BILKENT UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CS 484
HOMEWORK 2

BERAT BİÇER
21503050

Dec. 2, 2018
ANKARA, TURKEY

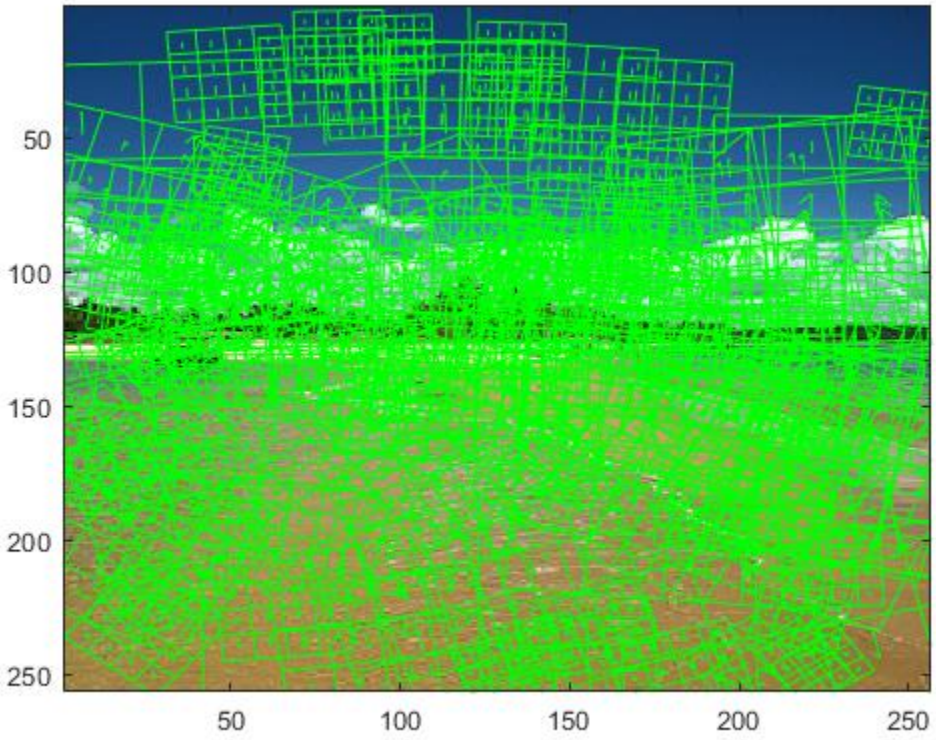
1 Introduction

This report includes description of homework 2 and its discussion, along with sample outputs at intermediate steps and final outputs. Project source code, which is written in MATLAB with 3rd party VLFeat library, is provided along with this report. For reference, **please execute the file *"main.m"* from inside MATLAB**. According to instructions provided on screen, provide appropriate inputs and wait for the output.

2 Detecting & Describing Local Features (3.2 & 3.3 in Homework)

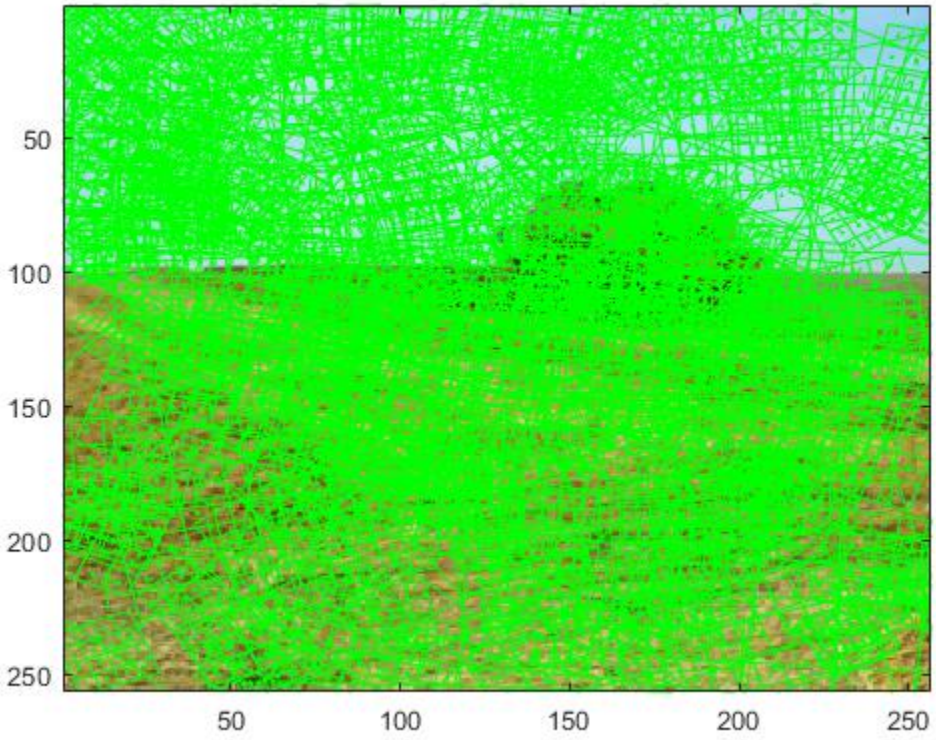
For detecting gradient-based descriptors, we used VLFeat library in MATLAB, specifically `[F,D] = vl_sift(image)` function to detect keypoints & compute their descriptors. `F` corresponds to image frames (which consists a 4-tuple: (X, Y, Scale, Orientation)) and `D` is the frame's descriptor. Each descriptor is a gradient-based descriptor in 128-dimensional vector format. Example descriptors are given below (visualized by `vl_plotsiftdescriptor(D, F)` function of VLFeat where `F` and `D` are outputs of `[F,D] = vl_sift(image)` function) for each category:

