CS 4/510: Computer Vision & Deep Learning, Summer 2022

Programming #1

A. Rhodes

---

*Please email your programming write-up (pdf) and source code in a single zip file to our TA by the due date: **Friday 7/8 @ 10pm**.

Each student will turn in an individual assignment. You are allowed to use any high-level programming language of preference, however the use of Python is strongly encouraged (C++, Java, Matlab, etc., are also fine). You may use standard libraries (NumPy, math and plotting libraries, etc.) for this assignment, with the exception that the **main algorithm in each part** (as clearly identified in the instructions) **must be written explicitly** by you. In other words, for example, in Part 1 you can't simply invoke the function applying Sobel filters from the Opencv library, etc.

This assignment consists of (3) parts:

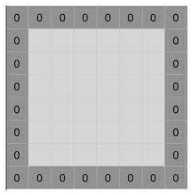
Part 1

Dataset: Two test images are provided: filter1_img.jpg and filter2_img.jpg; the images are available on Canvas.

(i) For the two test images provided, write a simple program to explicitly apply a *Gaussian filter*. Use the following standard 3x3 and 5x5 Gaussian filters, given by:

$$\frac{1}{16}\cdot\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \qquad \frac{1}{273}\cdot\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

This means that you will need to convolve each filter with different patches in the test images; please use a stride=1 in your code and apply zero padding of width 1 when using the 3x3 filter and zero padding of width 2 for the 5x5 filter. For instance, zero padding of width 1 results in:



Zero-padding added to image

(ii) For the same test images, write a program to apply DoG (*derivative of Gaussian*) g_x and g_y filters (use stride=1 and padding=1):

$$g_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, g_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(iii) Finally, compute the result of the *Sobel filter* for the test images:

$$\mathbf{X} * \mathbf{G} = \sqrt{\left(\mathbf{X} * \mathbf{g}_x\right)^2 + \left(\mathbf{X} * \mathbf{g}_y\right)^2}$$

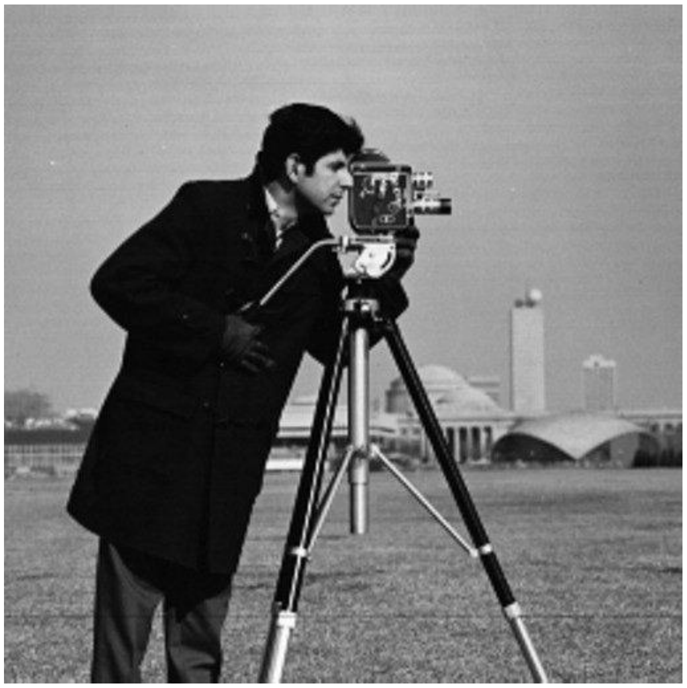
where **X** represents the test image.

*For part 1, in your write-up show the "before" and "after" of the original image(s) and the resultant image(s) after applying the stated convolutions.

- In this question, a program is developed to get some portions of the image as a matrix and it is multiplied with various filters. For each filter, the multiplying matrix is calculated to convolve with the portion of the image. The output generated with different filters present the blur and edge detection where impact of calculation transforms the 0 to 255 value. It's an interesting mathematics on the images that is able to give us insights as blur and edge detection.

