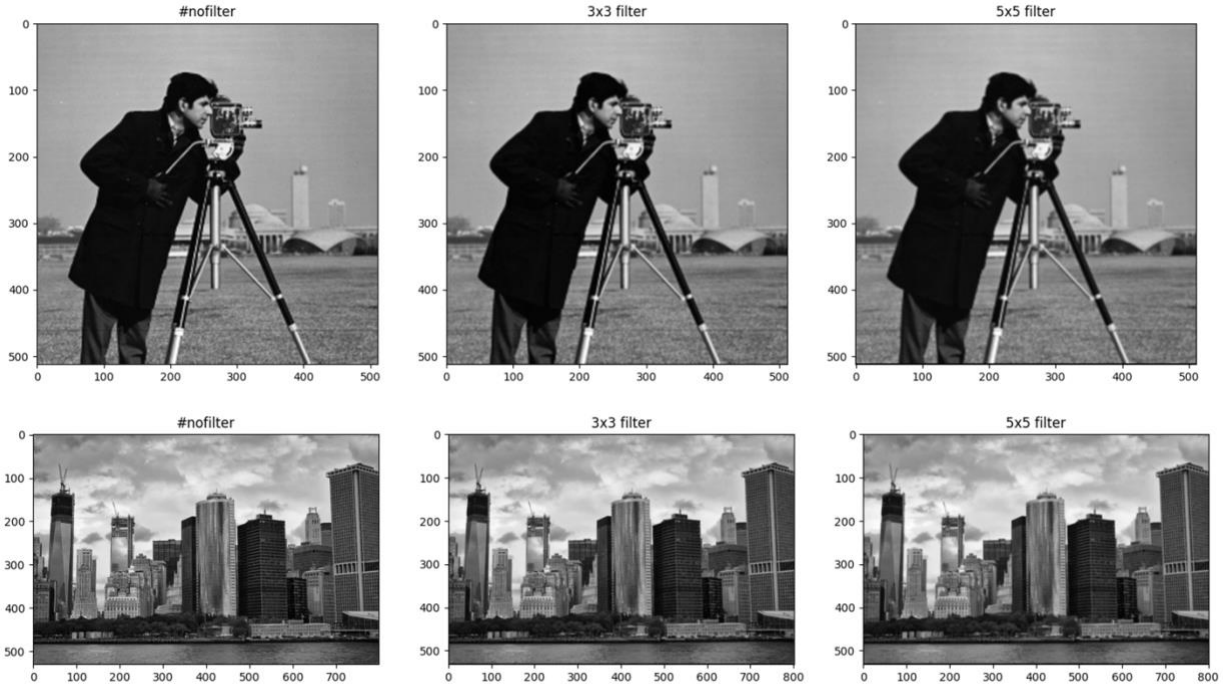
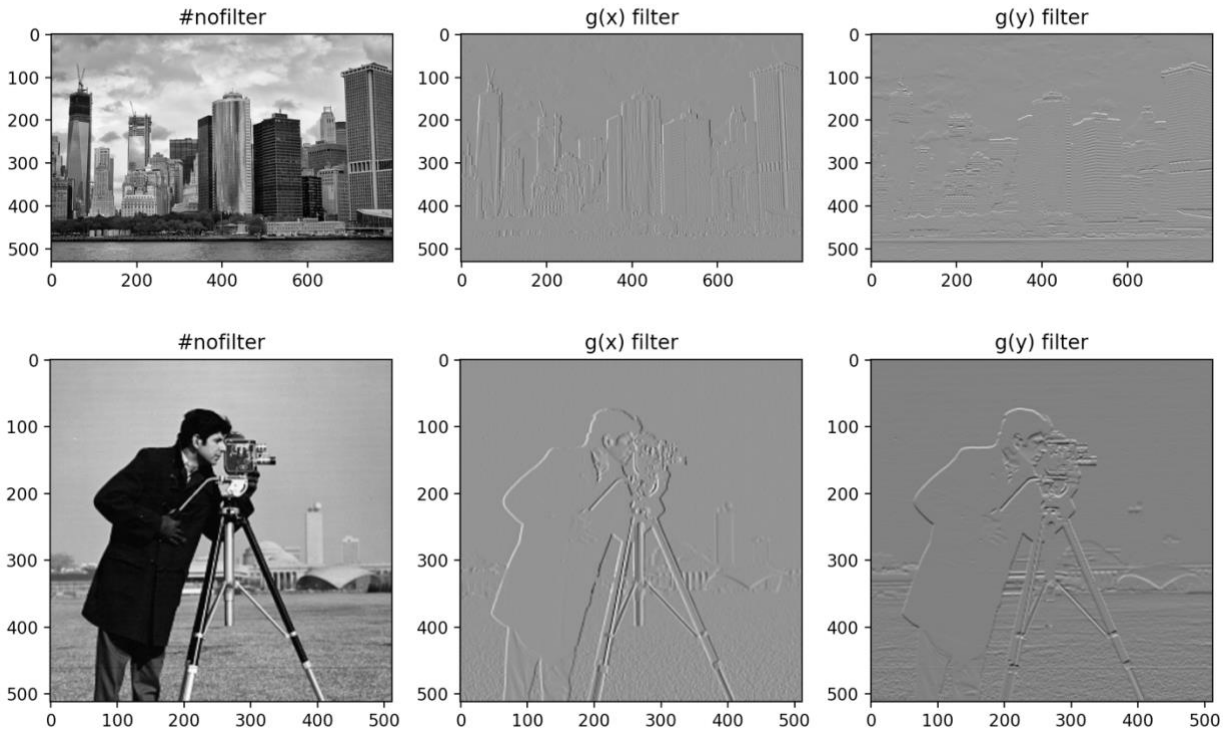