Category 1

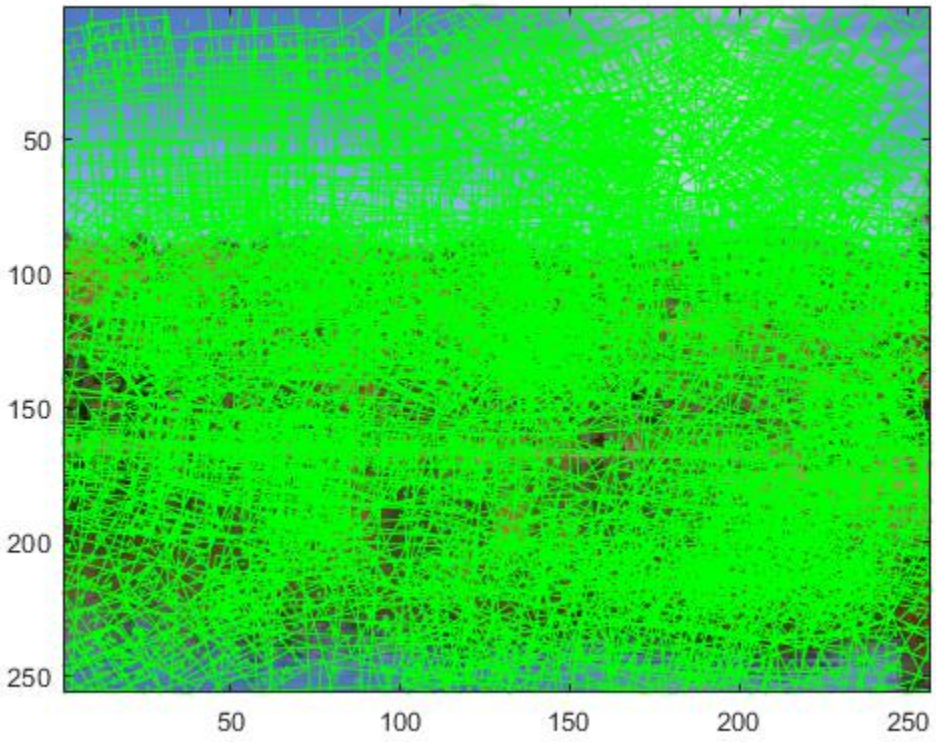


Category 2

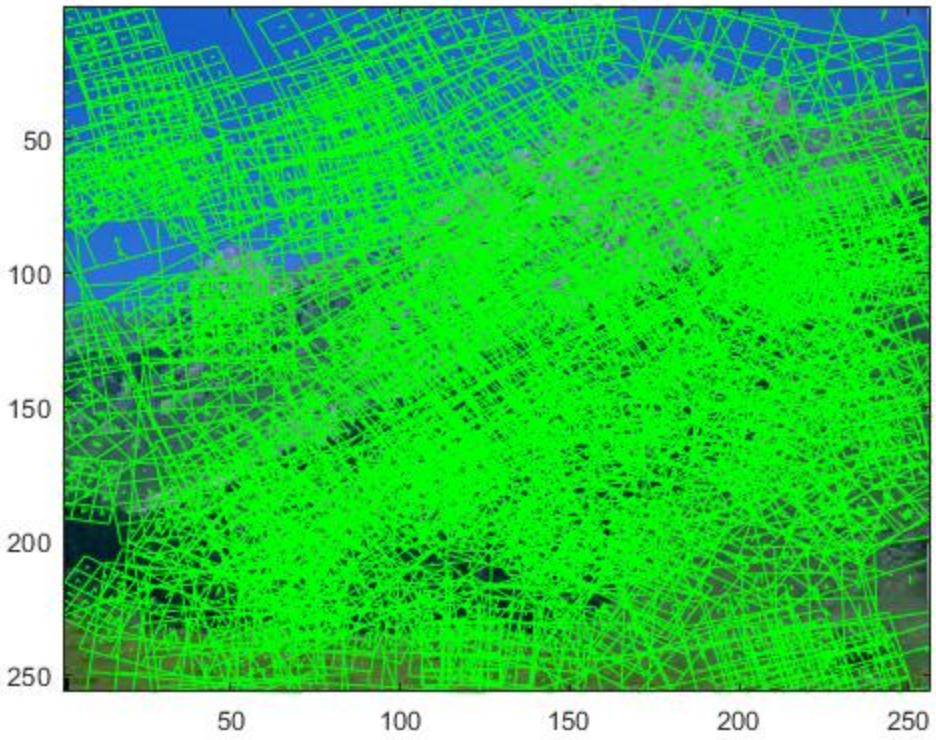
3


50
100
150
200
250
50
100
150
200
250

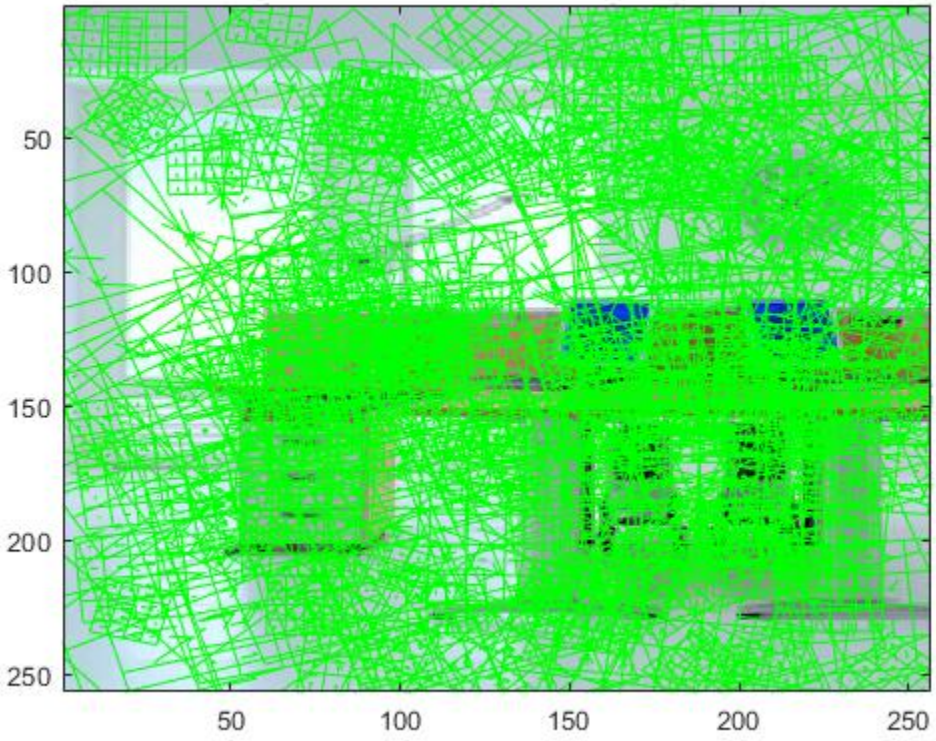
Category 3


50
100
150
200
250
50
100
150
200
250

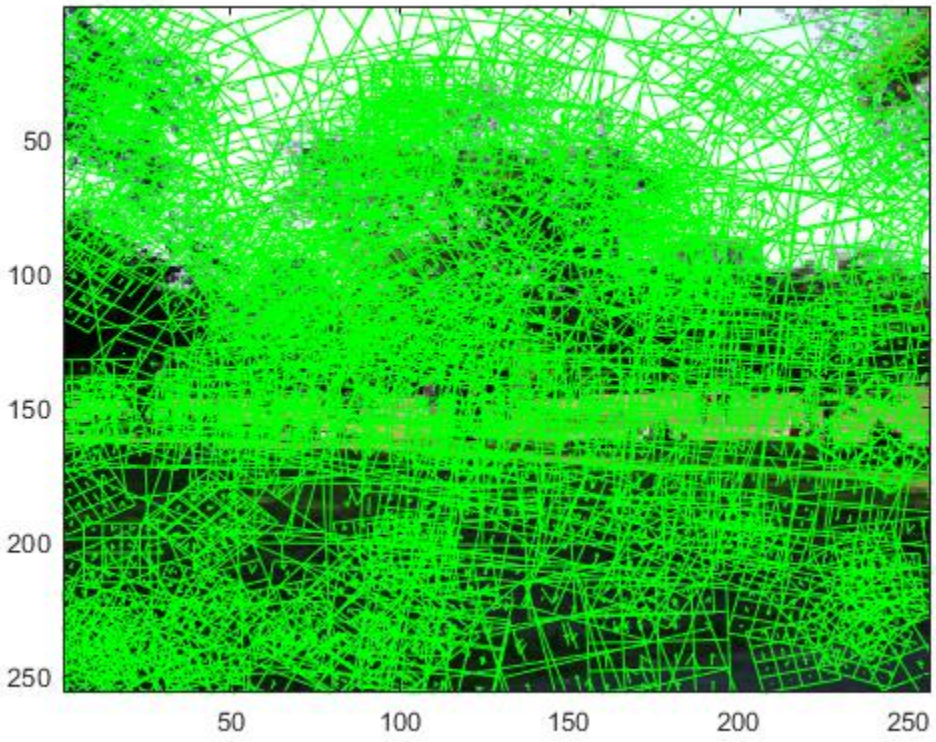
Category 4



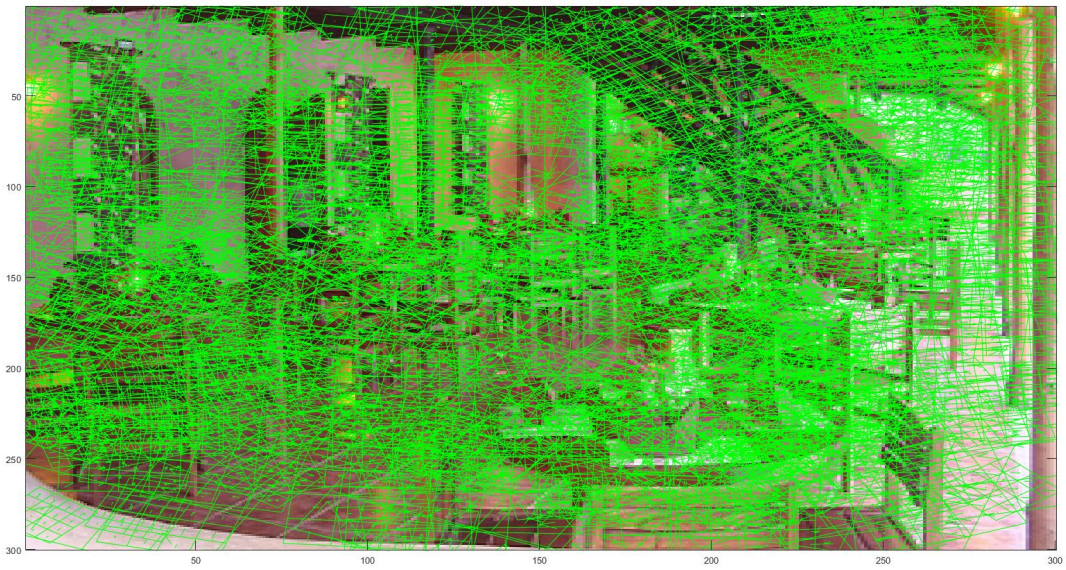
Category 5



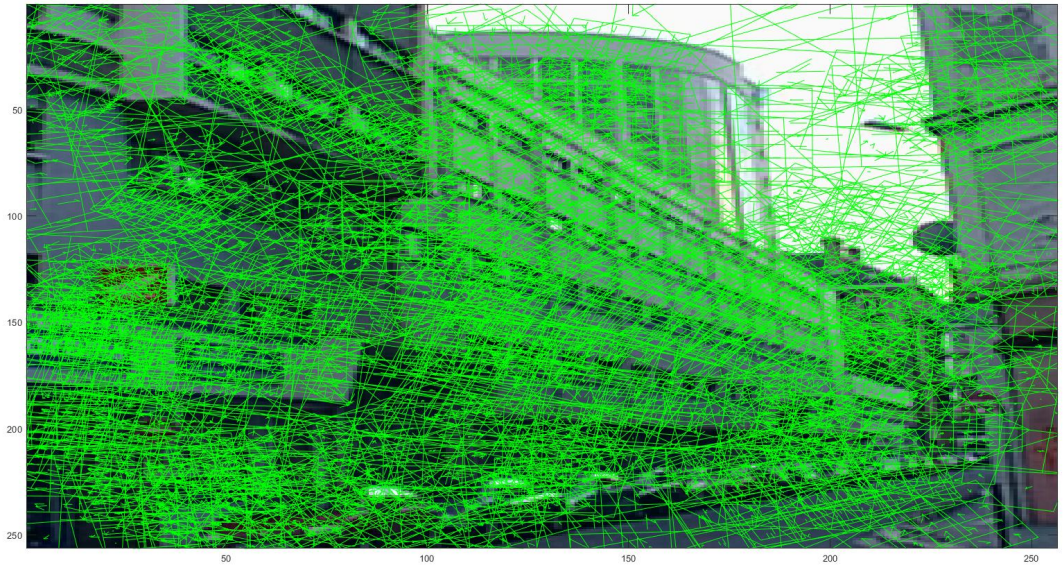
Category 6



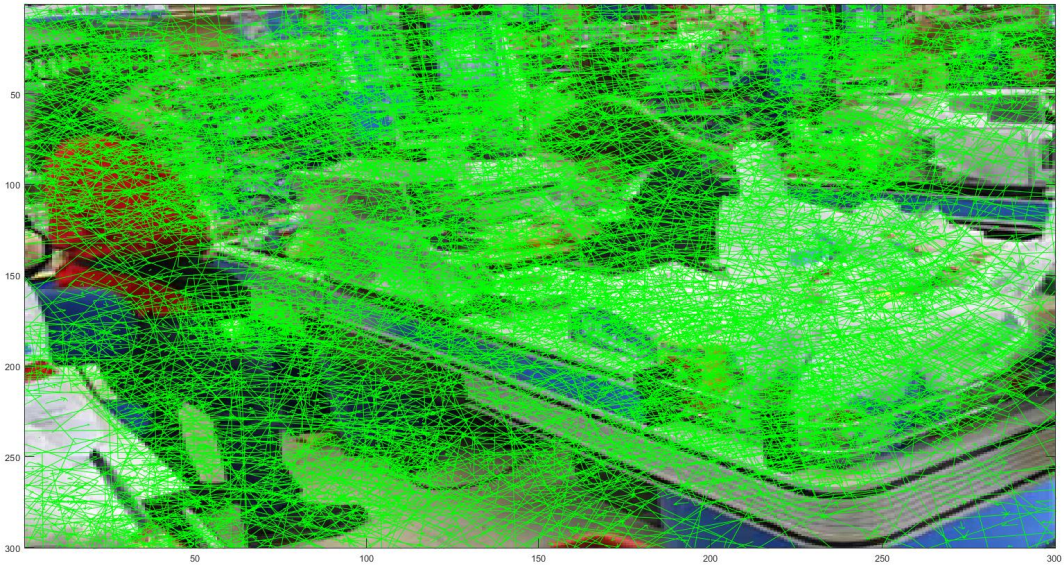
Category 7



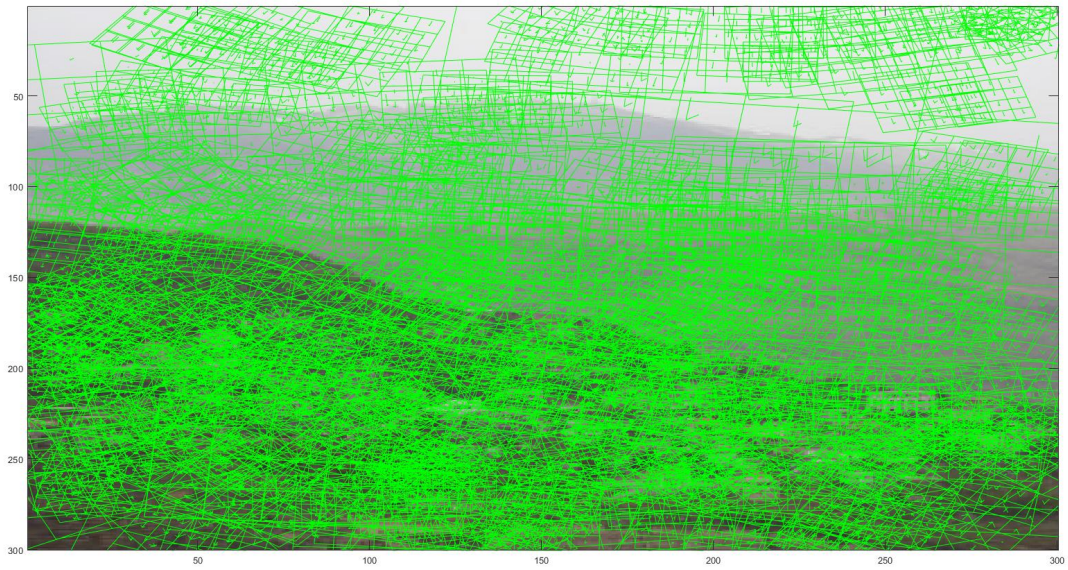
Category 8



Category 9



Category 10



For detecting color descriptors, we used `F = vl_sift(image)` function from VLFeat library to obtain keypoints of the image where `F` contains the following 4-tuple for each keypoint: `(x of center, y of center, scale, orientation.` Then, using the following algorithm we calculated the color histograms:

- First, detect the corner points before rotation as follows:

```
corner_topRight = [centerx + distance centery - distance];
corner_topLeft = [centerx - distance centery - distance];
corner_bottomLeft = [centerx - distance centery + distance];
corner_bottomRight = [centerx + distance centery + distance];
```

where `distance = scale * 6`, which is an empirical result: Bounding rectangles we draw to detect color histograms matches those VLFeat produces perfectly if we multiply the scale by 6.