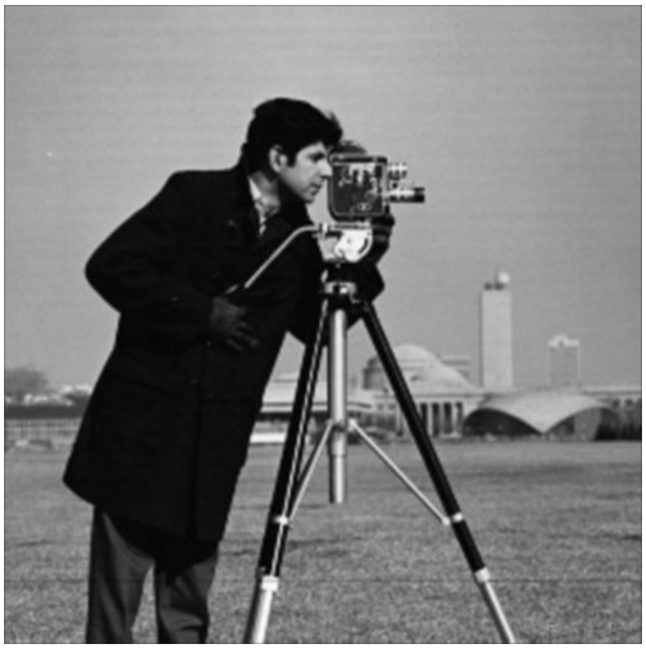
Image 1

Before

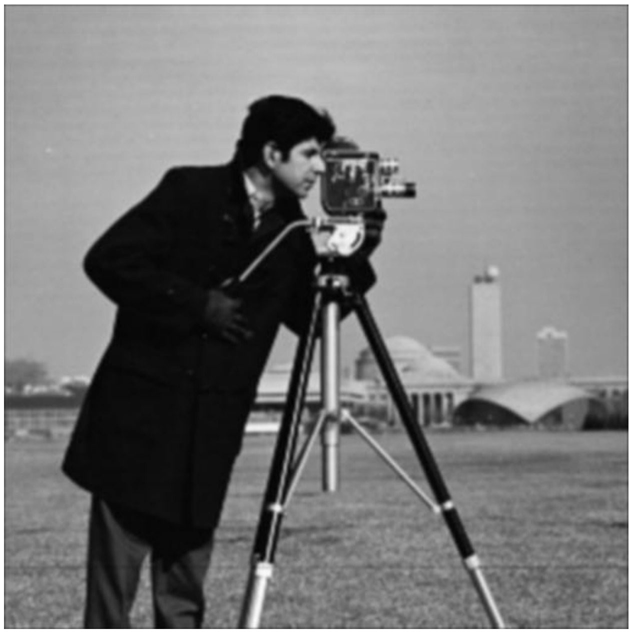


After

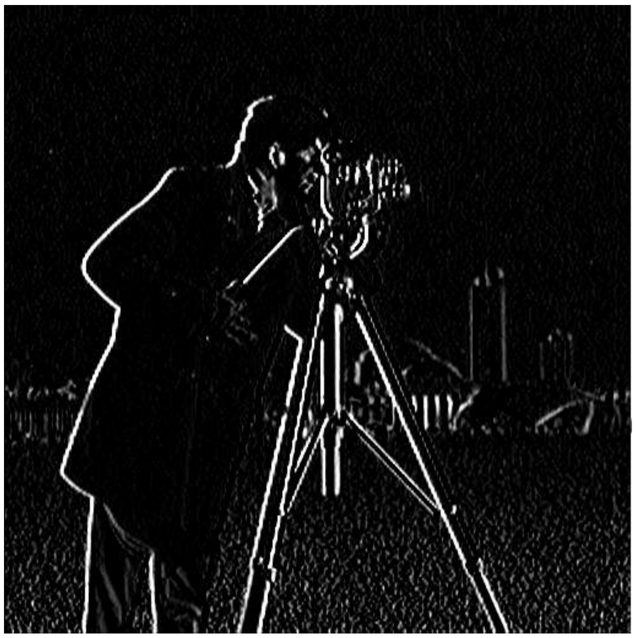
3x3 Gaussian Filter



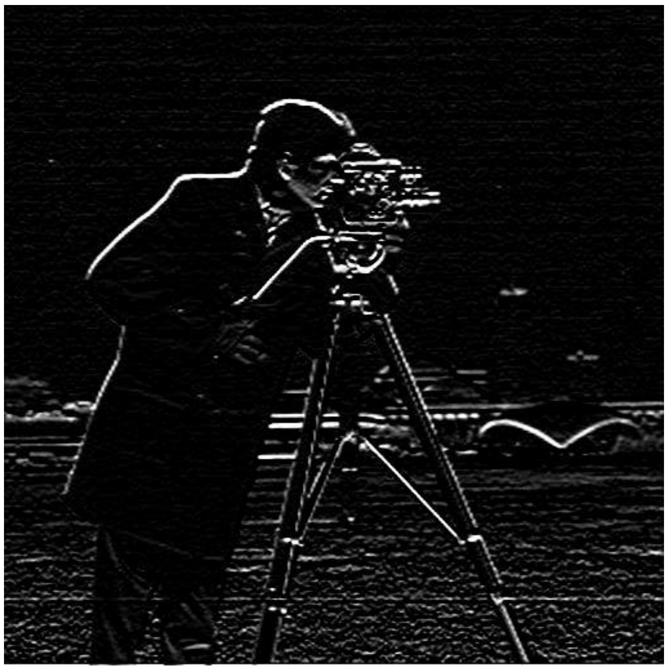
5x5 Gaussian filter



Derivative of the gaussian x filter



Derivative of the gaussian y filter



Sobel Filter

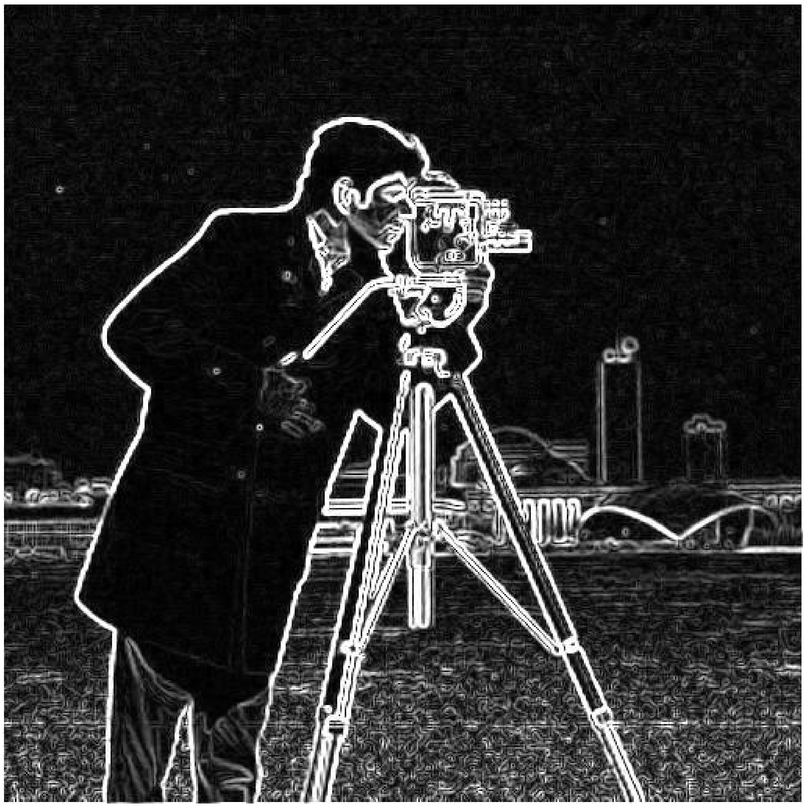
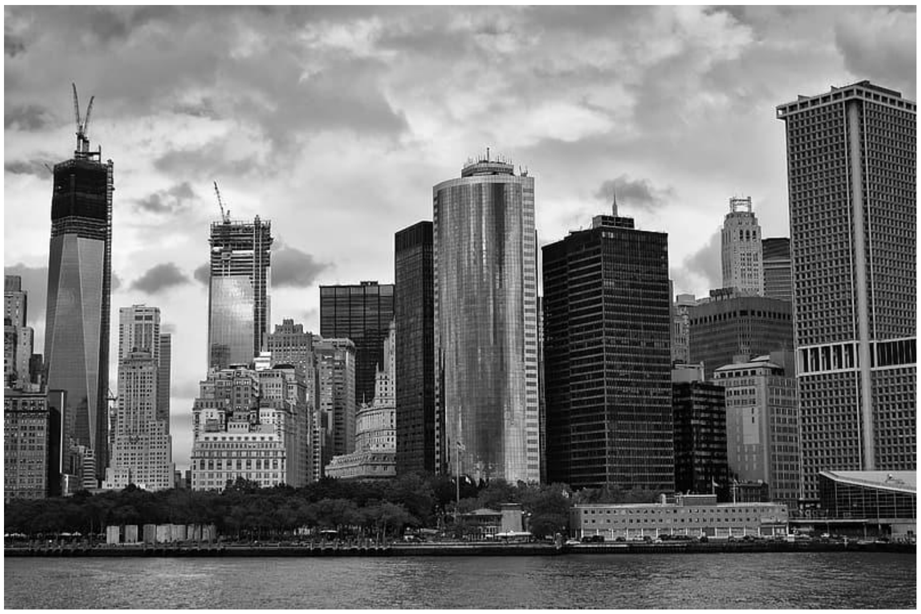