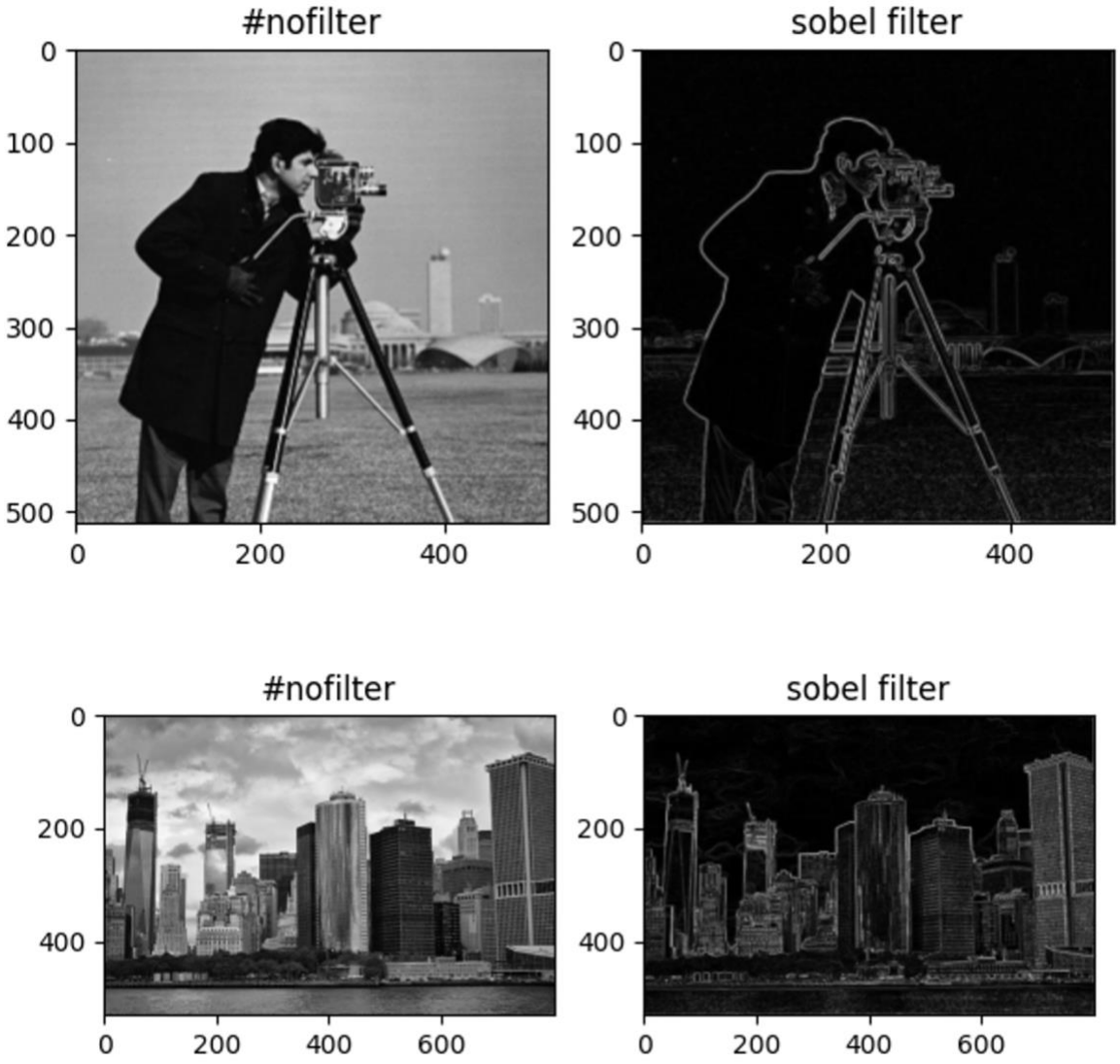
Ryan Filgas
CS510 Machine Learning
Programming 1