- Then, rotate the corners by `orientation` degree CCW around the keypoint, using the following formula:

```
x′ = (x * cosθ – y * sinθ) + centerx
y′ = (x * sinθ + y * cosθ) + centery
```

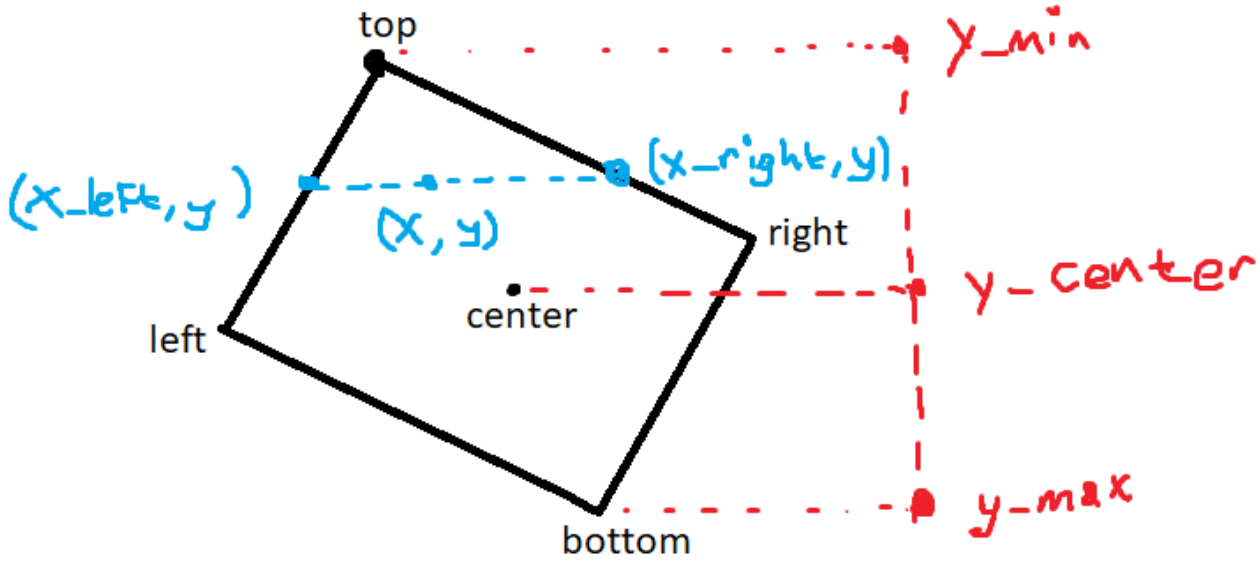
This formula works, since rotation is just multiplication with a unit vector having a different angle with x-axis.

- Then, we solve the line equations passing through right-top, top-left, left-bottom, bottom-right corners. We then save the coefficients of the equation with MATLAB's `polyfit` function (which solves the line equation passing through two points).

- Then, using the minimum and maximum y-coordinates of the rotated corners, for each y_min <= y <= y_max, we first determine whether y value is less than y-coordinate of the center. If so, we use the lines passing through left-top and top-right corners of the boundary to determine left-most and right-most x

coordinates we can traverse the descriptor. If y is greater than y-coordinate of the center, we repeat the same process for lines passing through left-bottom and bottom-right corners of the boundary. We record the boundaries we obtain in a matrix for future use.

We can visualize this process as follows:



This method gives the best solution to boundary finding problem, since we find the exact coordinate interval for points in the descriptor region for each y value. Compared to the common solution, which is to find a rectangular boundary whose edges are parallel to the axis, we do not evaluate whether a point in the traversal interval is in the descriptor region: Each point we obtain is guaranteed to be inside the descriptor.

- Then, we use the boundaries obtained previously to iterate the descriptor region. At each pixel, we obtain its RGB values using image indexing:

```
pixel = original_image(index_x, index_y, :);
pixel_red = pixel(1);
pixel_green = pixel(2);
pixel_blue = pixel(3);
```

Here, the input is the original image before processed to be used by VLFeat, and outputs are RGB values of the pixel.

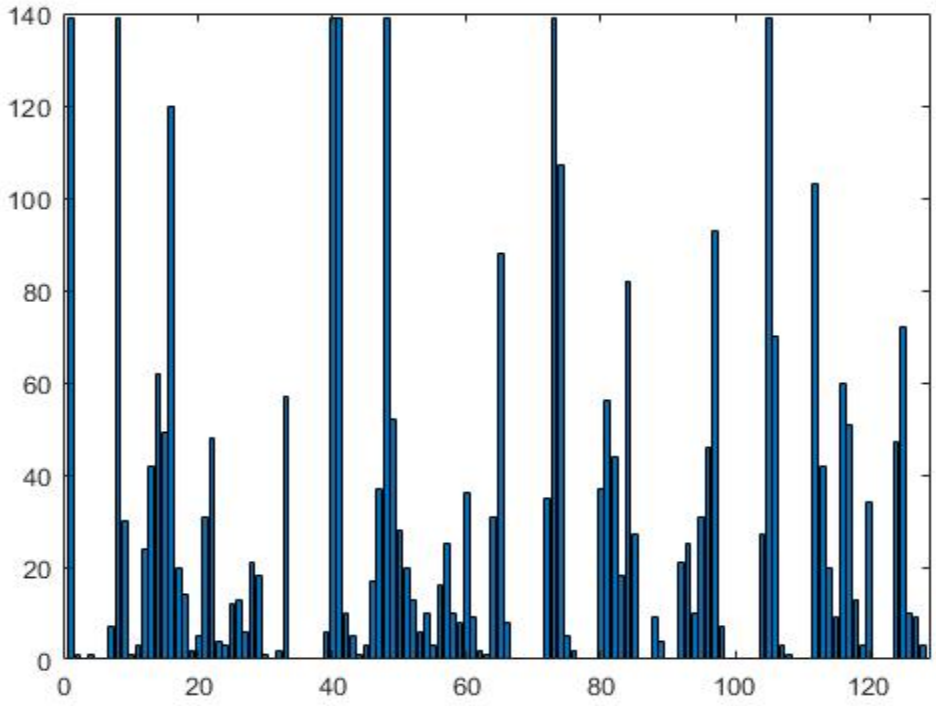
Then, we categorize color values into 4, each 64 intensity range is a new category. Using the categories, we increment size-64 vector's elements with respect to the following formula:

```
color_histograms(i, ((category_red - 1) * 16 + (category_green - 1) * 4 + category_blue)) =
color_histograms(i, ((category_red - 1) * 16 + (category_green - 1) * 4 + category_blue)) + 1;
```

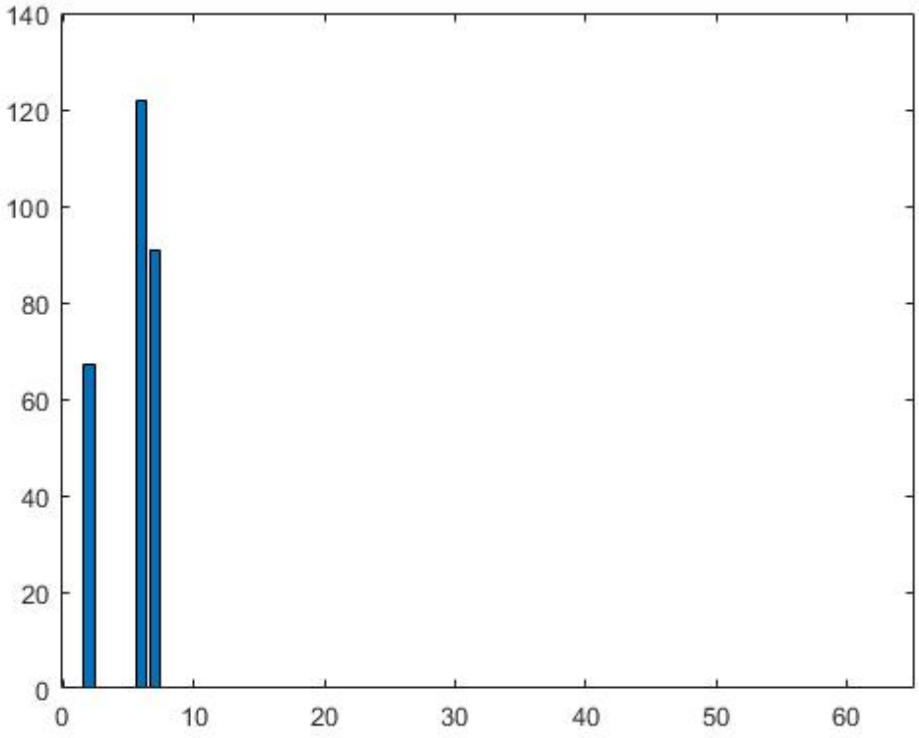
Here, color_histograms is the matrix storing size-64 color descriptors of each interest point. The index `((category_red - 1) * 16 + (category_green - 1) * 4 + category_blue)` is calculated by multiplying red category by 16, green category by 4, and blue category by 1. Reason we substract 1 from red and green category is that the maximum index we need to have is 64 and minimum one is 1. Without substraction, maximum index we'd get is 84 and minimum one is 21.

This concludes the description of obtaining descriptors. Now we present bar plots for gradient descriptors and color histograms.