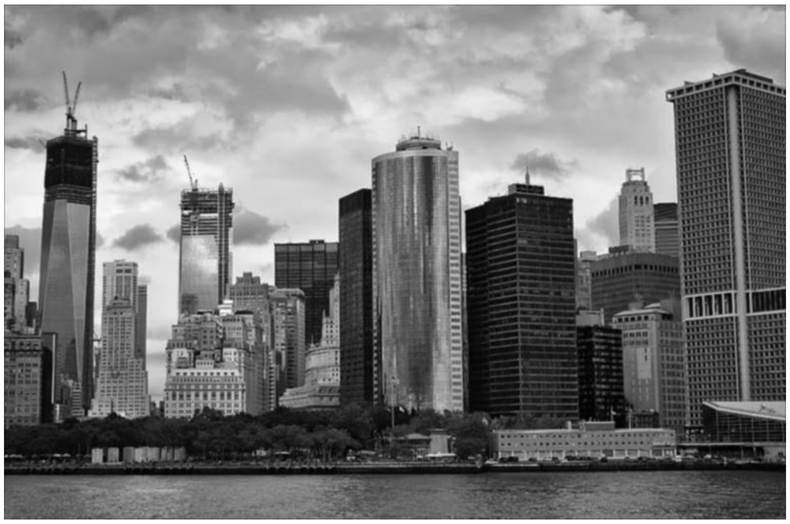
Image 2

Before

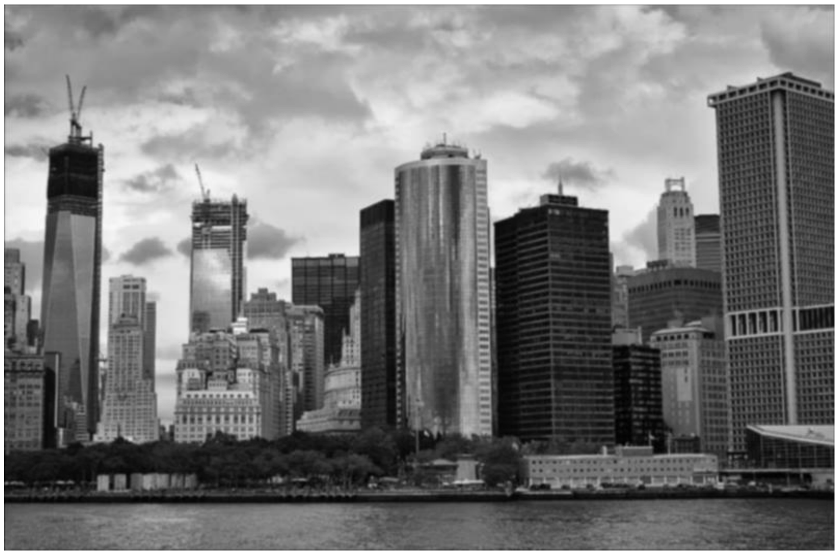


After

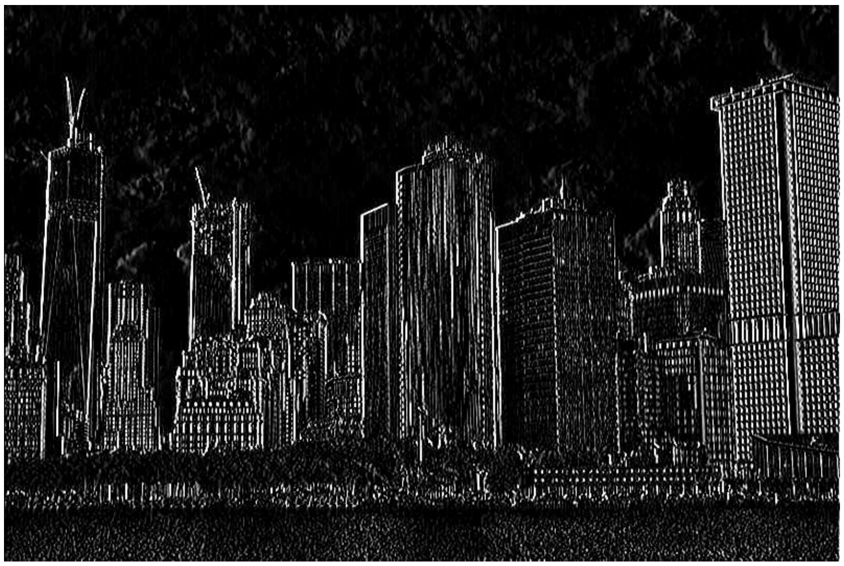
Gaussian filter 3x3



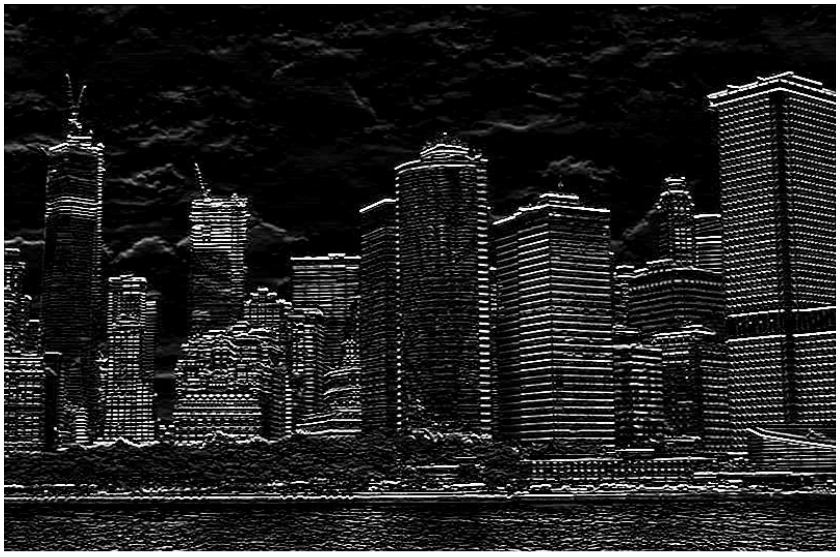
Gaussian filter 5x5



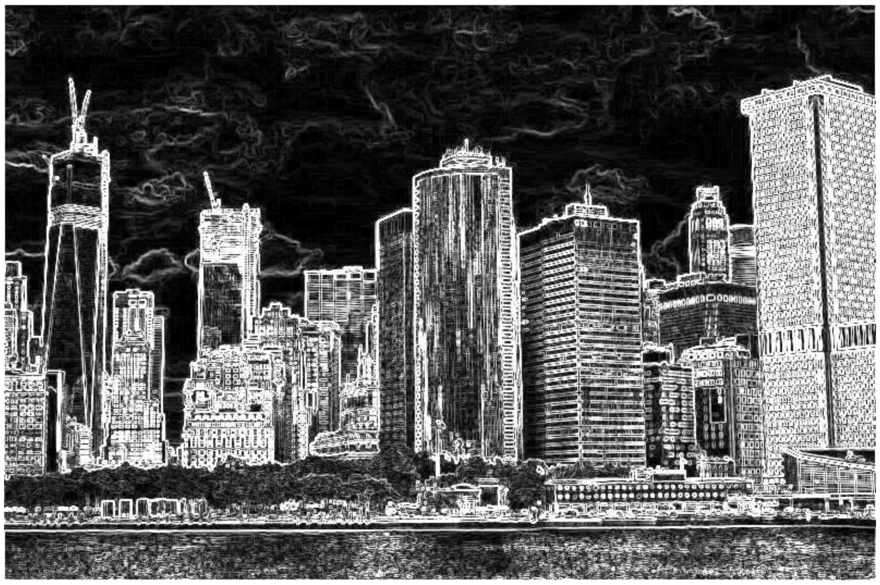
Derivative of Gaussian X filter



Derivative of Gaussian Y filter



Sobel Filter



Part 2

Dataset: A 2D dataset generated by sampling 3 Gaussian distributions is provided. There are 500 points from each Gaussian, ordered together in the file. The dataset is posted on Canvas with this assignment.

(i) Implement the standard version of the K-Means algorithm as described in lecture. The initial starting points for the K cluster means can be K randomly selected data points. You should have an option to run the algorithm *r* times from *r* different randomly chosen initializations (e.g., *r* = 10), where you then select the solution that gives the lowest sum of squares error over the *r* runs. Run the algorithm <u>for K=2, K=3, and K=4 and report the sum of squares error for each of these models</u>. Please include a 2-d plot of several different iterations of your algorithm with the data points and clusters.