1a. For part 1a the images were run through a Gaussian filter using 3x3 and 5x5 filters. For both images this caused progressively more blur going from no filter to the 3x3, then the 5x5 filter. The resulting images are found using convolution. An image window around each pixel is multiplied by the filter matrix, and summed to create the new corresponding pixel in the output image. A 1 pixel padding was added to each image for the 3x3 filter, and 2 pixel padding was added for the 5x5 filter. The algorithm uses a stride of 1.

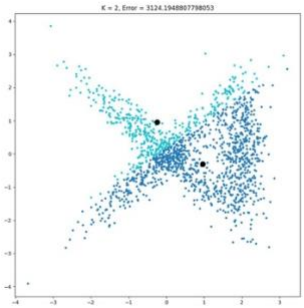


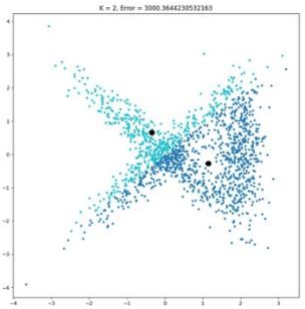
For part 1b derivative of gaussian was applied with the same convolution method using gx and gy filters and a stride of 1 and padding of 1. As can be seen in the examples they act as edge detectors in the x and y axis respectively.

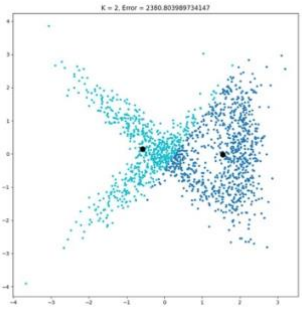


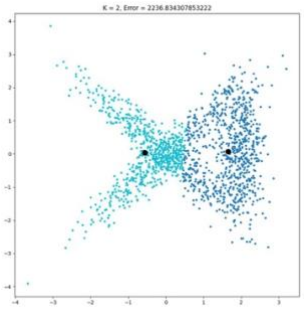
For part 1c the DoG values were used to compute and apply a sobel filter to each image. The filter has amplified stronger edges in both directions, and reduced weaker edges.

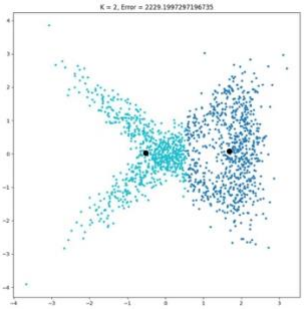
KMEANS: See comments at bottom of three runs.