Point 1
Gradient Descriptor

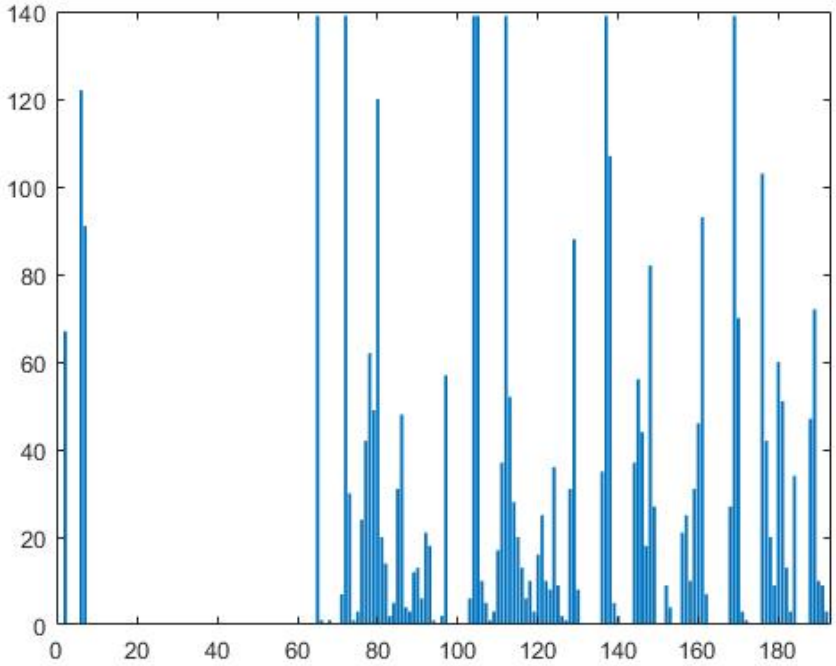


Color Descriptor

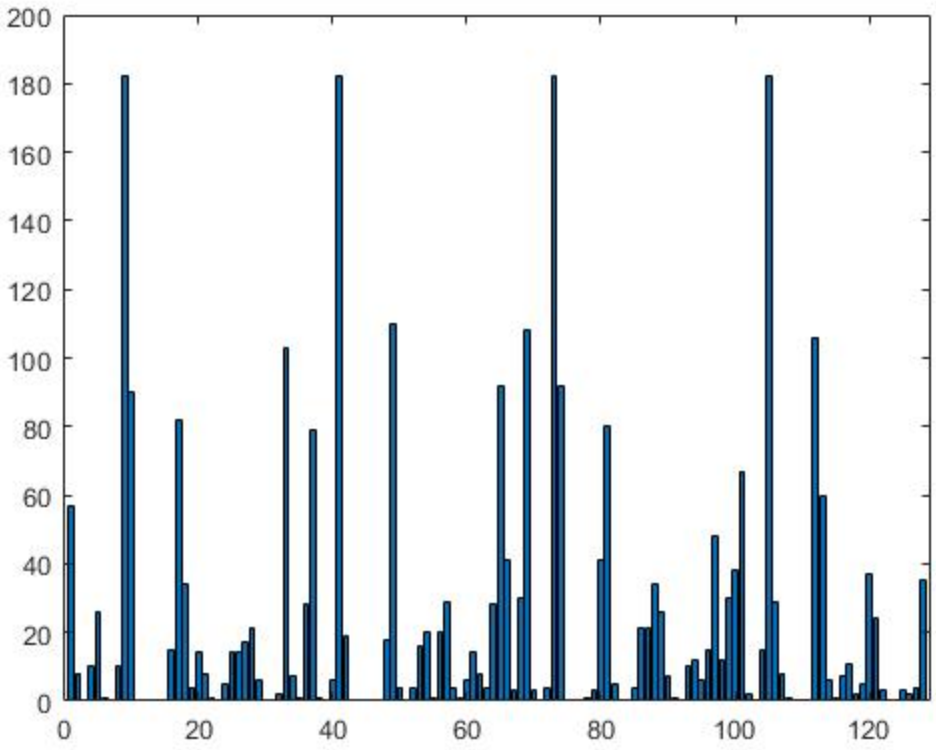


10

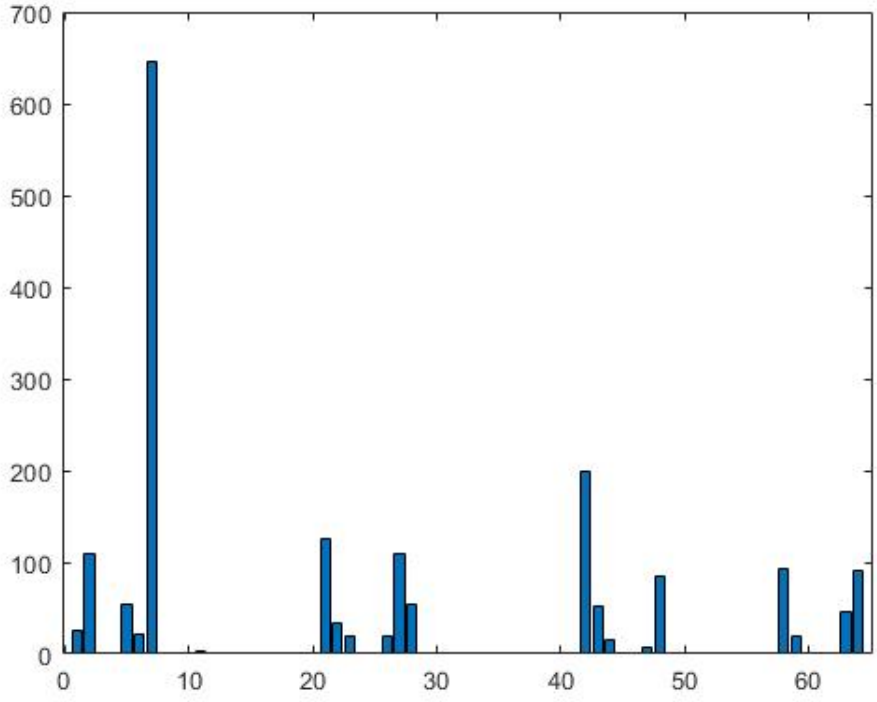
Gradient-Color Concatenated



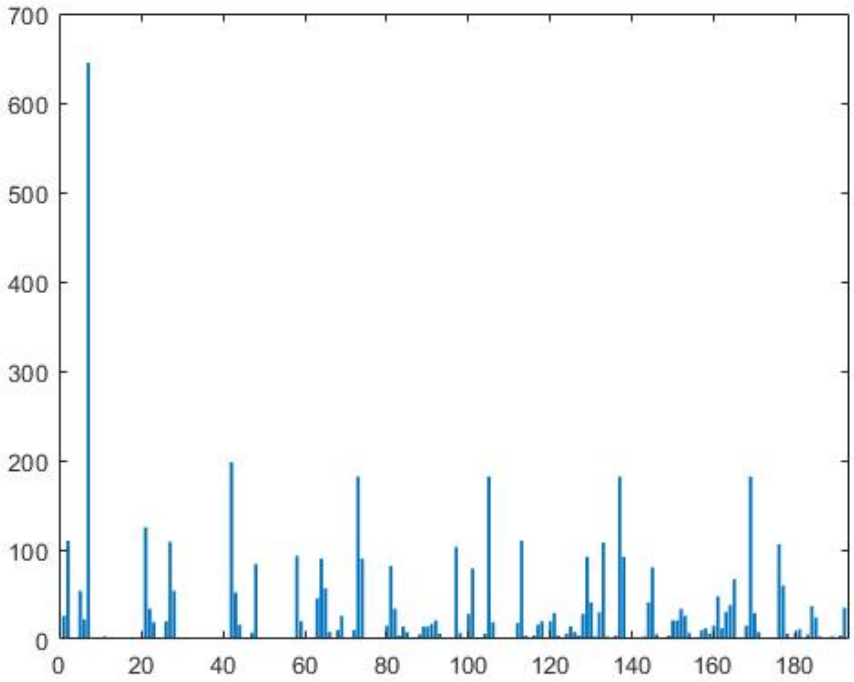
Point 2
Gradient Descriptor



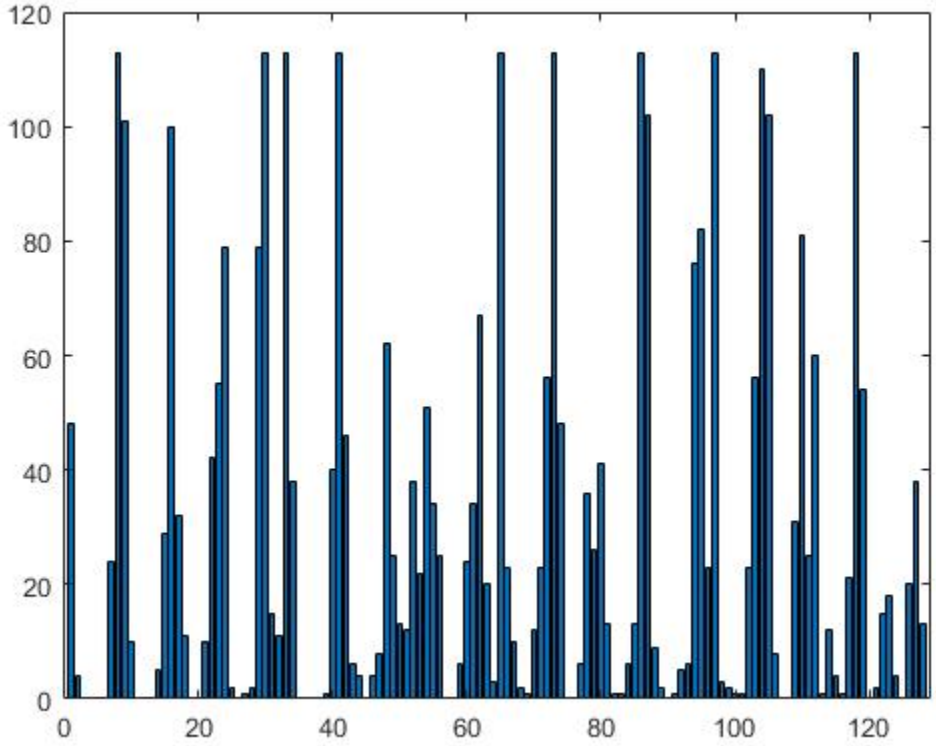
Color Histogram



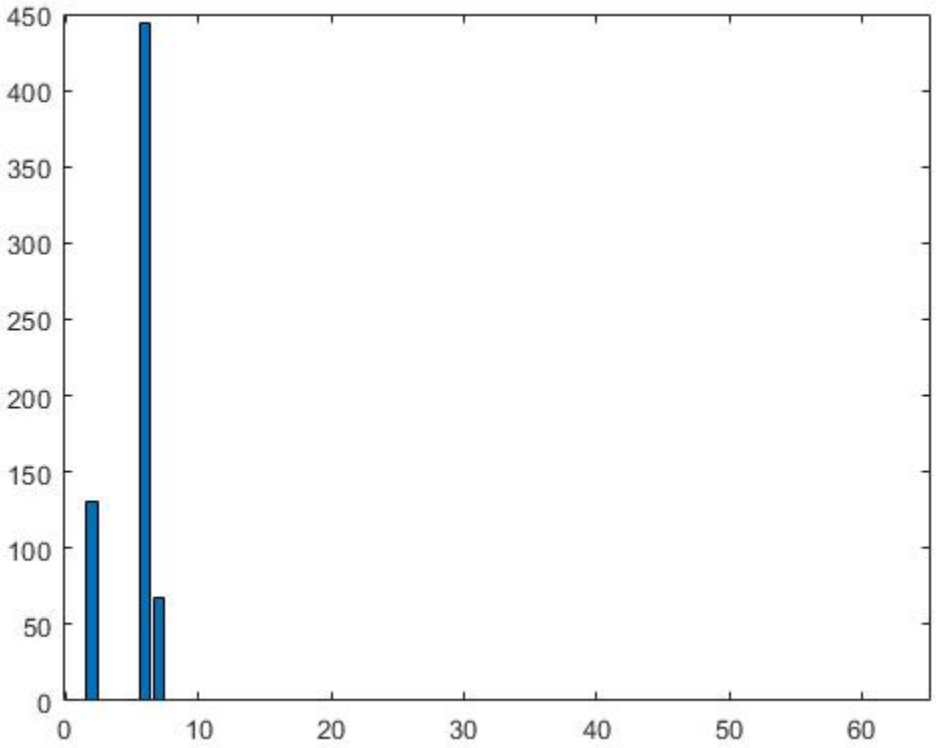
Gradient-Color Concatenated



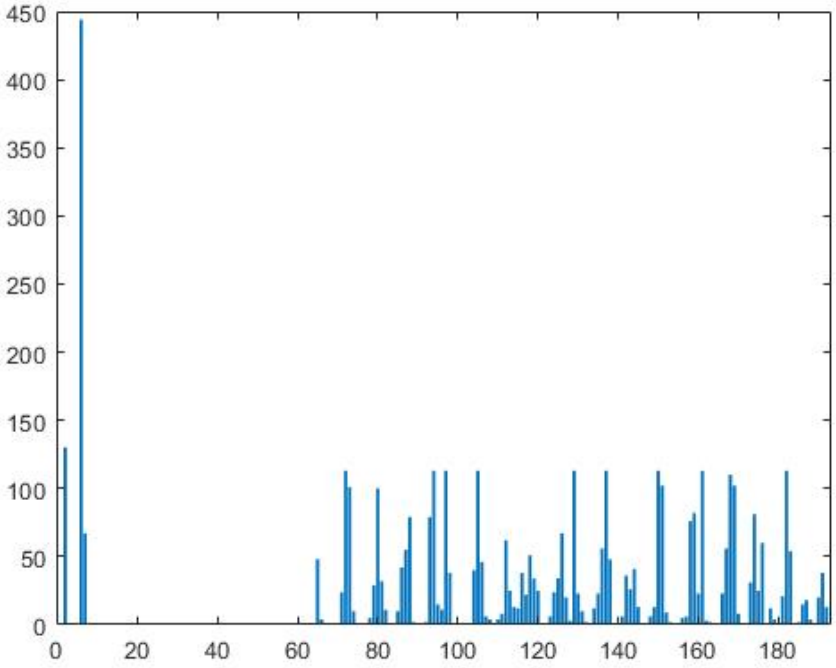
Point 3
Gradient Descriptor



Color Histogram



Gradient-Color Concatenated



3 Finding Visual Words (3.4 in Homework)

For this part, we use *vl_kmeans* function from VLFeat library. The function prototype is as follows: `[C, A] = vl_kmeans(X, NUMCENTERS)` where `X` contains the descriptors of local features and `NUMCENTERS` is the number of clusters the points are clustered into. Output `C` represents the center of clusters, and output `A` represents the assignment of data points in `X` to `NUMCENTERS` clusters. We use 2 cluster count, 500 and 1000, for each descriptor, gradient only, color only, and gradient-color concatenated. We utilize the function with extra arguments, namely 3 more: `'algorithm', 'ELKAN', 'maxnumcomparisons', 100, 'maxnumiterations', 100`. Argument `'algorithm'` defines the algorithm to use in k-means clustering. `'ELKAN'` is a faster alternative, so we decided to use it. `'maxnumcomparisons'` defines the maximum number of comparisons in an iteration when determining the cluster a data point belongs to. `'maxnumiterations'` defines maximum number of iterations before the algorithm reaches convergence. Limiting the last two allows the algorithm to be converge faster, however, the output might not be the optimum solution since termination might occur before optimum solution is reached. However, we are interested in a time-efficient solution, so we decided to limit the last two options.

4 Bag of Words Representation (3.5 in Homework)

For this part, we use the clustering output from previous part. The algorithm is as follows:

- First, for each image, we separate descriptors belonging to that image from `A`, which is an output of `vl_kmeans` that shows which data point is assigned to which cluster. Since we concatenated descriptors of images together vertically, we can separate them by indexing. We repeat this process for each cluster count & descriptor type.

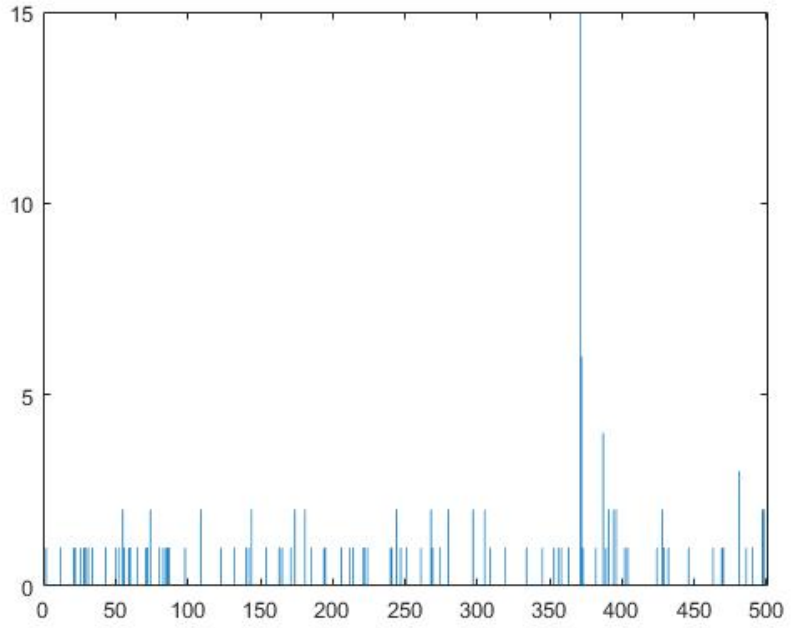
- Once we acquire cluster assignment & local features themselves, for each descriptor, we index to the bag-of-words representations; and using the cluster of the descriptor as an index for the bag-of-words representations, increment the cluster descriptor belongs to by one:

```
bag_of_words_gradient_500_A(i, descriptors_gradient_500_A(j)) =
bag_of_words_gradient_500_A(i, descriptors_gradient_500_A(j)) + 1;
        bag_of_words_gradient_1000_A(i, descriptors_gradient_1000_A) =
bag_of_words_gradient_1000_A(i, descriptors_gradient_1000_A(j)) + 1;
        bag_of_words_color_500_A(i, descriptors_color_500_A(j)) =
bag_of_words_color_500_A(i, descriptors_color_500_A(j)) + 1;
        bag_of_words_color_1000_A(i, descriptors_color_1000_A(j)) =
bag_of_words_color_1000_A(i, descriptors_color_1000_A(j)) + 1;
        bag_of_words_concat_500_A(i, descriptors_concat_500_A(j)) =
bag_of_words_concat_500_A(i, descriptors_concat_500_A(j)) + 1;
        bag_of_words_concat_1000_A(i, descriptors_concat_500_A(j)) =
bag_of_words_concat_1000_A(i, descriptors_concat_500_A(j)) + 1;
```

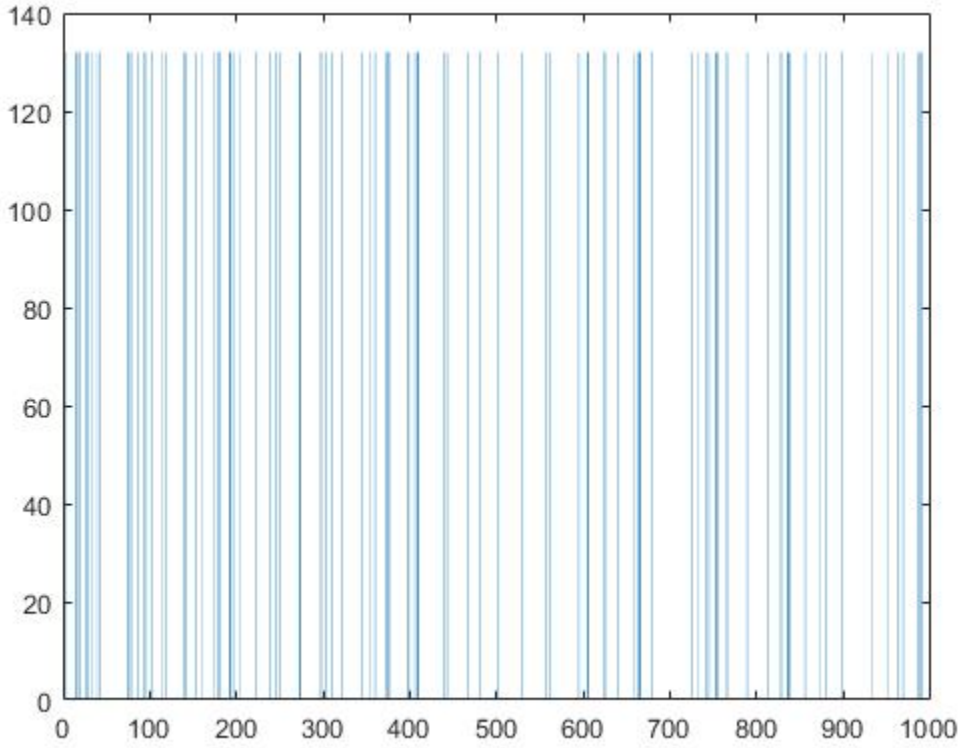
Here, variables starting with substring `bag_of_words_` are bag-of-words representations and variables starting with substring `descriptors_` are descriptors belonging to an image. This process is a simple counting algorithm using two indices: one index `descriptors_gradient_500_A(j)` which is jth descriptor of ith image; and another index `bag_of_words_gradient_500_A(i, descriptors_gradient_500_A(j))` belonging to bag-of-words representations.

We conclude our discussion on bag-of-words representations. Now, we present bar plots for some bag-of-words representations.