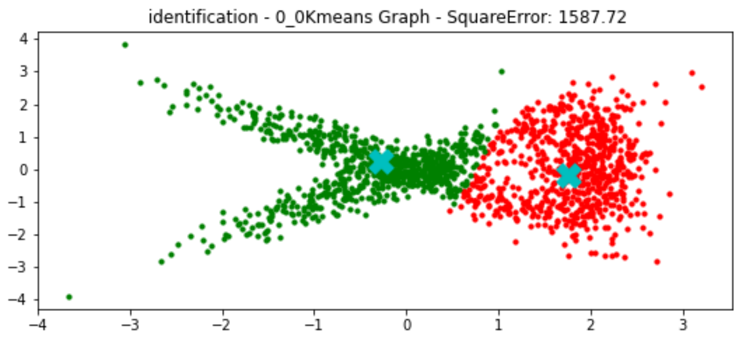
(ii) For the two test images provided: Kmean_img1.jpg and Kmean_img2.jpg, execute your K-means algorithm from part 2(i) in the RGB color space (i.e. in 3D space) with K=5 and K=10, respectively.

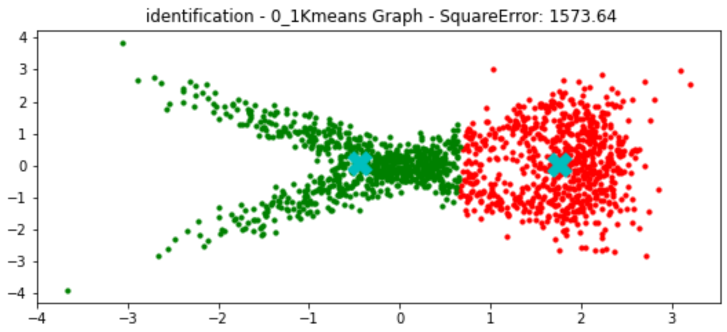
*For part 2(i), in your write-up include the visualizations of the execution of K-means for K=2, 3, and 4 on the Gaussian data; for part 2(ii), include a visualization of your K-means clustering in the color space of each image for K=5 and K=10.

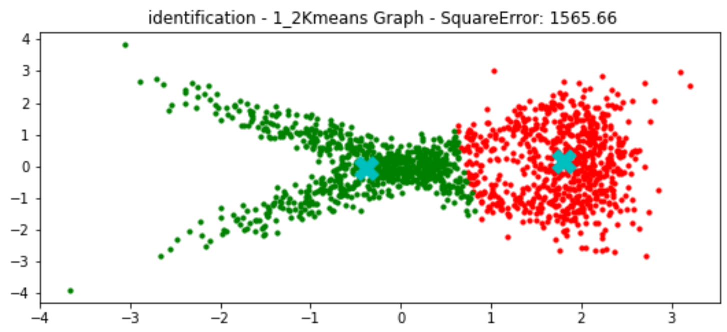
In this programming question, K means algorithm is applied which consists of 3 parts creating random center points from a given dataset, then dataset points are assigned the centers and the average of these point is found to update the centers till defined iterations and random point assignment is also updated as per the defined iteration. Furthermore, square sum error is calculated to present the error rate change in updation. For the dataset, firstly gaussian distribution points are used and then images are used.

