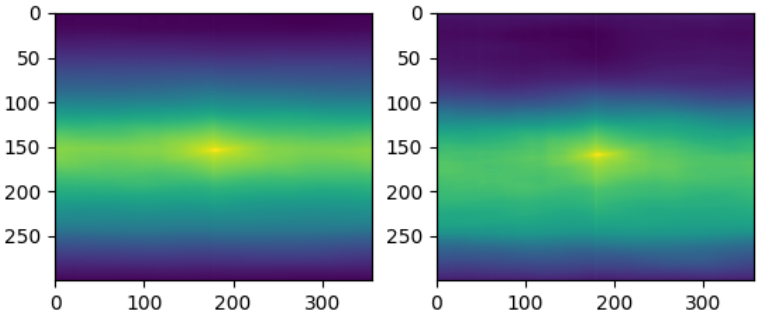


00398v.jpg

G to B and G to R alignment with preprocessing

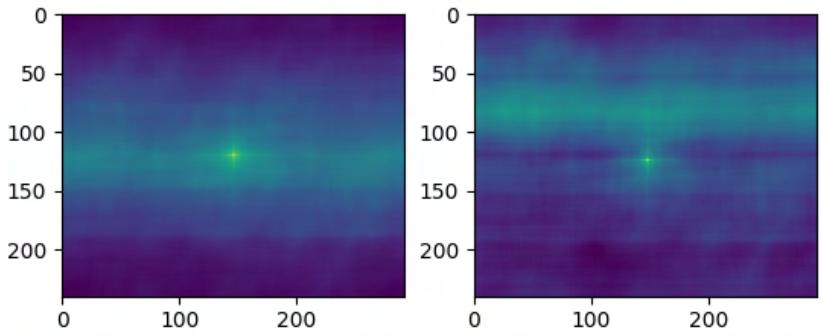


G to B and G to R alignment without preprocessing

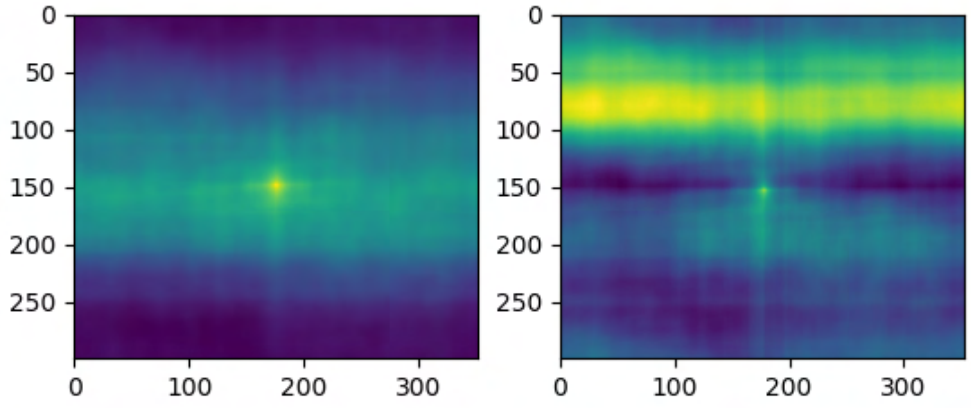


01112v.jpg

G to B and G to R alignment with preprocessing

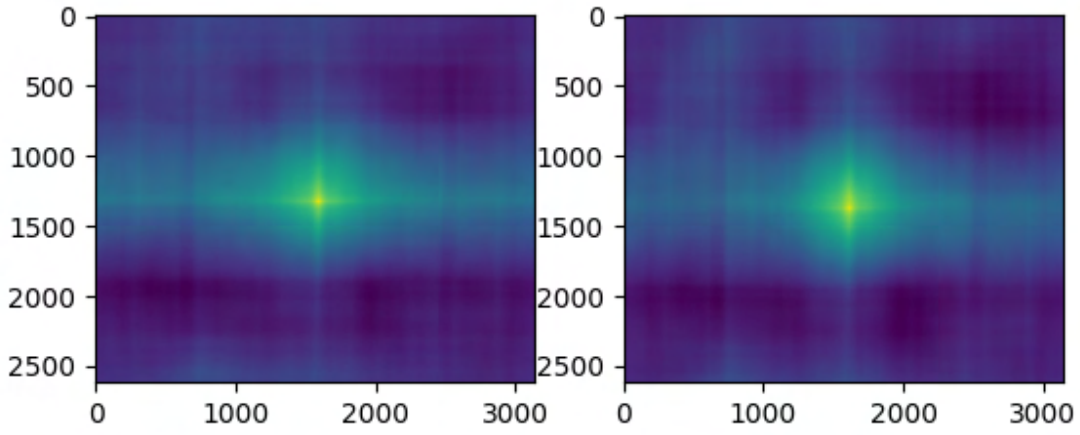


G to B and G to R alignment without preprocessing

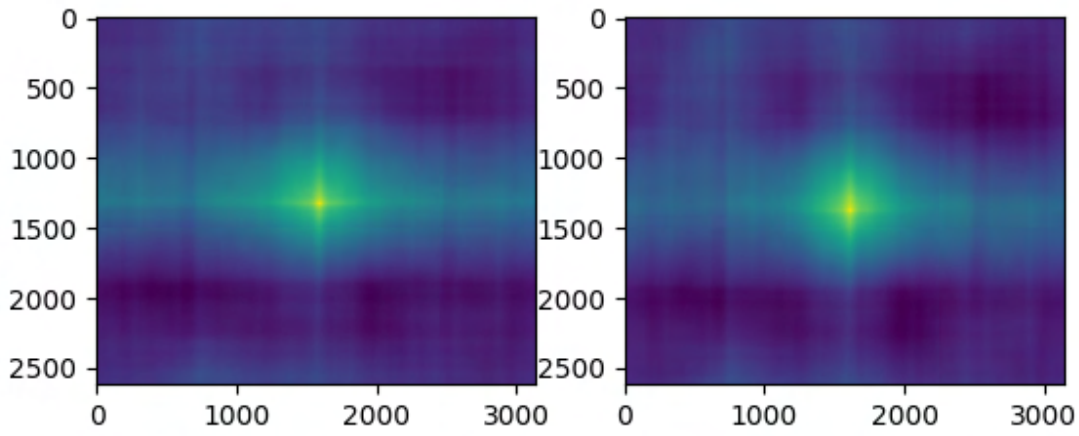


01047u.tif

G to B and G to R alignment with preprocessing



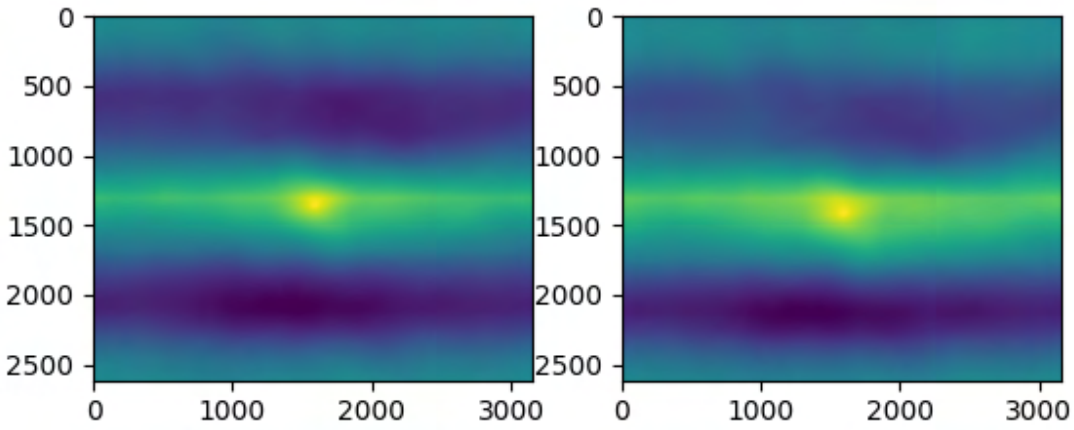
G to B and G to R alignment without preprocessing



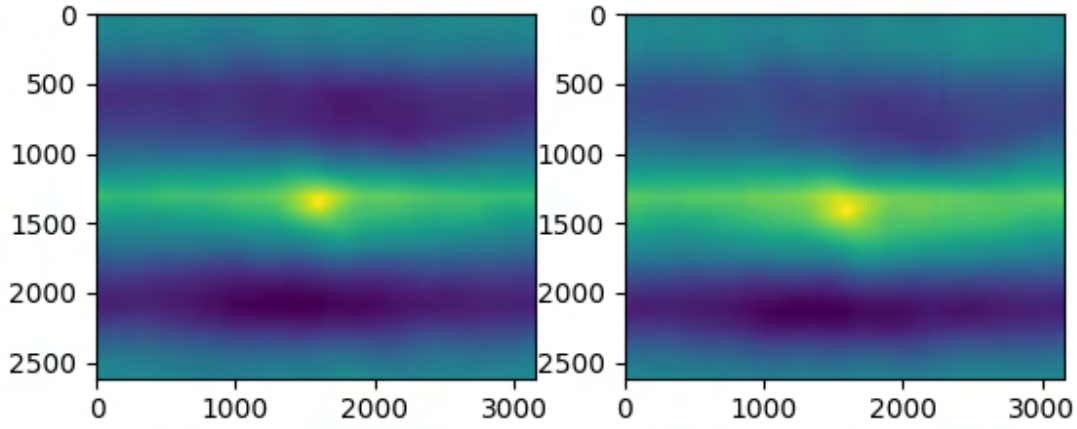
Total runtime : 3.56 seconds
Runtime in previous alignment : 45.96s

01657u.tif