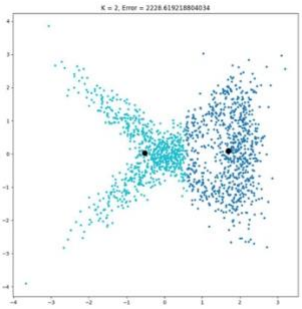


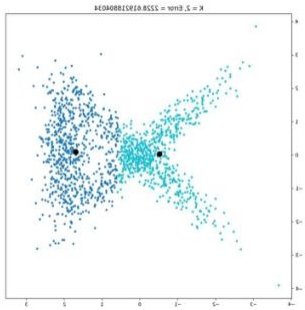


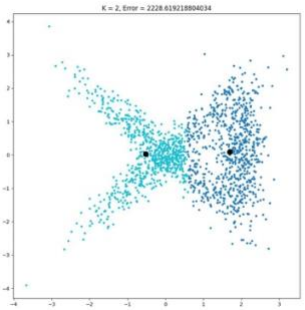


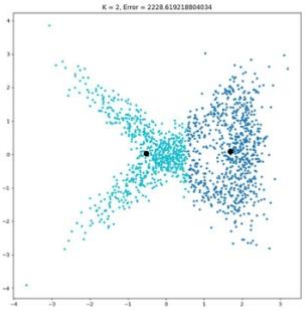


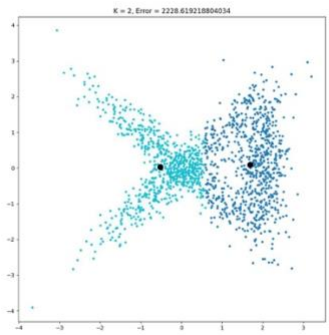


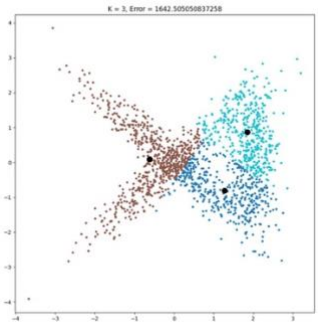


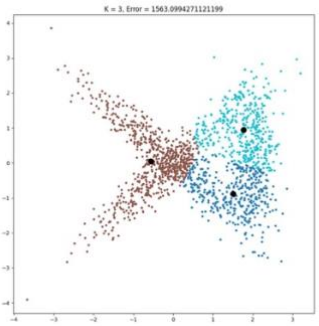


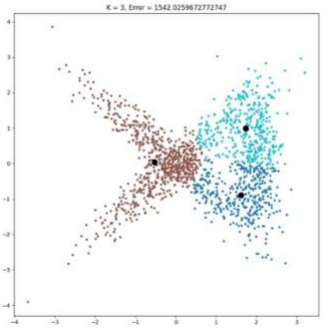


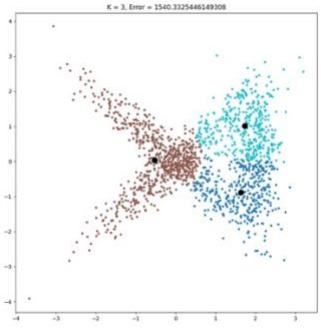


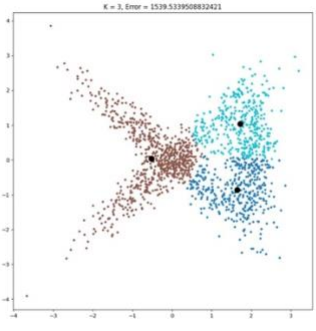


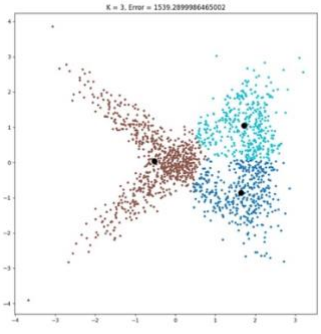