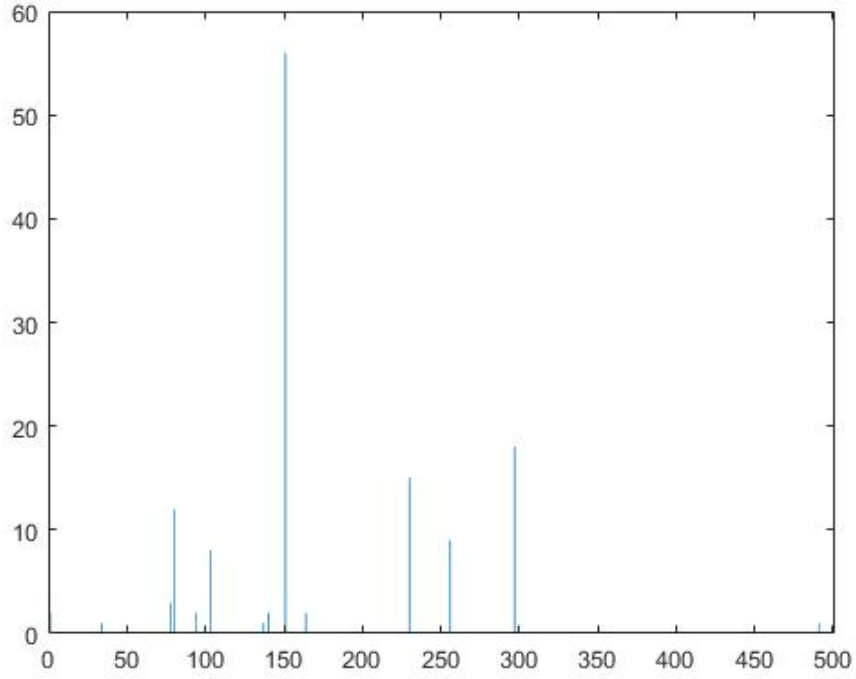
Point 1
Gradient Based, 500 Clusters



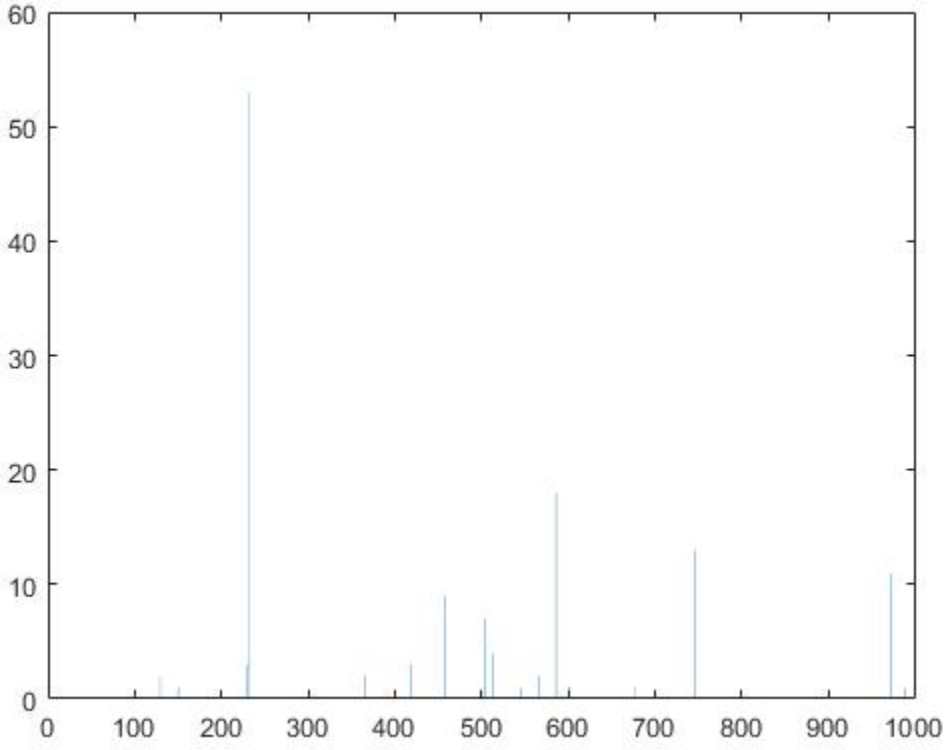
Gradient Based, 1000 Clusters



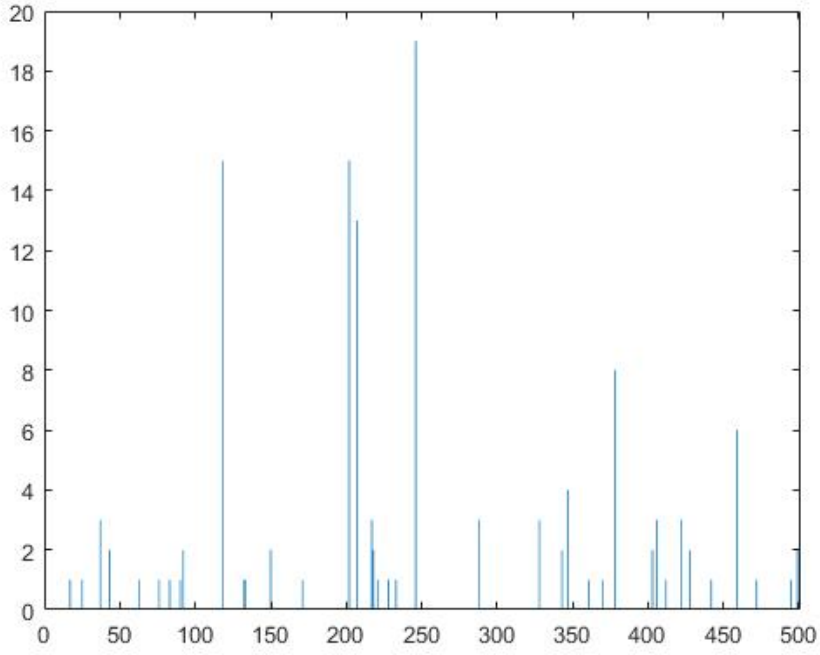
Color Based, 500 Clusters



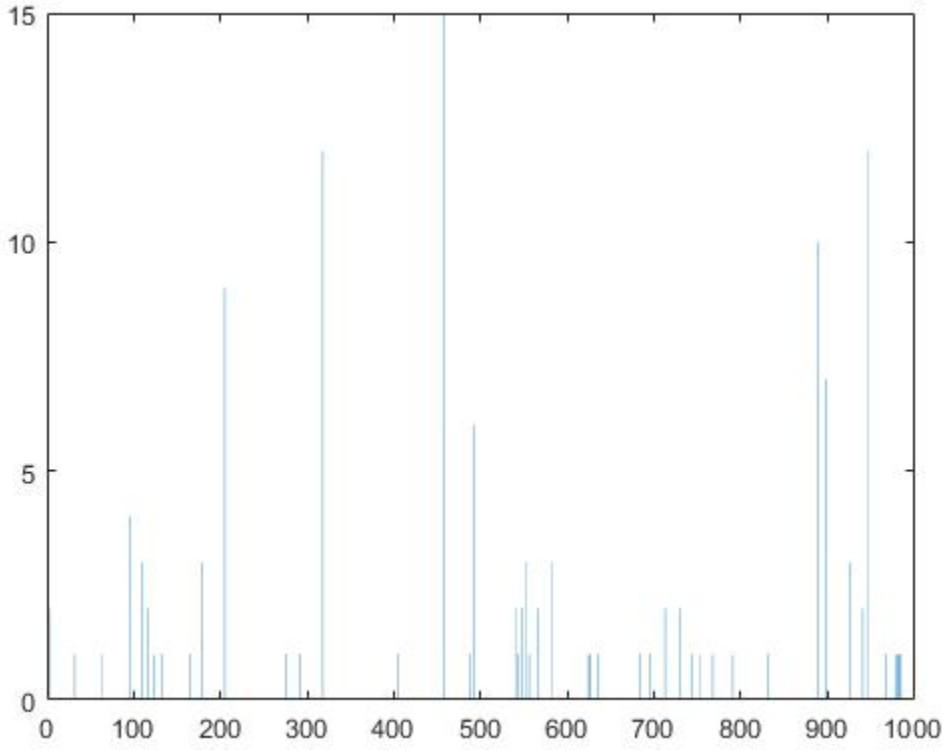
Color Based, 1000 Clusters



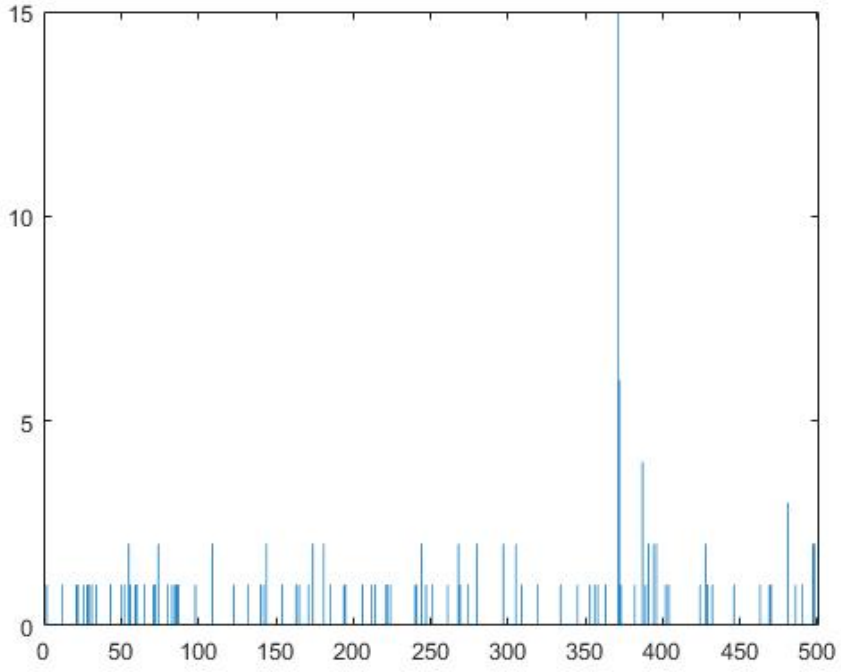
Gradient-Color Concatenated, 500 Clusters



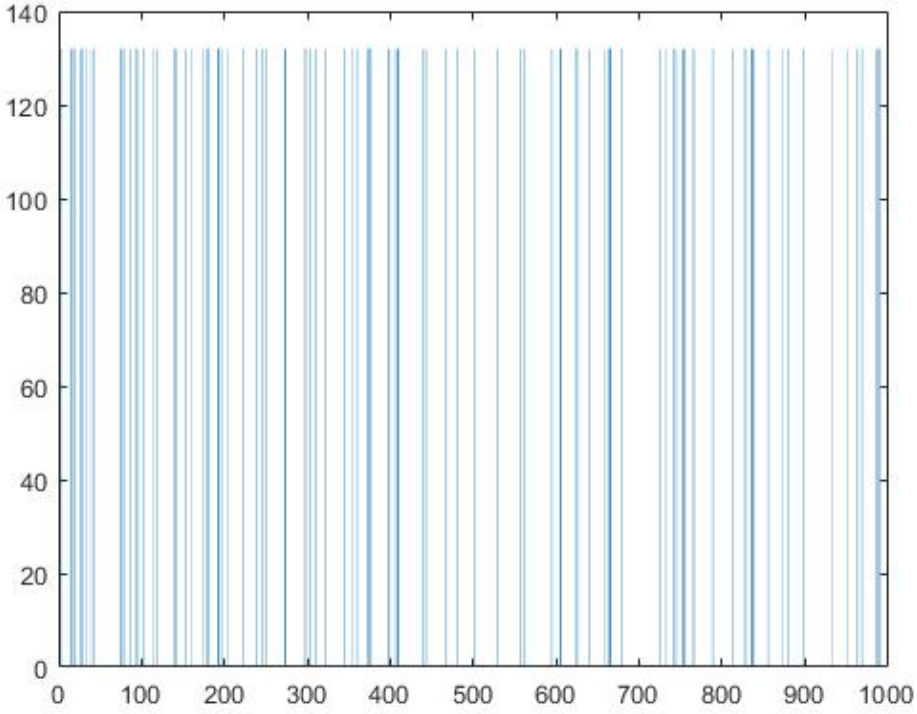
Gradient-Color Concatenated, 1000 Clusters