G to B and G to R alignment with preprocessing



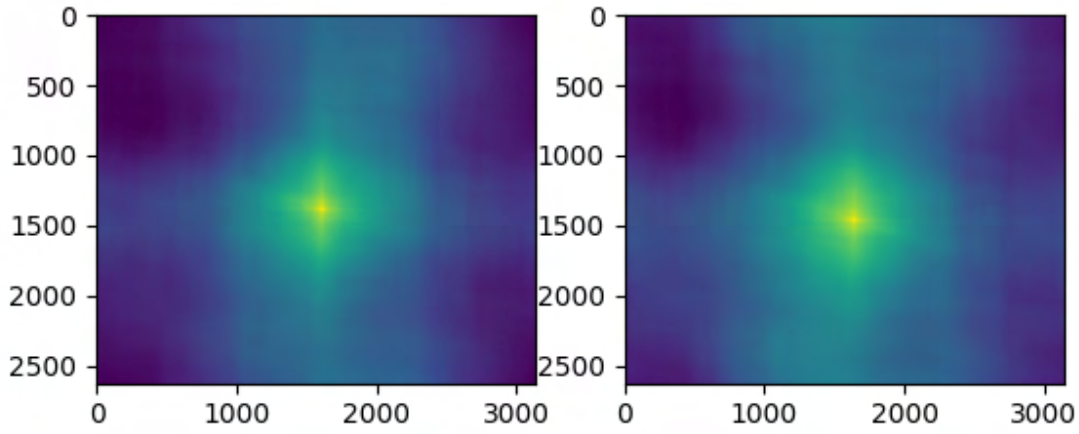
G to B and G to R alignment without preprocessing



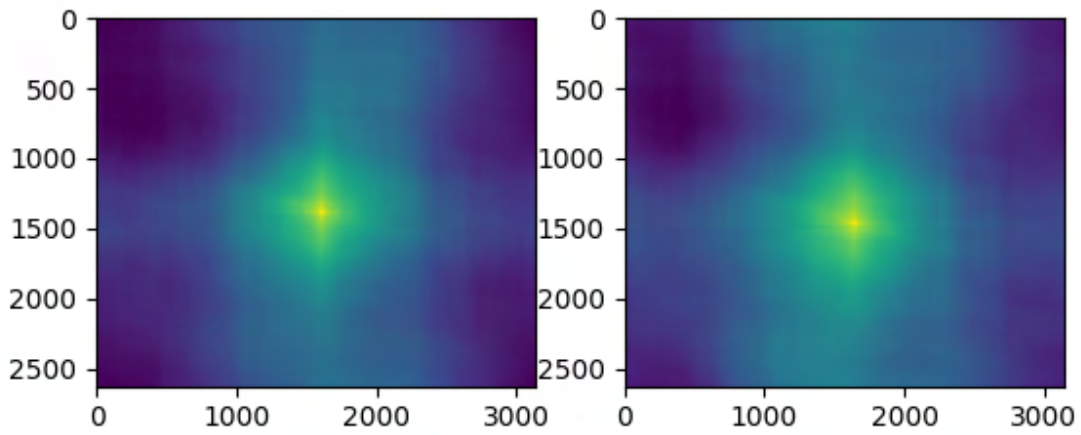
Total runtime : 3.43 seconds
Runtime in previous alignment : 43.49s

01861a.tif

G to B and G to R alignment with preprocessing



G to B and G to R alignment without preprocessing



Total runtime : 3.19 seconds
Runtime in previous alignment : 49.06s

C: Discussion and Runtime Comparison

Q1) Describe any preprocessing you used on the color channels to improve alignment and how it changed the outputs?
Answer:
I have used image sharpening to make the edges more visible which helps in getting a sharper point in fourier transform hence leading to better alignment.