K = 3, Error = 1642.505050837258

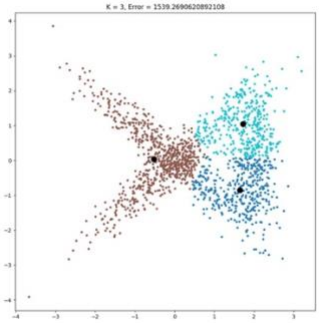

K = 3, Error = 1563.0994271121199

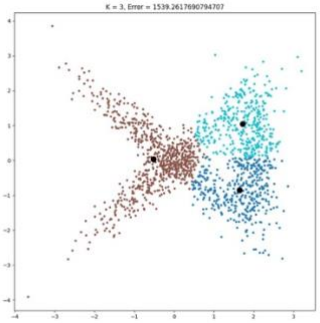

K = 3, Error = 1542.0259672772747

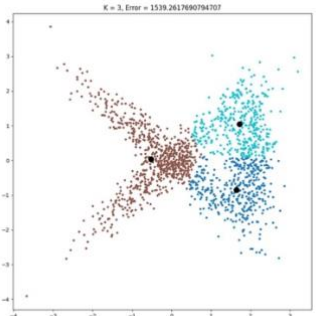

K = 3, Error = 1540.3325446149308

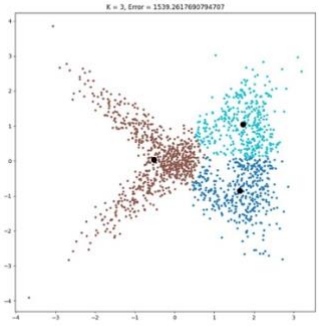

K = 3, Error = 1539.5339508832421

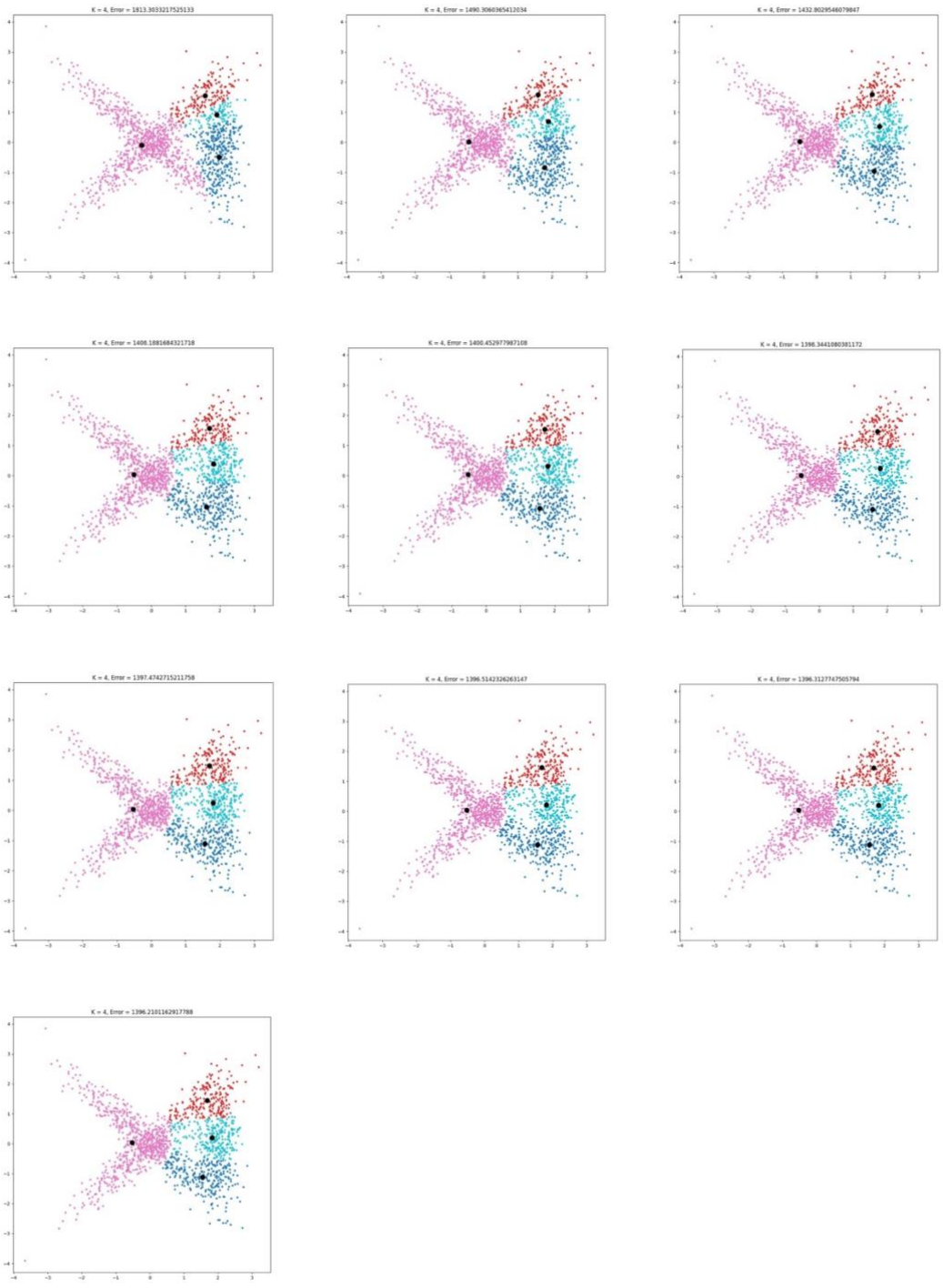

K = 3, Error = 1539.2899986465002


K = 3, Error = 1539.2690620892108


K = 3, Error = 1539.2617690794707