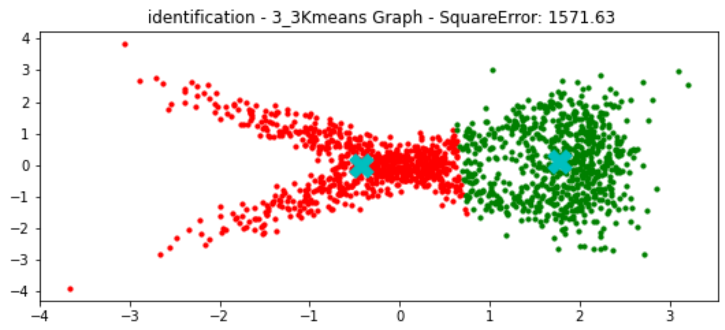
Q 2 - Part 1

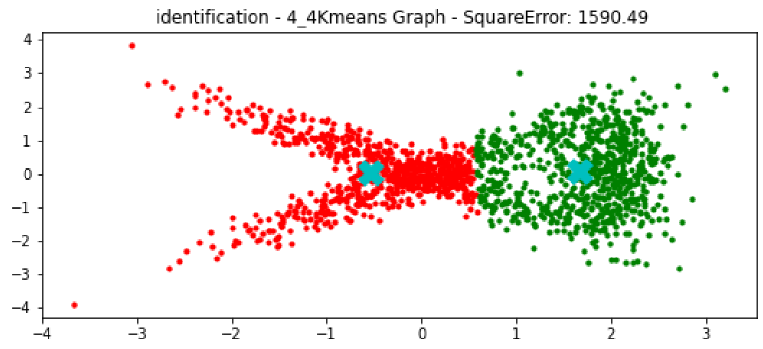
For K = 2



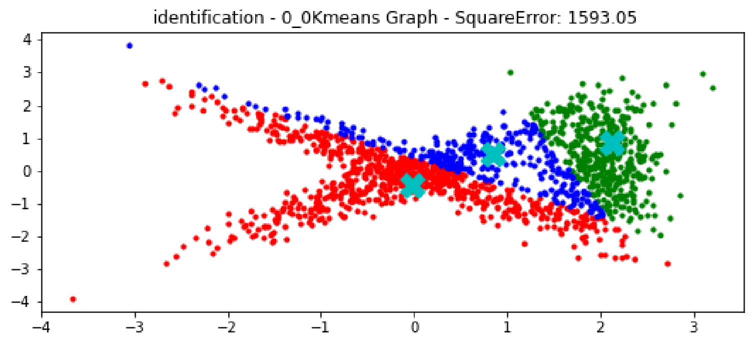


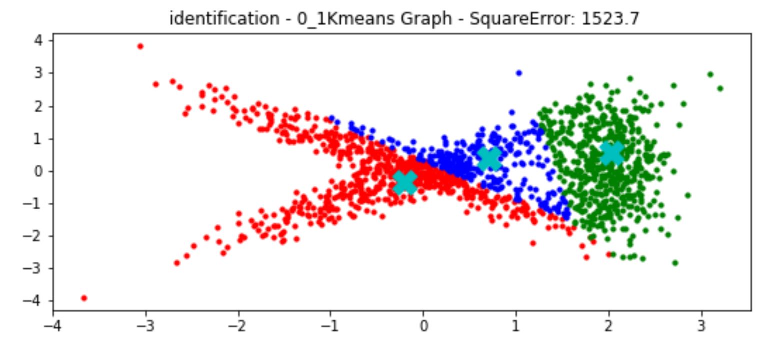

identification - 1_2Kmeans Graph - SquareError: 1565.66

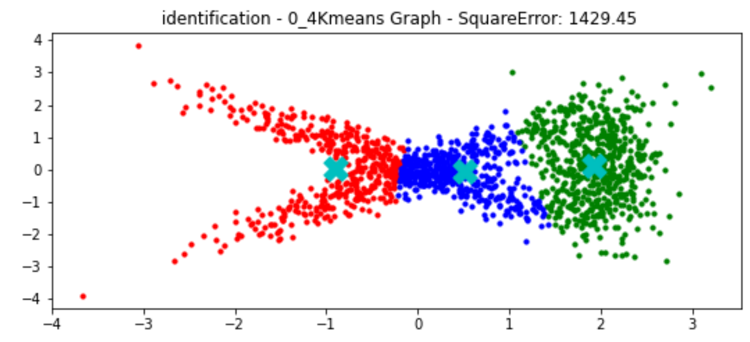

identification - 3_3Kmeans Graph - SquareError: 1571.63

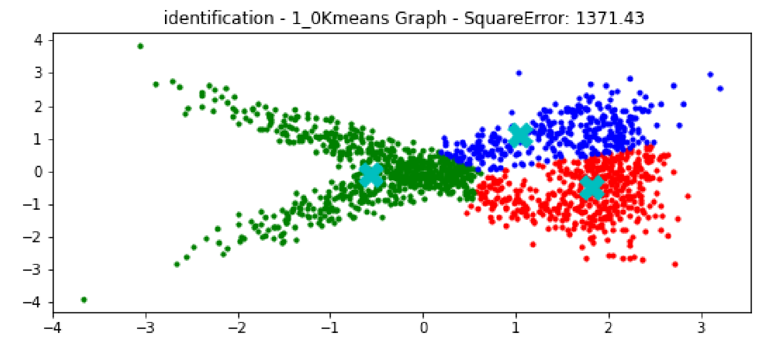

identification - 4_4Kmeans Graph - SquareError: 1590.49

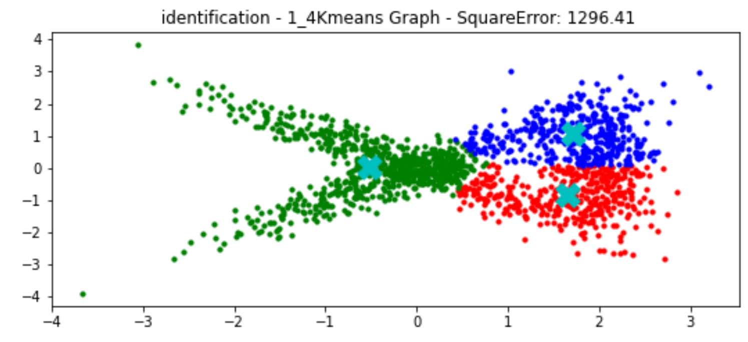
For K = 3,


identification - 0_0Kmeans Graph - SquareError: 1593.05

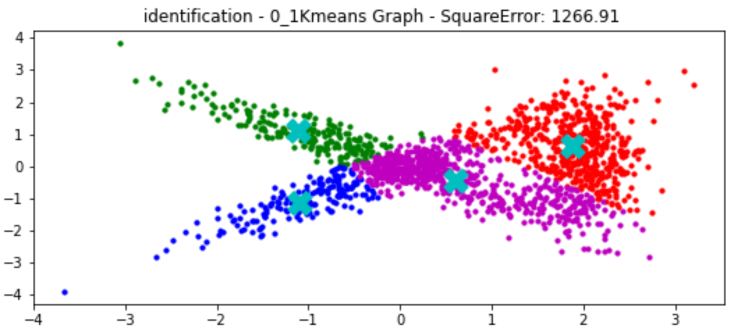

identification - 0_1Kmeans Graph - SquareError: 1523.7

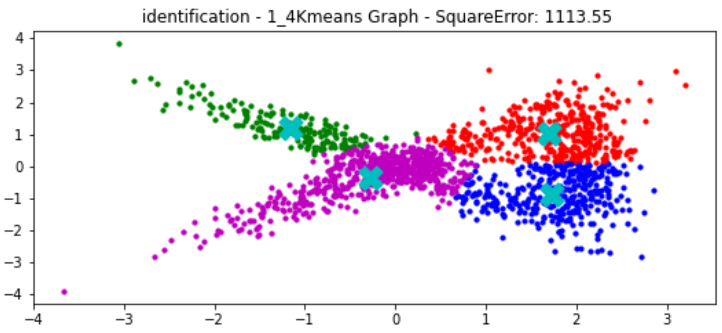

identification - 0_4Kmeans Graph - SquareError: 1429.45