Q2) Measure the Fourier-based alignment runtime for high-resolution images (you can use the python time module again). How does the runtime of the Fourier-based alignment compare to the basic and multiscale alignment you used in Assignment 1?

Answer:
Runtime in fourier based alignment was 10x faster than in pyramid scale alignment done in the previous assignment.

Part 2 Scale-Space Blob Detection:

You will provide the following for ***4 different examples*** (4 provided, 4 of your own):

- original image
- output of your circle detector on the image
- running time for the "efficient" implementation on this image
- running time for the "inefficient" implementation on this image

You will provide the following as further discussion overall:

- Explanation of any "interesting" implementation choices that you made.
- Discussion of optimal parameter values or ones you have tried

Example 1:

The original image

The base image

The image shifted about 20% to the left and cropped

The image shifted about 20% to the right and cropped

The image rotated by 90 degrees counterclockwise

The image rotated by 90 degrees clockwise

The image enlarged by a factor of 2 and center cropped

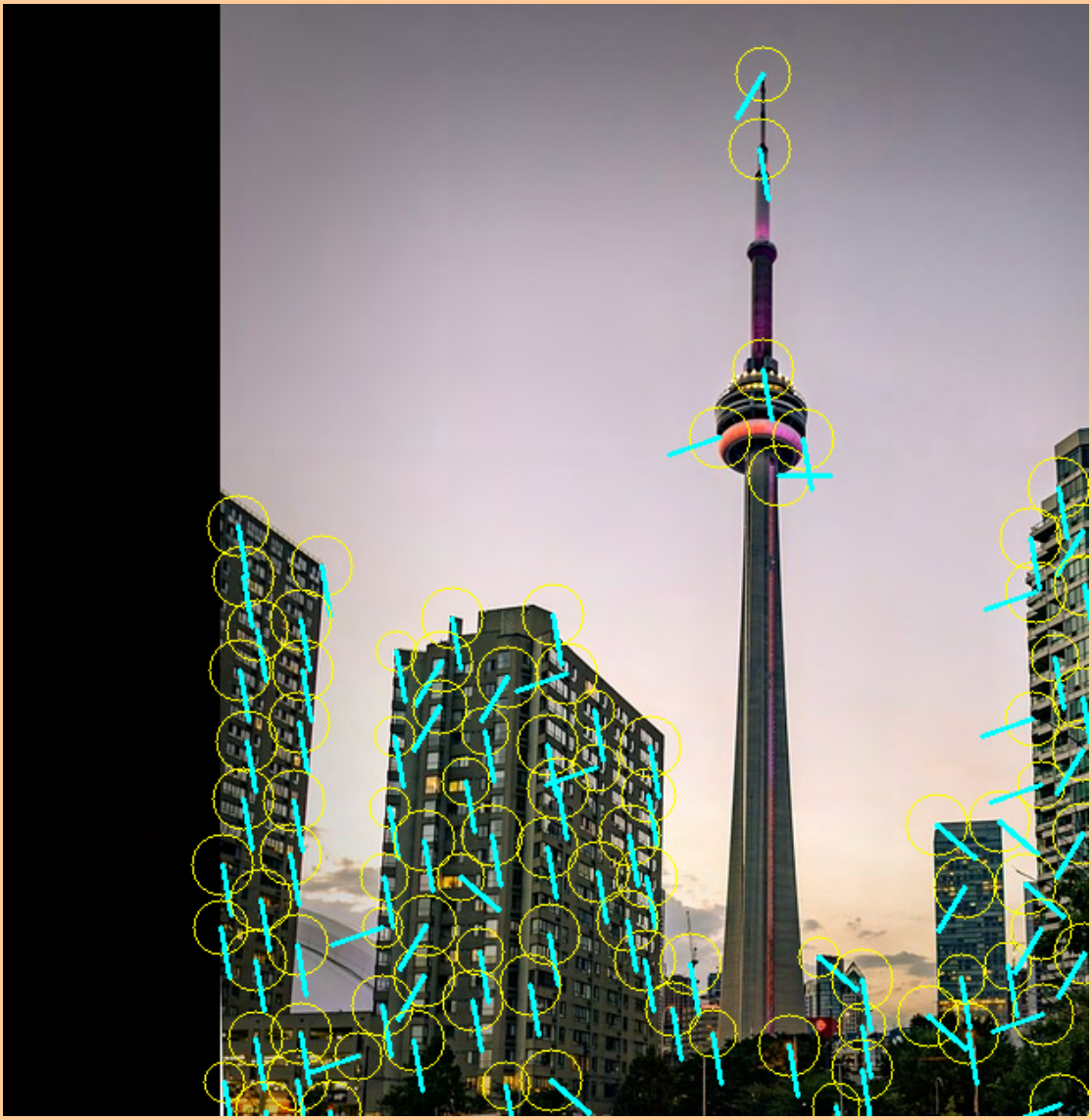
Example 2:

The original image

The base image

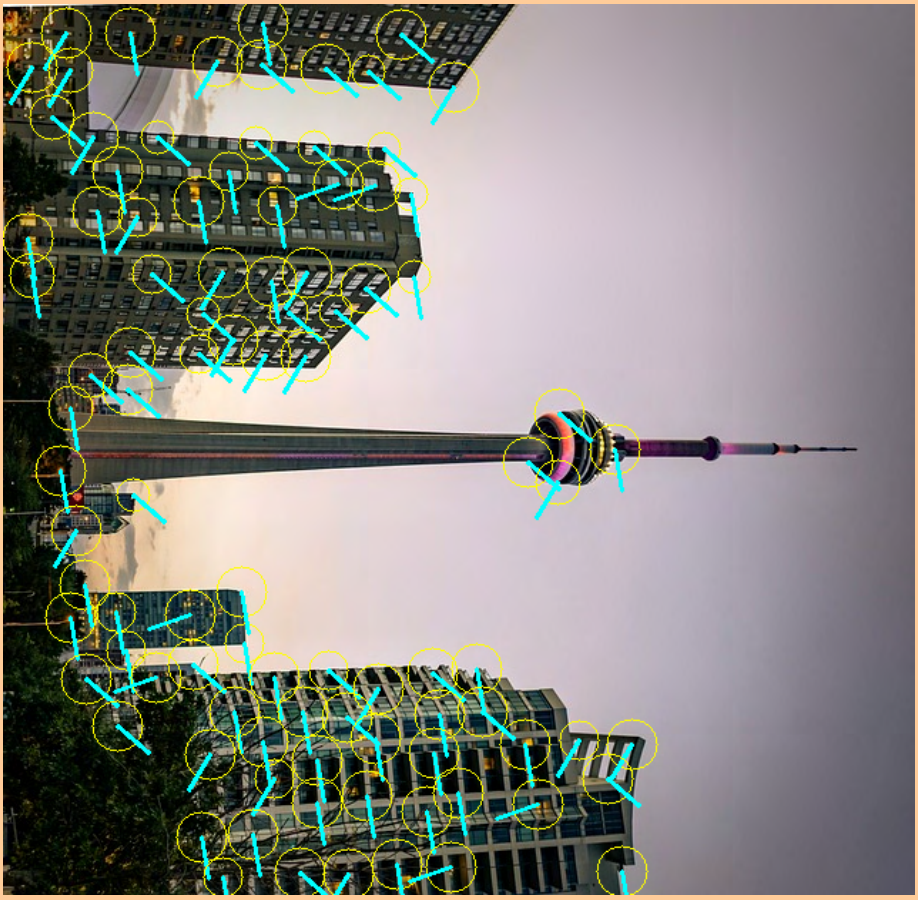
The image shifted about 20% to the left and cropped