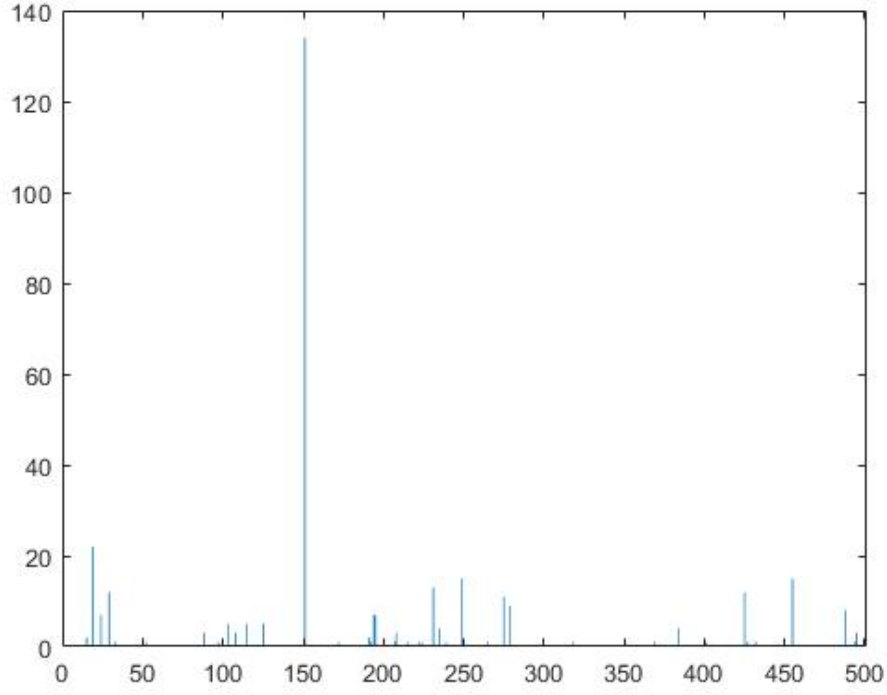
Point 2
Gradient Based, 500 Clusters



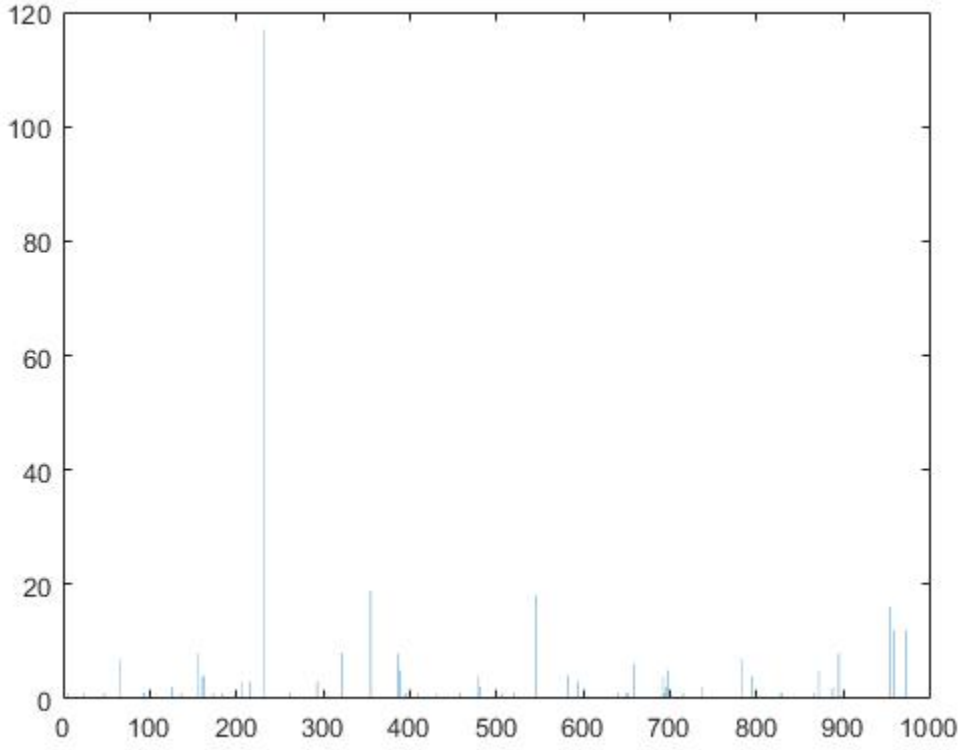
Gradient Based, 1000 Clusters



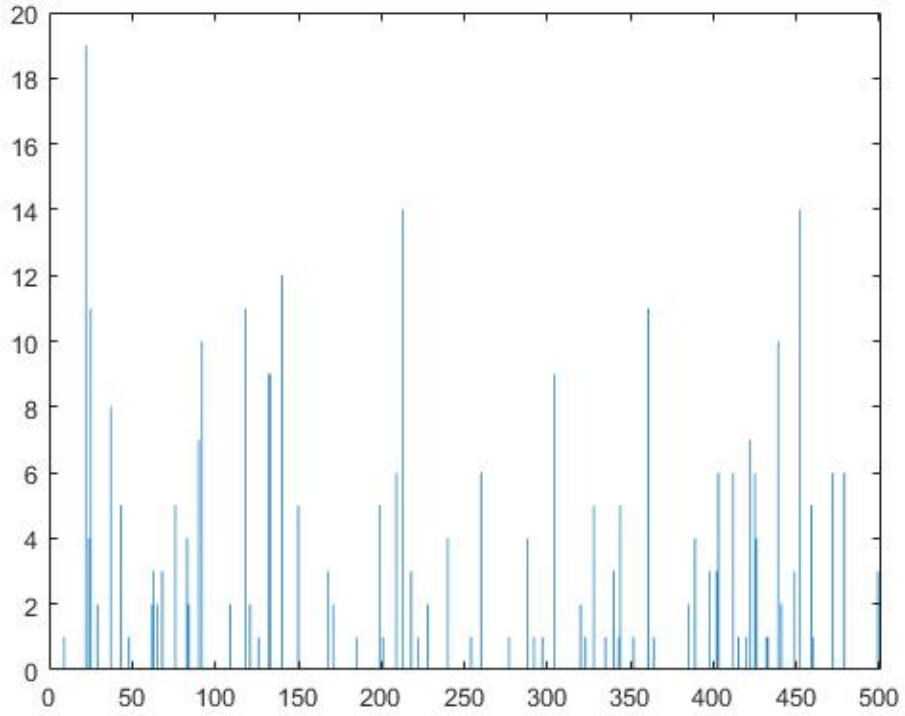
Color Based, 500 Clusters



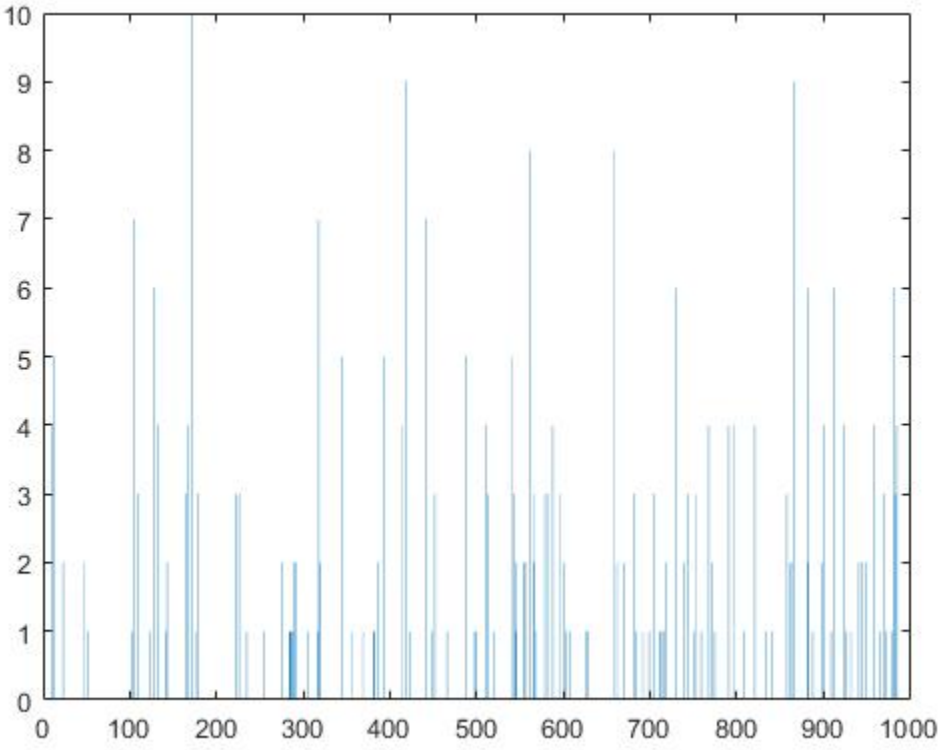
Color Based, 1000 Clusters



Gradient-Color Concatenated, 500 Clusters



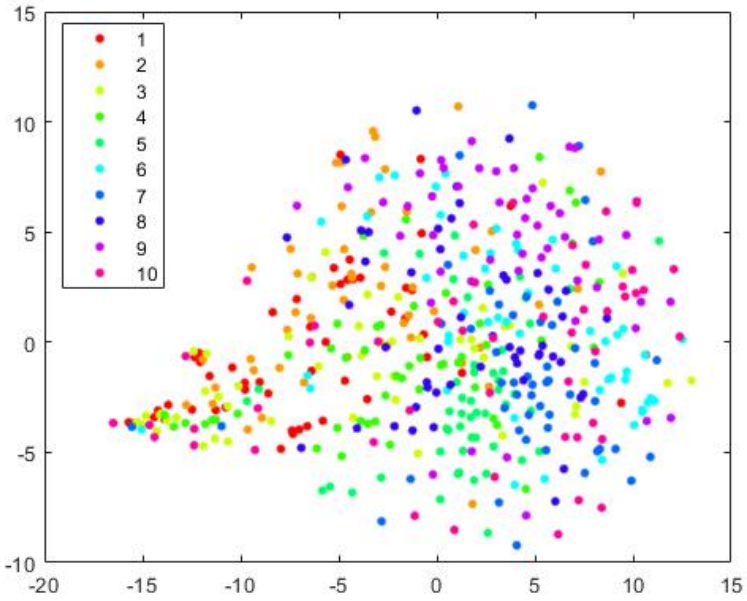
Gradient-Color Concatenated, 1000 Clusters



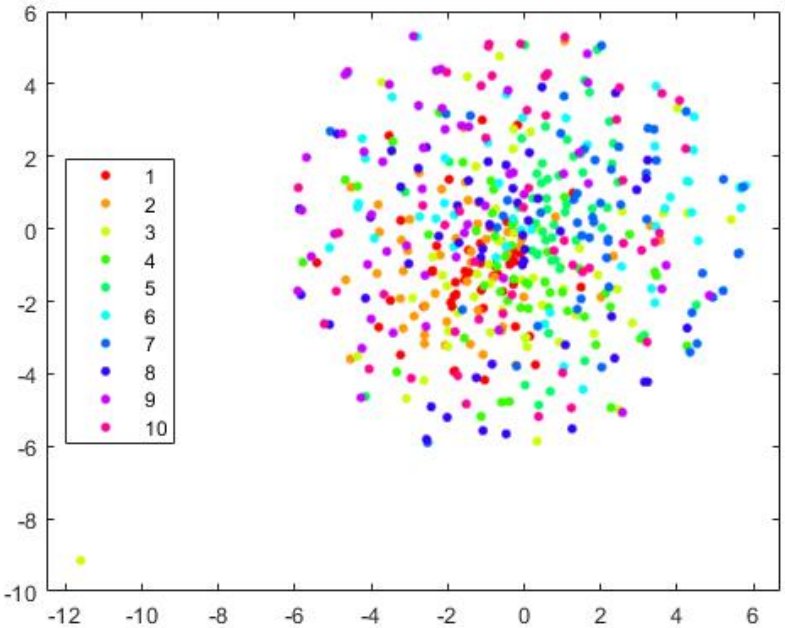
5 t-SNE Visualization (3.6 in Homework)

Using built-in MATLAB function `tsne`, we flatten the bag-of-words representation for each image into a point in 2D space. The function prototype is as follows: `Y = tsne(X)` where rows of `X` contains high-dimensional data points and `Y` contains the x and y coordinates of flattened points. We apply this function for each bag-of-words representation & plot the results using built-in MATLAB function `gscatter`. Prototype of gscatter is as follows: `gscatter(x, y, label)` where `x` and `y` are x and y coordinates in 2D plane, and `label` is the label of data points. We now present the final outputs:

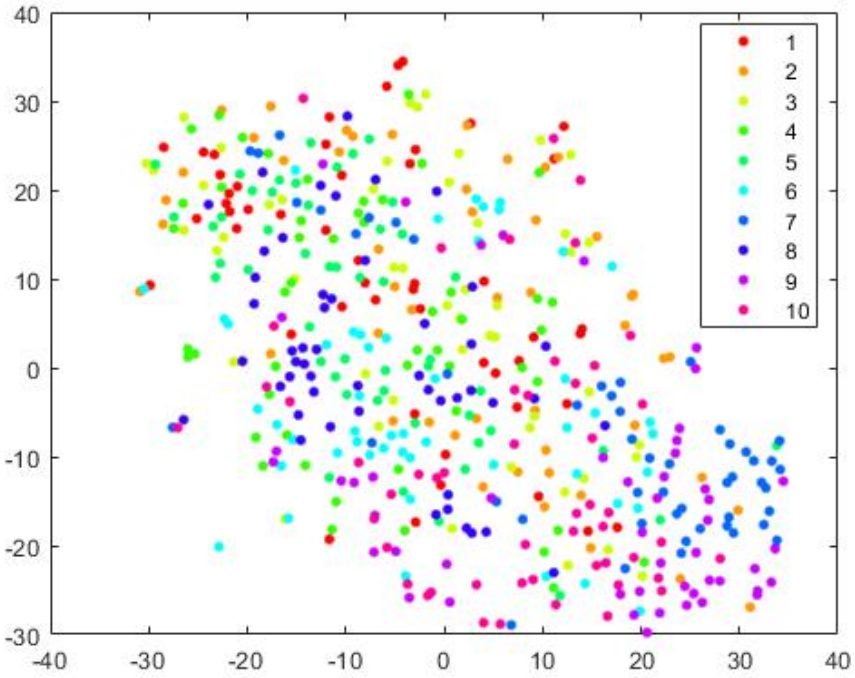
Gradient Based, 500 Clusters



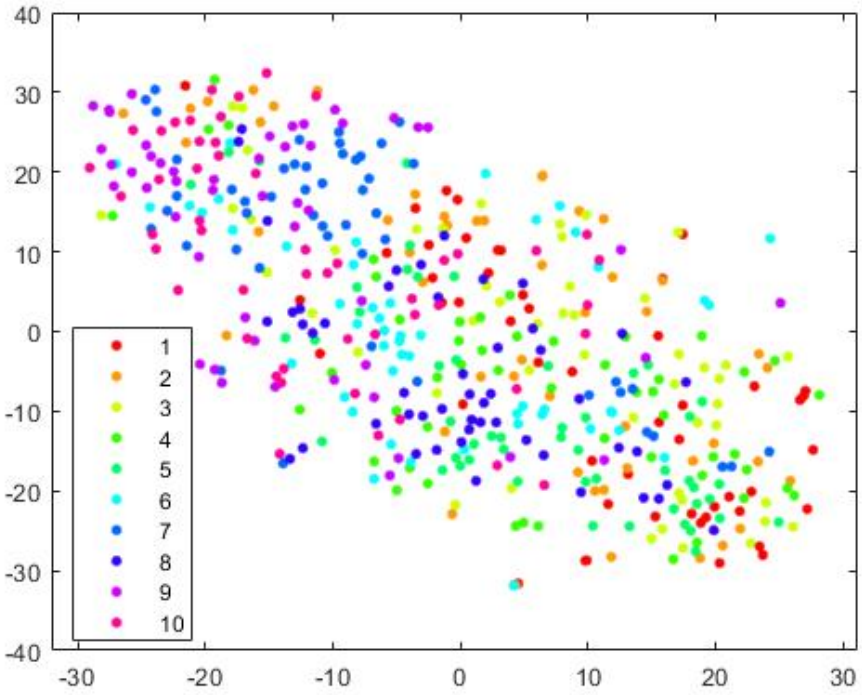
Gradient Based, 1000 Clusters



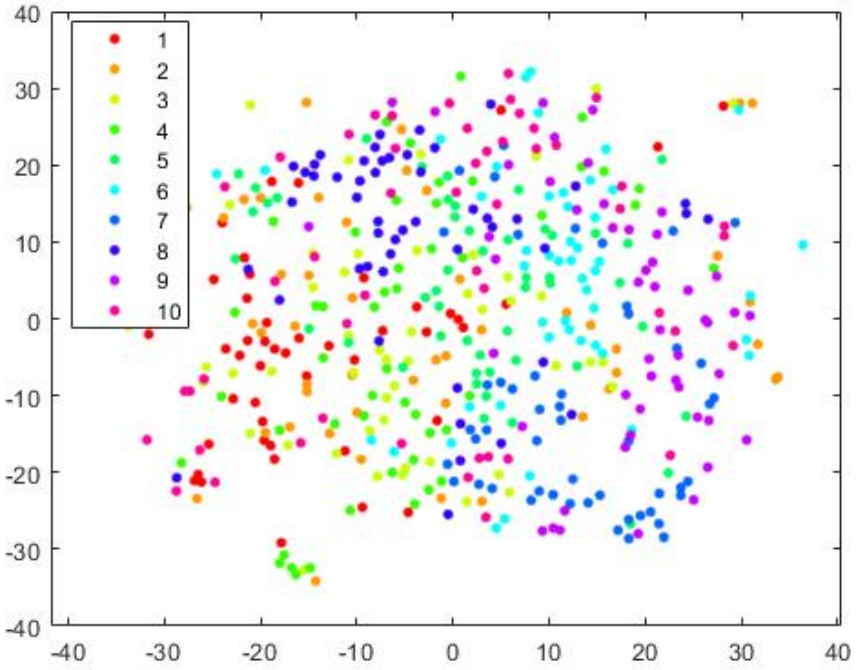
Color Based, 500 Clusters



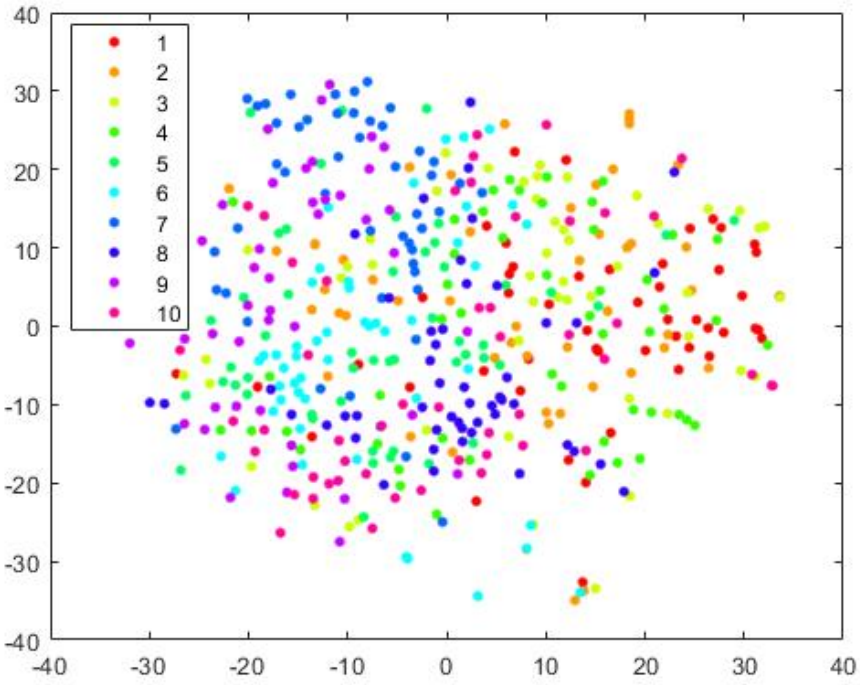
Color Based, 1000 Clusters



Gradient-Color Concatenated, 500 Clusters



Gradient-Color Concatenated, 1000 Clusters



6 Discussion

The most challenging obstacle we encountered was to design an algorithm that efficiently calculates traversal interval for a given descriptor. Since some descriptors includes regions that are outside the image, and the rotated corners does not allow us to use their coordinates directly, we had to develop a new algorithm ourselves. We believe the outcome of this endevour is rewarding: we are able to find the exact boundaries in the most efficient way possible. Compared to the previous version in which we find a bounding rectangle whose edges are parallel to the x and y axis, which checks points outside the actual descriptor boundary, we observed a significant improvement in completion time specifically when the descriptor size is significant compared to the image size.

A strategy we utilized during the calculation of descriptors was to downscale images larger than a threshold, in this case 1000x1000 pixels, to a lower resolution, 300x300 pixels. Our purpose here is to reduce time spent on some inputs having a large resolution, around 1 million pixels per image. The reason why such inputs are problematic is as follows: When calculating color descriptors, we traverse every pixel in the descriptor boundary to calculate a color histogram. As the number of pixels in this boundary increases, time spent on this calculation increase as well. In some cases, this value is as high as 8 minutes per image on an Intel Core i7-6700 CPU. A downside of this approach is that we loose a significant amount of local features since we simply remove the pixels from the image rather than reducing its size. However, the reward outweighs the compromise in our opinion.

Another problematic part is visualizing bag-of-words representation using gradient-based descriptors with 1000 clusters. If we look at the bag-of-words representations, we see that this specific representation is different than the others in terms of distribution and weights of samples, which tend to be unusually regularly distributed. Since bag-of-words representation using gradient-based descriptors with 500 clusters' output for the same data points seem to be non-uniformly distributed, we are unable to explain why such a representation occurred. One possibility might be the cluster count, 1000, is a unique value for the number of descriptors in the dataset so each cluster either gets a specific number of feature points or none.

In discussion of t-SNE outputs, we observe that as the number of clusters increase, distribution of data points seem to be more compact &| regular. For example, compared to Color Based 500, Color Based 1000 is more flat in $y = -x$ direction. Similarly, Gradient Based 1000 forms an accurate circle around data points whereas in Gradient Based 500, we observe some points outside this circle. In Gradient Color Concatenated 1000, data points outside the central circle are closer to it compared to Gradient Color Concatenated 500.

Compared to Gradient Based 500, classes in Gradient Based 1000 are more compact. For example, class 9 has points all around the circle in Gradient Based 500 whereas in Gradient Based 1000 they are grouped in left-half of the circle. Similarly, class 6 is more visible in Gradient Color Concatenated 1000 compared to Gradient Color Concatenated 500 and points are closer to each other. In Color Based 1000, we see that samples in class 4 occur more frequently in the right part of the $y = -x$ direction whereas in Color Based 500 these points are distributed almost evenly in the elliptic structure surrounding all data points.

However, it is difficult to clearly distinguish every class by looking at either t-SNE output. This is so partly because of the number of classes in the input. If we were to show less number of clusters in the

graphs, the difference between clusters would be more clear. However it is logical to assume that, since the number descriptors are large, a greater number of cluster centers in k-means & t-SNE output is a better choice compared to a lesser one. Although there is no obvious choice here, it is logical to assume that using Gradient-Color Concatenated 1000 will give the best results, since we take into account not only the gradient-based descriptors but also the color-based ones as well; which results in more data to work on and naturally a better performance.