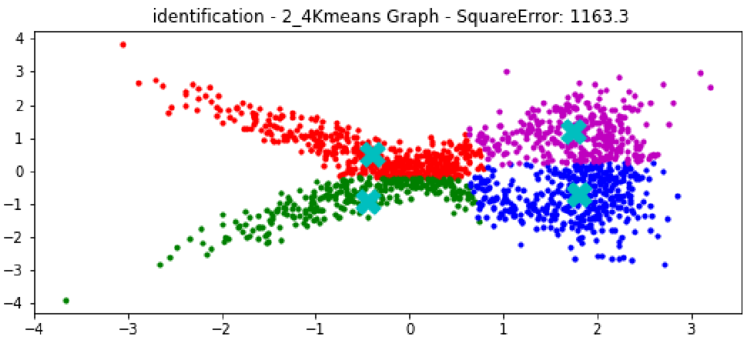

identification - 1_0Kmeans Graph - SquareError: 1371.43

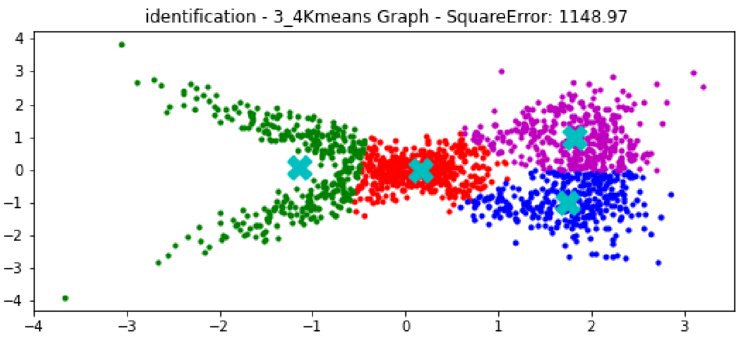

identification - 1_4Kmeans Graph - SquareError: 1296.41

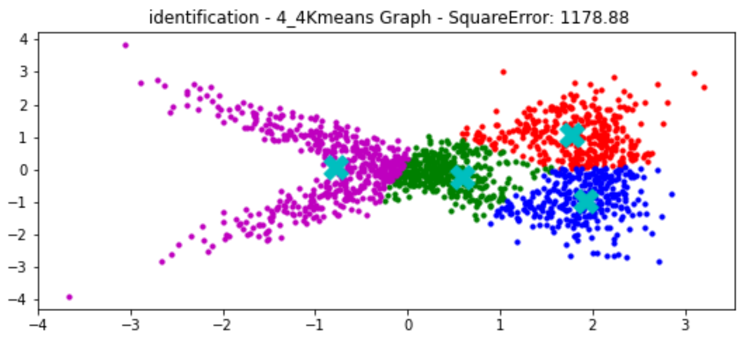
For K = 4


identification - 0_1Kmeans Graph - SquareError: 1266.91
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


identification - 1_4Kmeans Graph - SquareError: 1113.55
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

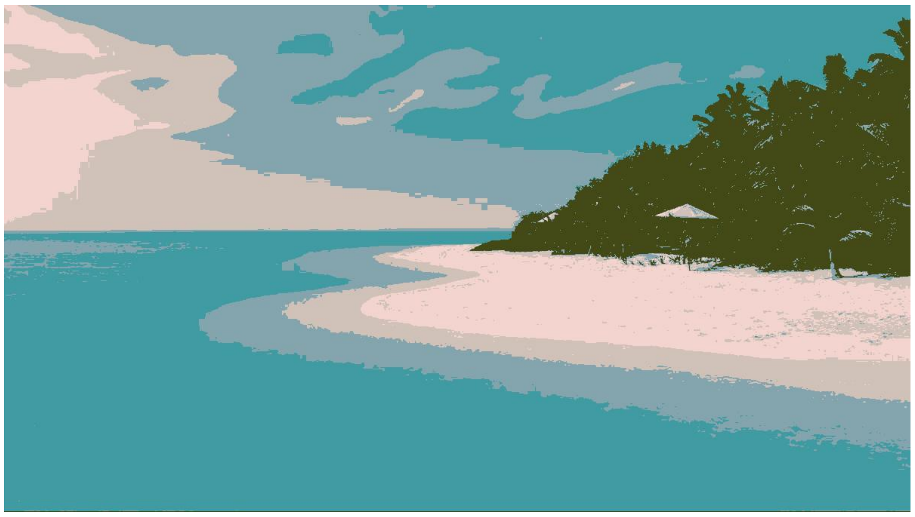

identification - 2_4Kmeans Graph - SquareError: 1163.3
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


identification - 3_4Kmeans Graph - SquareError: 1148.97
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