The image shifted about 20% to the right and cropped



The image rotated by 90 degrees counterclockwise

The image rotated by 90 degrees clockwise



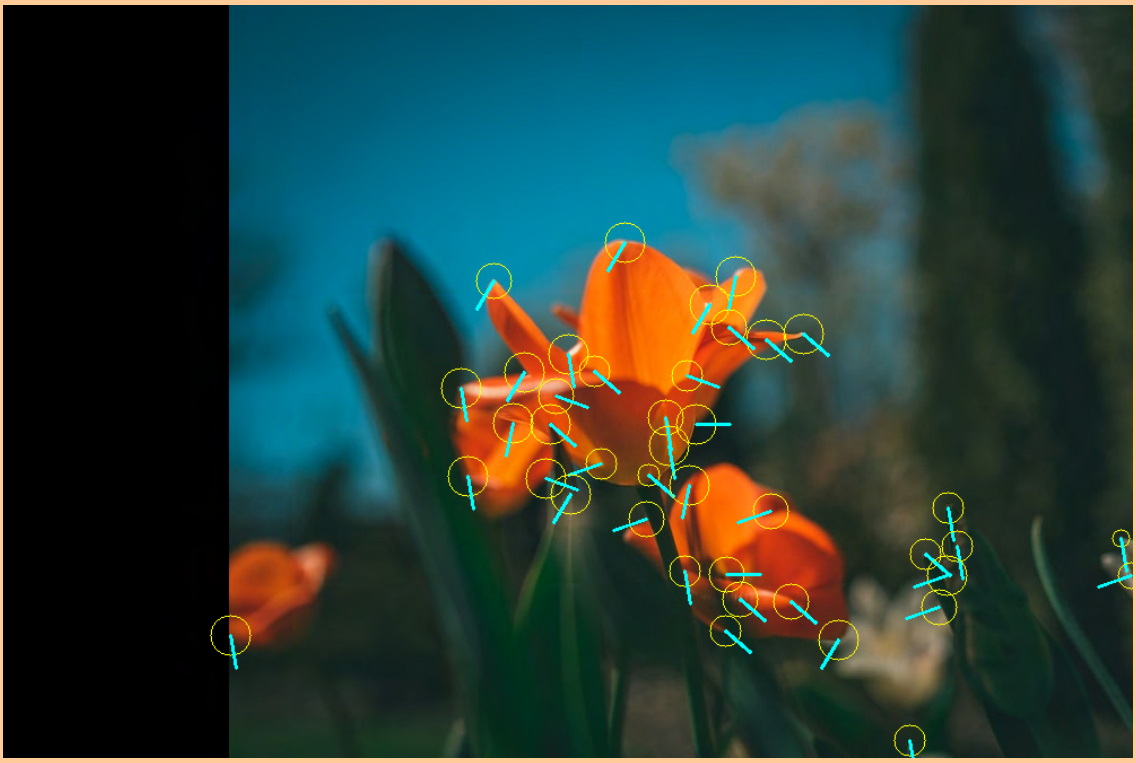
The image enlarged by a factor of 2 and center cropped

Example 3:

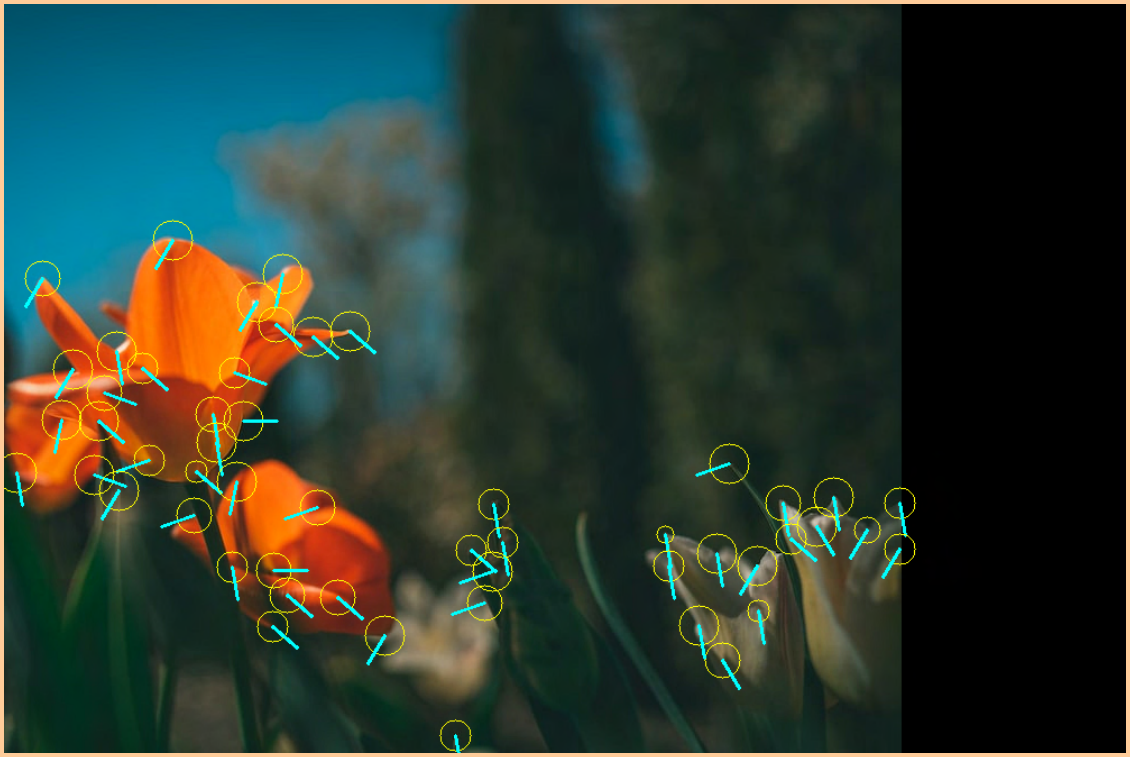
The original image

The base image

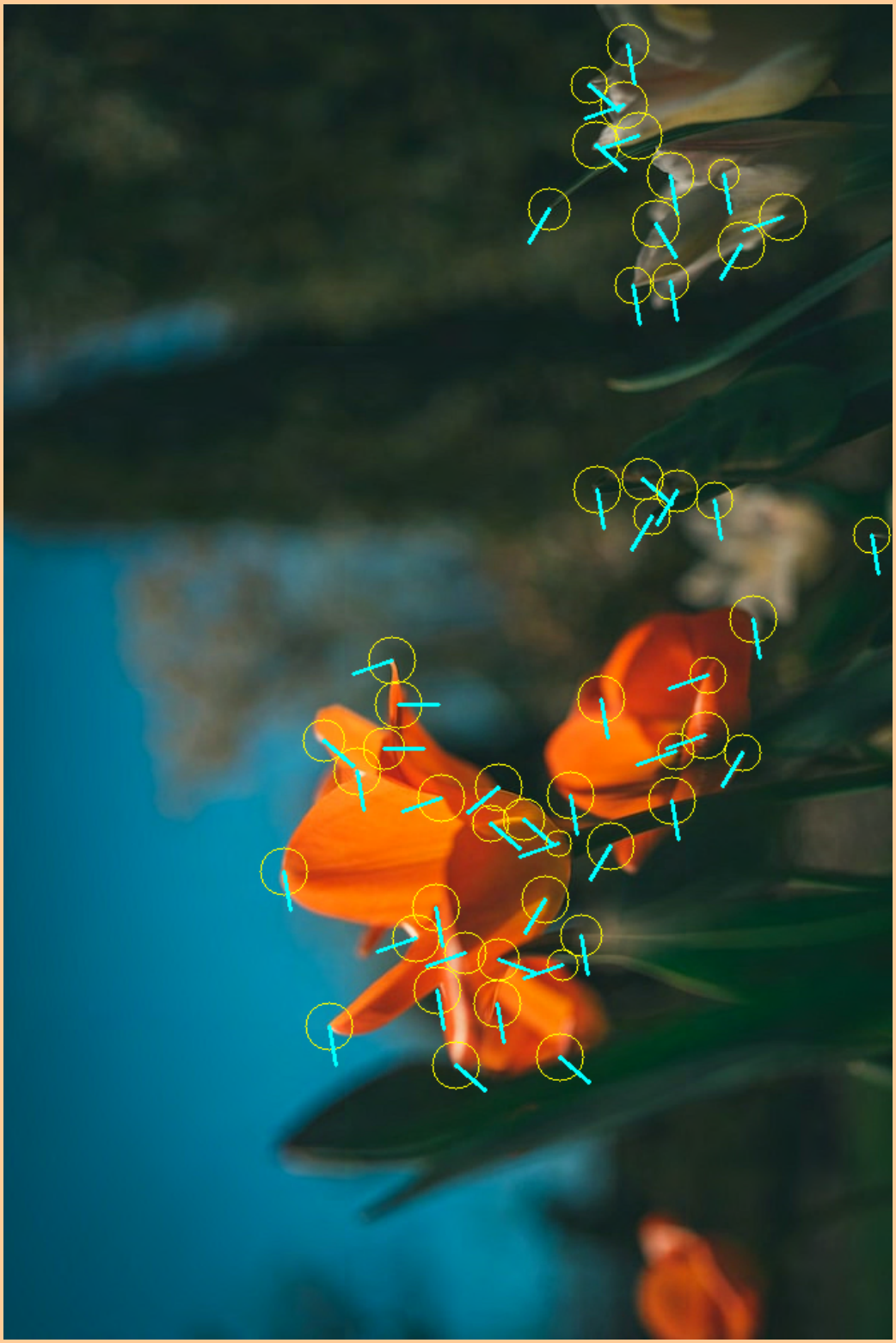
The image shifted about 20% to the left and cropped



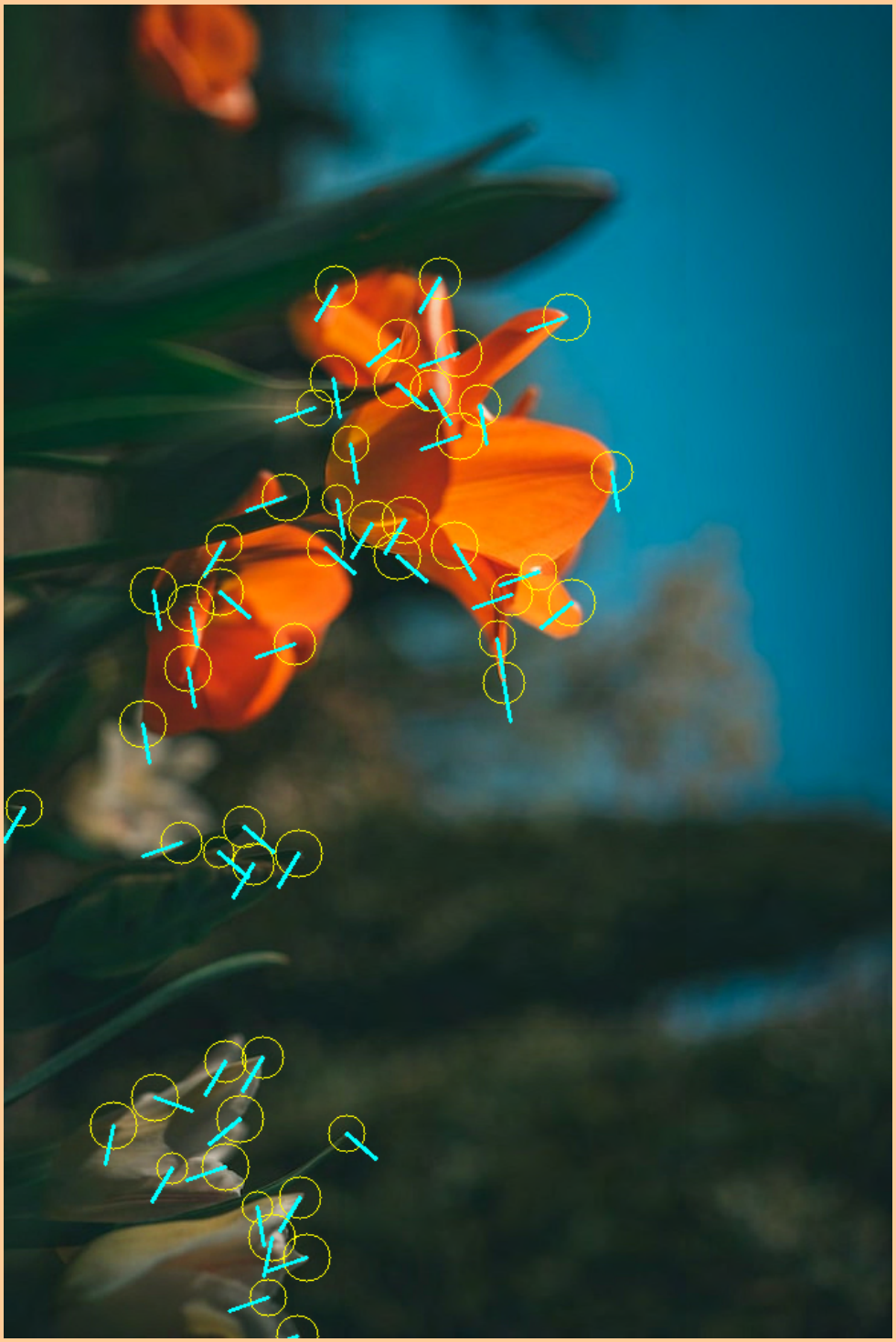
The image shifted about 20% to the right and cropped