K = 3, Error = 1539.2617690794707

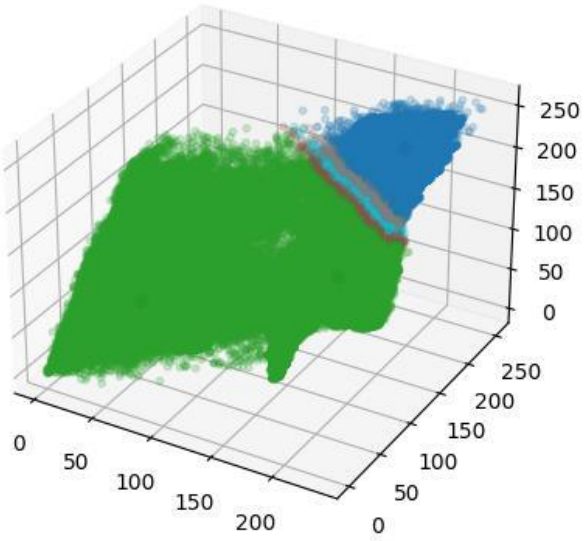

K = 3, Error = 1539.2617690794707



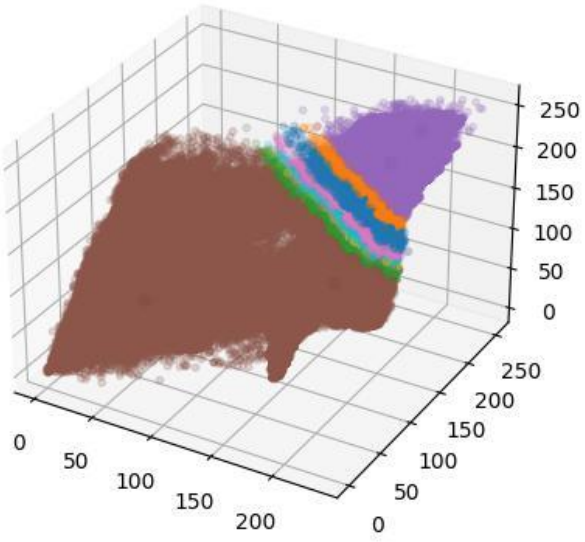
Part 2a: Kmeans was applied to a 2D gaussian dataset by starting with random points as centroids, assigning the closest points to each centroid into a cluster, and moving the centroid to the middle of that cluster; this process is repeated several times and the centroids can be seen converging in the examples.