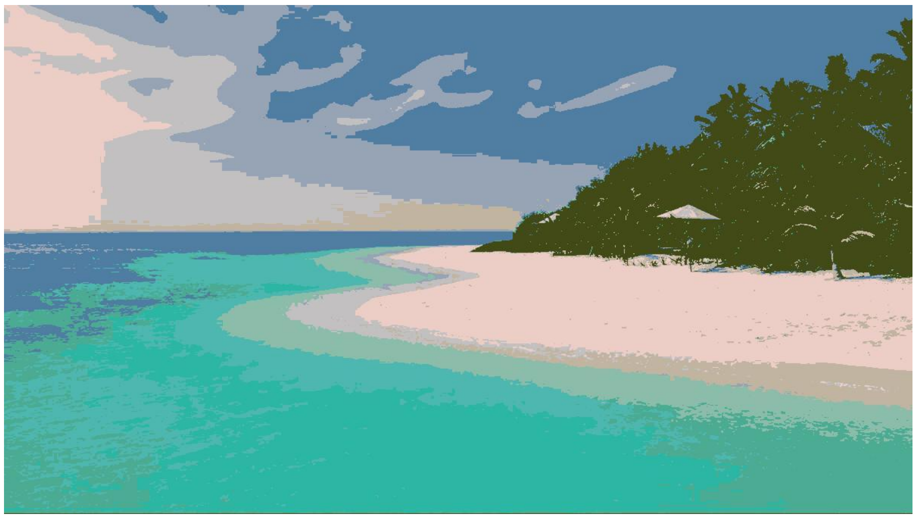


Q 2 - Part 2

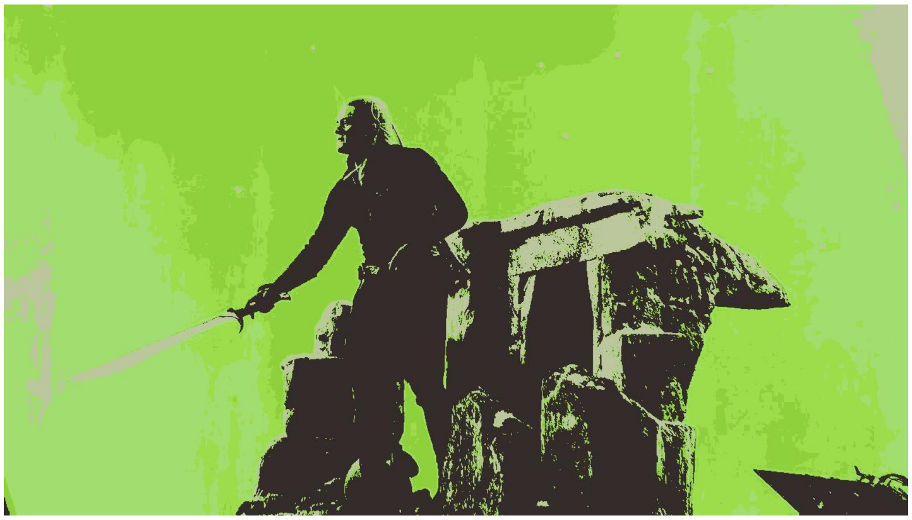
For K = 5



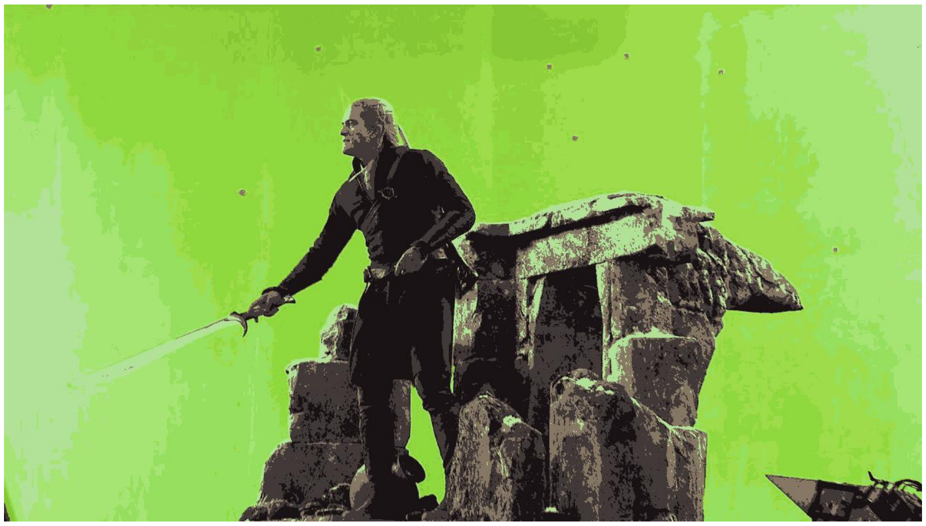
For K = 10



For K = 5



For K = 10



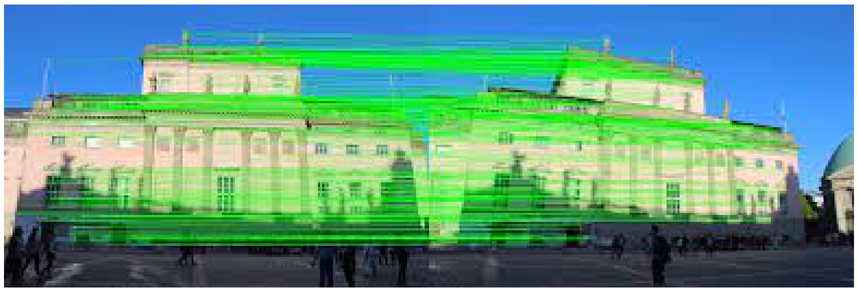
Part 3

Dataset: Two test images are provided: SIFT1_img.jpg and SIFT2_img.jpg.

(i) Using a built-in library (e.g., OpenCV), calculate the **SIFT feature**s for the test images. Visualize the SIFT features for each of the test images by superimposing the locations of these SIFT-based keypoints on each test image, respectively (if the library includes a visualization of the descriptor orientation, etc., you may also include this in the visualization).

From the generated set of SIFT keypoints for the first test image, determine the keypoint matches between the two images. To do this, calculate the nearest-neighbor (NN) for each keypoint detected in image 1 with respect to the set of keypoints detected for image 2. Here the nearest-neighbor criterion is the **minimum L2 distance** between the key point SIFT vectors. Next you will determine the "top 10%" of keypoint matches. Once you have determined the NN keypoint (in image 2) associated with each keypoint in image 1, determine the top 10% of the image 1 keypoints and their matches by retaining only the pairs of keypoints with that represent the top 10% per minimum L2 distance.

*For part 3, in your write-up include visualizations of all of the SIFT keypoints detected for the test images. After pruning the bottom 90% of matches, include a visualization of the top 10% of keypoint matches that correlates each of the remaining keypoints in image 1 with its corresponding keypoint in image 2 (this visualization can be done through a common color scheme so that matching keypoints have the same color, or by drawing lines between matching keypoints as shown below).



Scale Invariant Feature Transformation is a methodology used to create interest points which can be mapped to other images irrespective of orientation and magnitude. This interest points or patches are firstly detected using a derivative of gaussian distribution to analyze points or point of interest that can be present for same object in other images then in this algorithm using L2 norm meaning euclidean distance and finding nearest neighbor with more than 90% accuracy interest points are identified and matched.