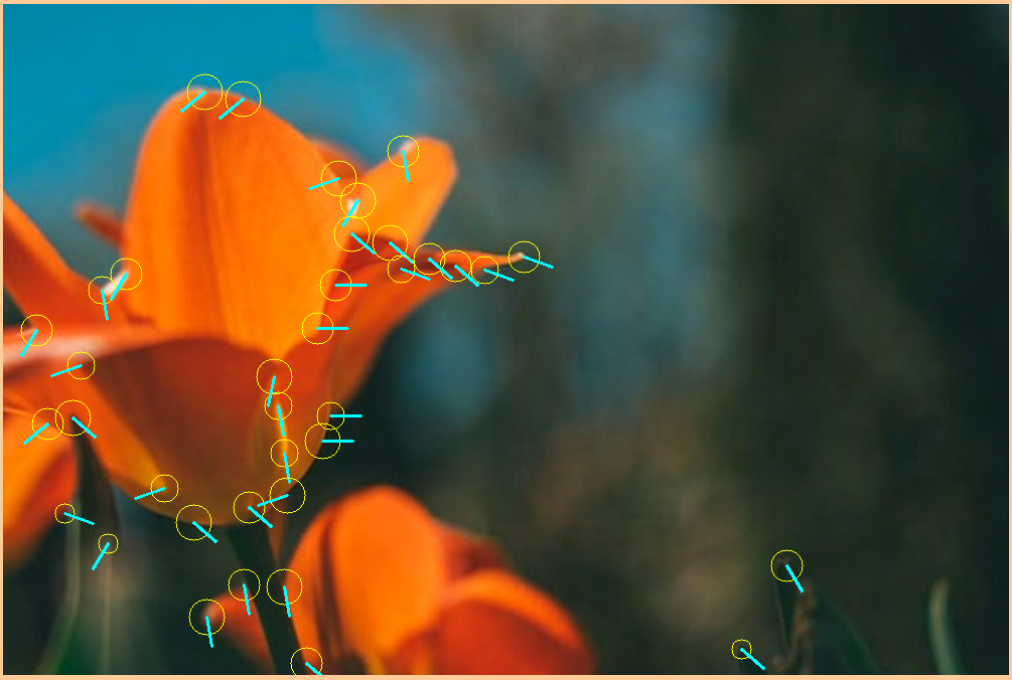
The image rotated by 90 degrees counterclockwise



The image rotated by 90 degrees clockwise



The image enlarged by a factor of 2 and center cropped

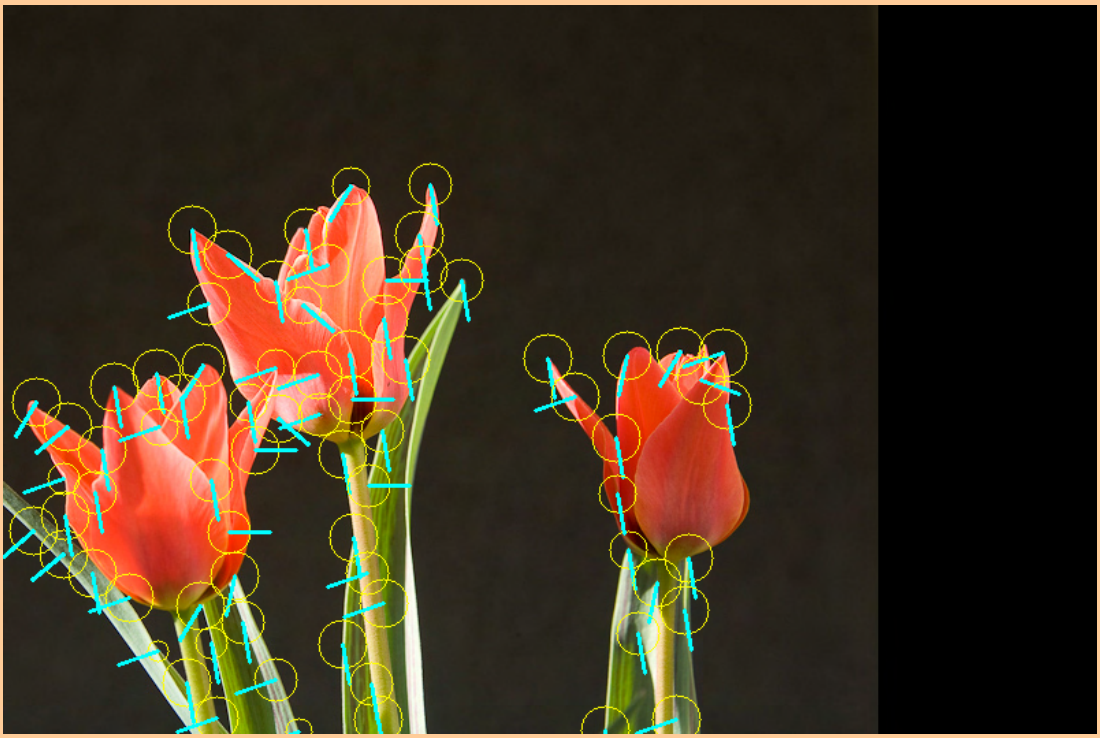


Example 4:

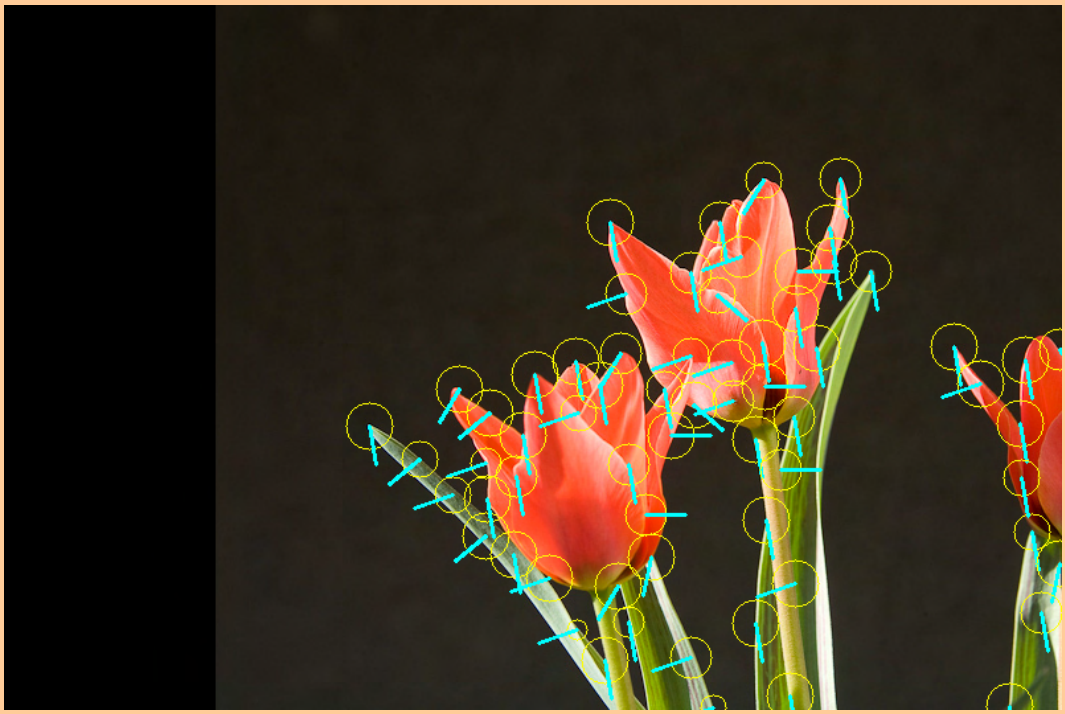
The original image

The base image

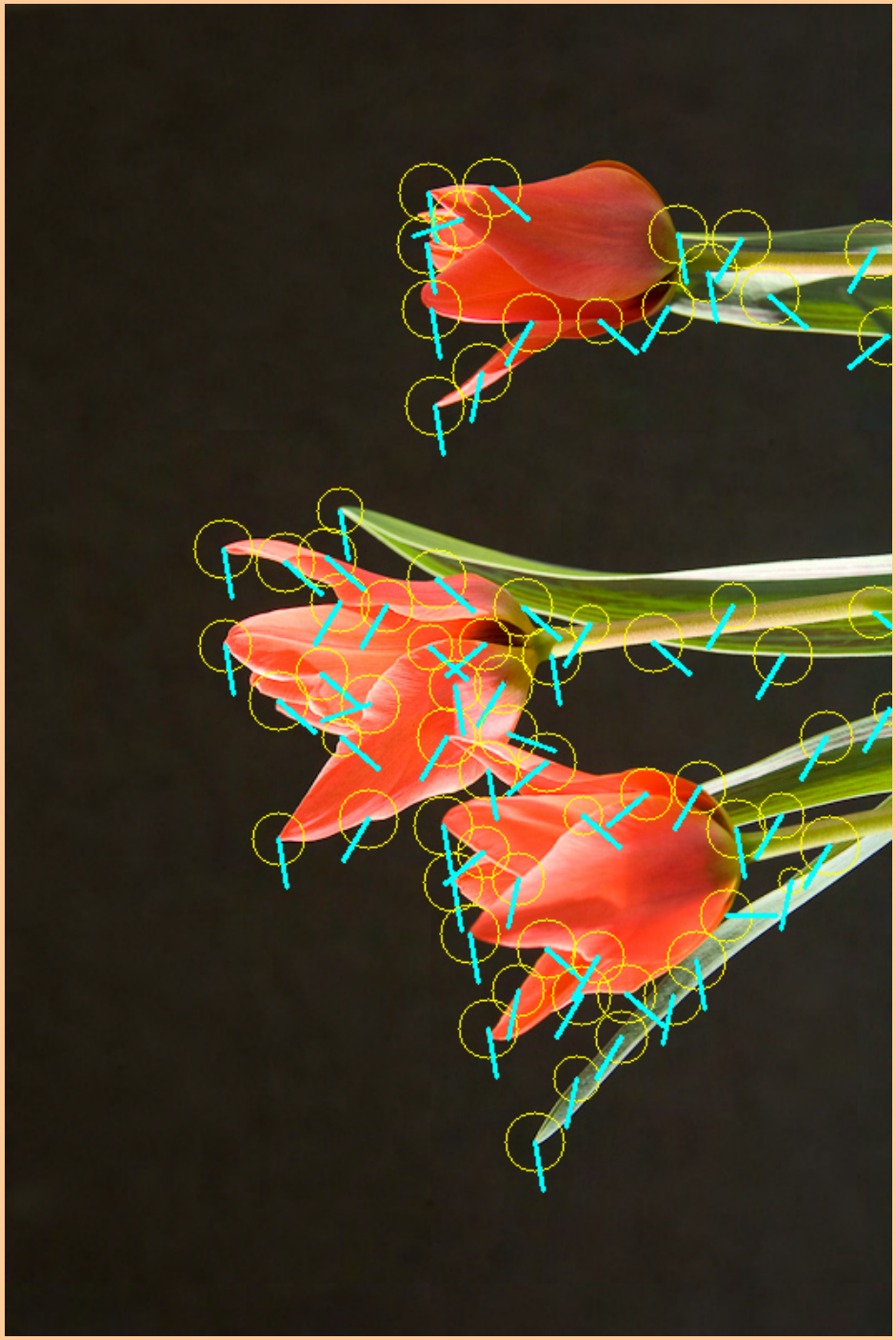
The image shifted about 20% to the left and cropped