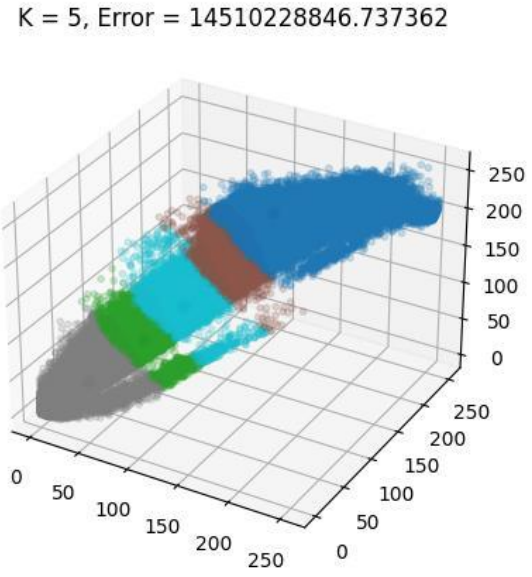
Part 2b: Kmeans on image data

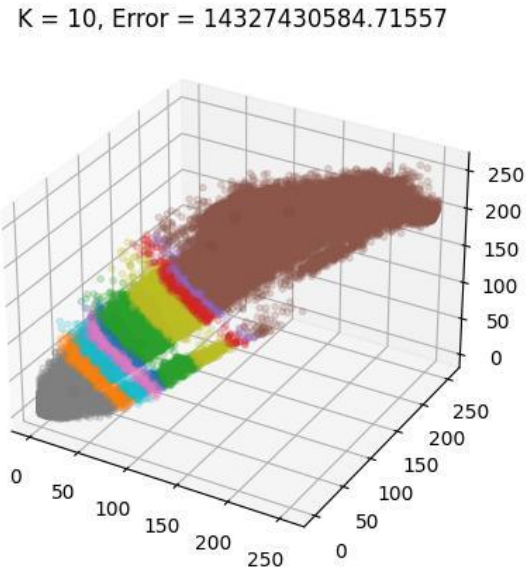
K = 5, Error = 4911707045.943958



K = 10, Error = 4727805595.765665

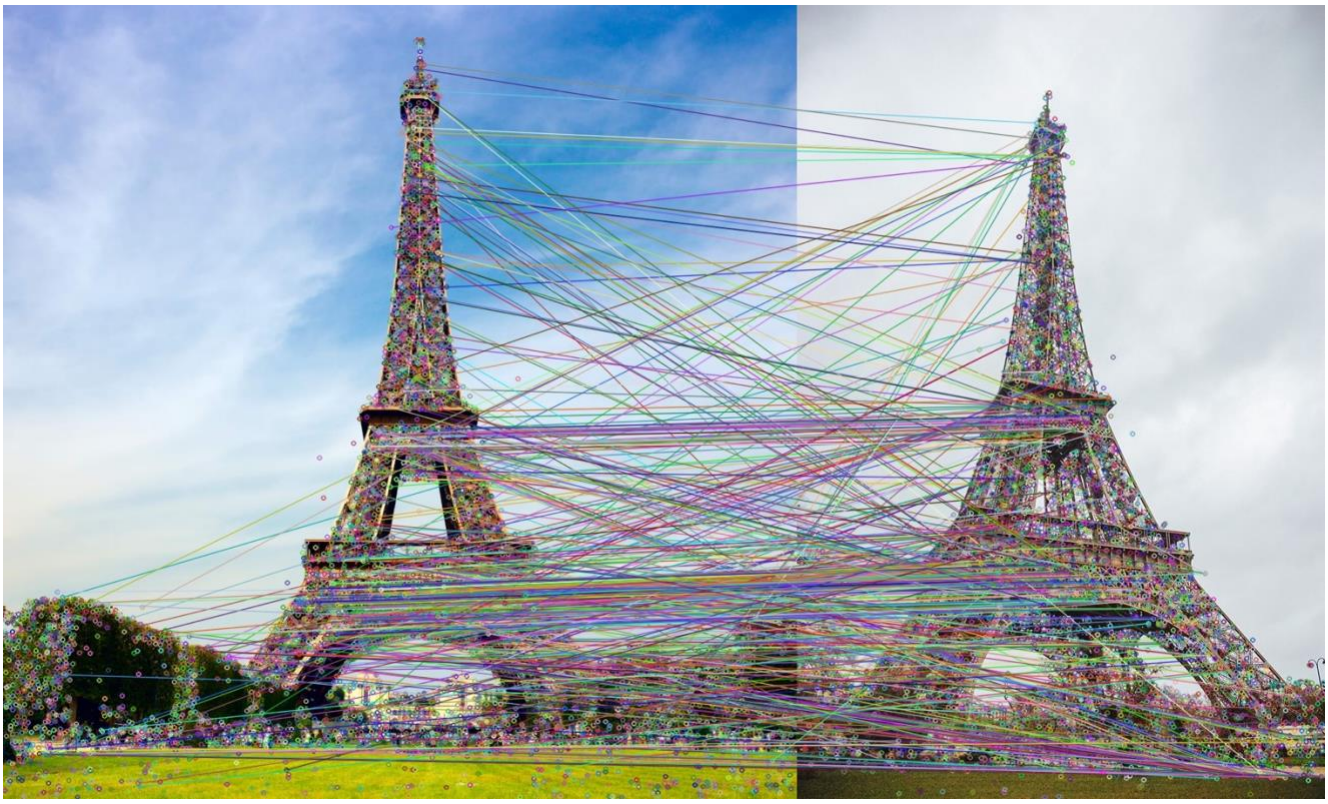






2b. The kmeans results for both images are curious. Image 1 was of a beach scene with a large amount of blue sky, and image two was a movie scene where much of the scene is a green screen. For the purposes of image segmentation this might suggest that the green screen and ocean/sky comprised the largest section of points in each plot. The algorithm for the 3d image information vs the 2d gaussian dataset was essentially the same, however I had to make major modification for efficiency. Even after some optimizations it still took 20 minutes per image in total processing time, so this is something that will need improving.

SIFT:



OpenCV was used to generate the descriptor vectors and key points for the images. Prior to running the algorithm I resized the images to have the same height for clarity in the generated result. In general the algorithm was effective at identifying similar features in each image, however it did also identify feature points that don't match. To calculate matches the algorithm iterates through each descriptor vector to find the closest ones in each image based on the minimum L2 distance. The 10% of matches with the lowest L2 distance are shown.