Image 1 with Keypoints

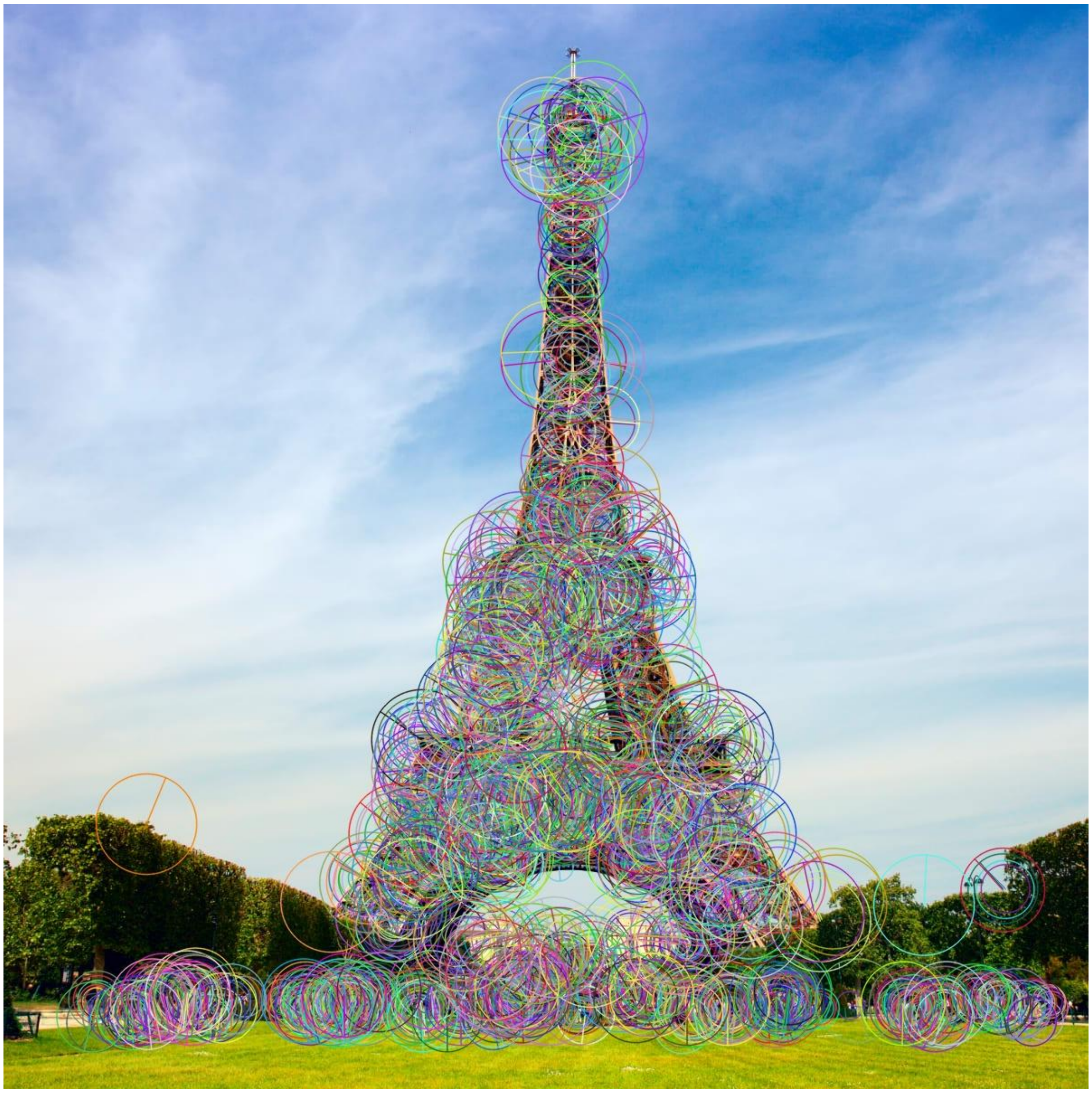
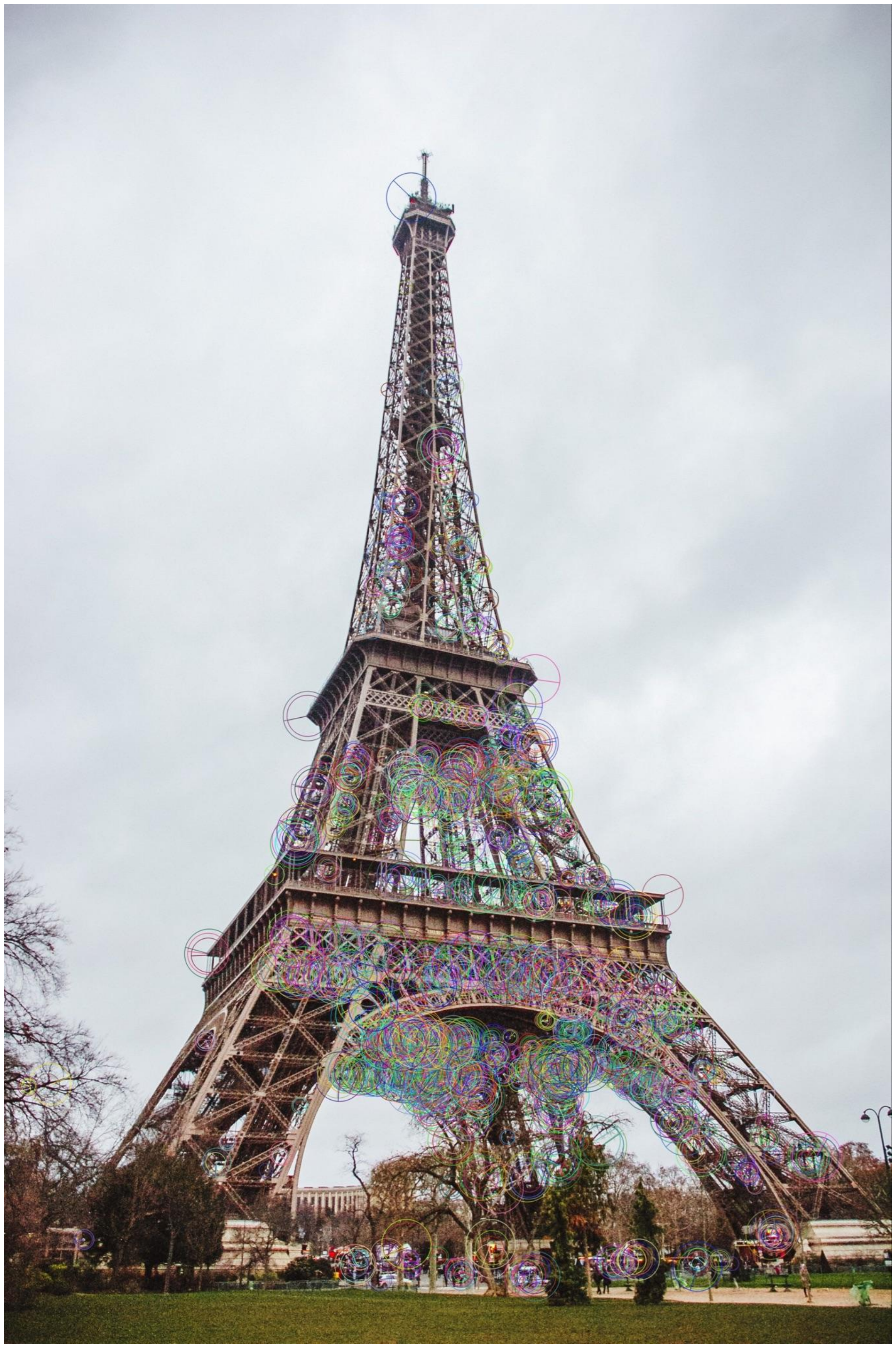


Image 2 with Keypoints



Matched Image