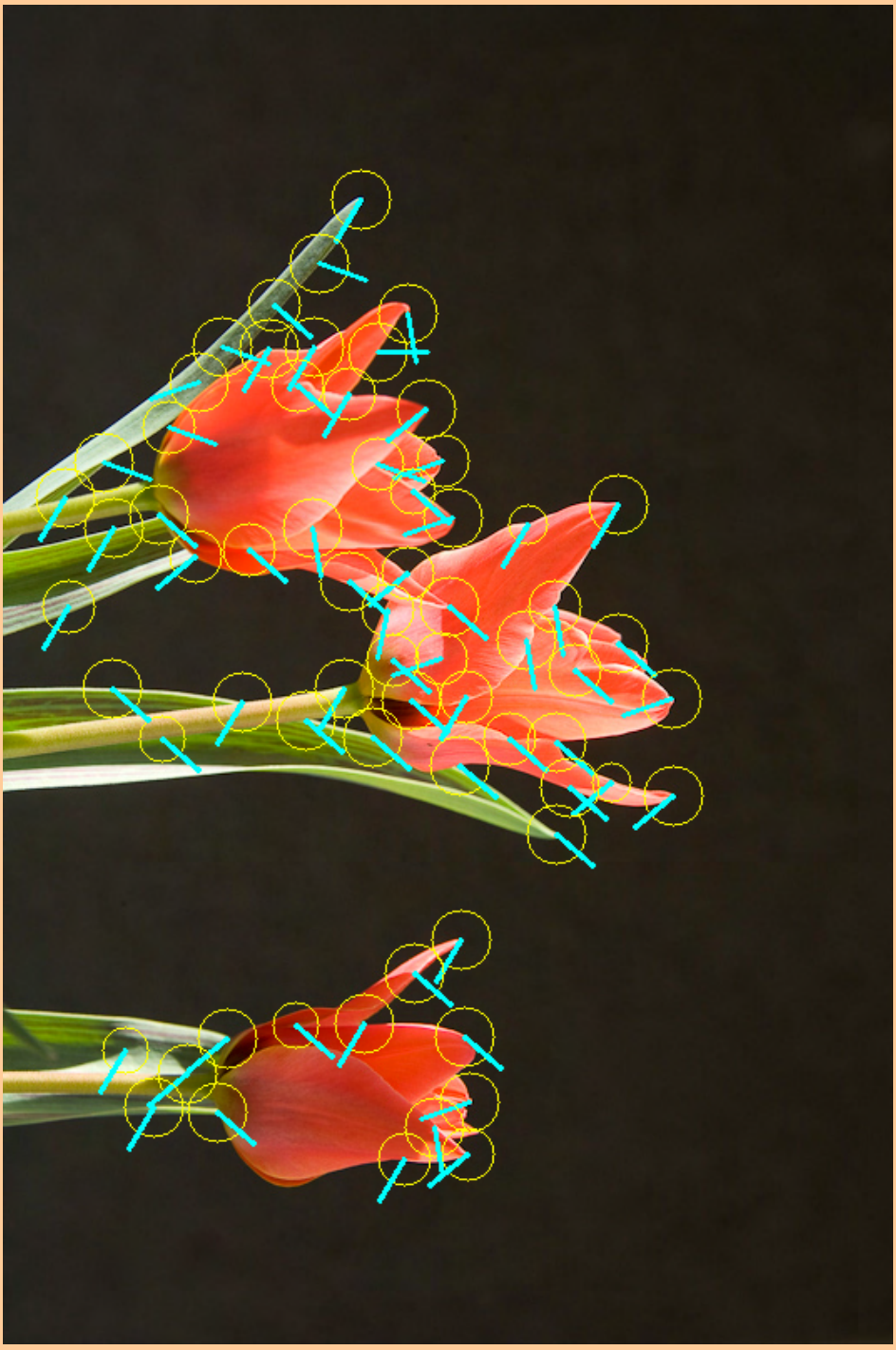
The image shifted about 20% to the right and cropped



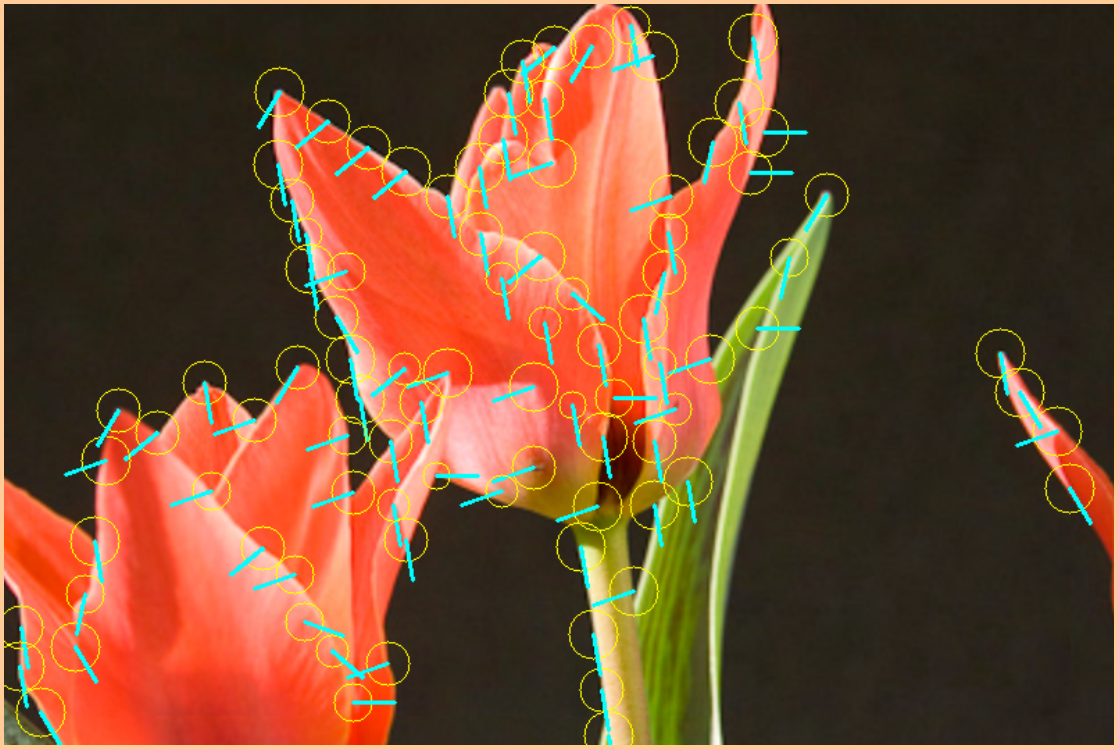
The image rotated by 90 degrees counterclockwise



The image rotated by 90 degrees clockwise



The image enlarged by a factor of 2 and center cropped



Discussion:

I tried different window sizes for checking the corners around pixels and window size of [10,10] gave really good results for all images. Smaller window sizes did not give good results and too large window sizes gave inconsistent results across different scales because more neighborhood pixels caused corner points to bunch up inside the window of a single corner point.