Mean Shift Image Segmentation

January 31, 2016

1. Overview

Image segmentation provides us with an easier way to analyse an image by *clustering* similar parts of image together so the they may be analysed as a single unit. The result is a set of segments or contours which significantly differentiate different areas of the image. We will be implementing the Mean Shift Image Segmentation Method.

Mean Shift Clustering is a non-parametric kernel density estimation technique, that clusters points based on repeatedly updating the mean. The means will dictate how the image will be segmented and the number of such segments.

The algorithm for this method in general involves the following steps.

- Apply a kernel on a data point over neighbours within a sufficient bandwidth distance, with the chosen point as the mean of the kernel distribution.
- Compute a new mean after applying the kernel
- Repeat till convergence.
- Join nearest means
- Assign a label to every pixel equivalent to label of nearest mean

There exists a rigorous proof for convergence of the mean by this method, we will not be covering it here. But this procedure *as it is*, is very heavy computationally. So we employ different methods to speed up the segmentation process. We have used *Tensorflow*, an open source software library for numerical computation using data flow graphs, which has very efficient implementations of matrix computations.

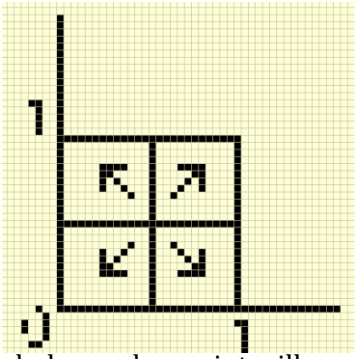
2. Optimizations

We have also tweaked the algorithm with various heuristics to bring down the complexity.

2.1. Binned Estimator

Traditionally we are to apply the kernel on every point and find the convergence point. Although it is very accurate to do so, we will not be doing it on every point. Instead we'll be choosing some initial seeds, which will represent the neighbourhood! And will approximately converge to final means. The seeds are chosen the following way

- Divide all the coordinates of all the pixels of the image by the bandwidth.
- Round off the coordinates to the nearest whole number.
- Create bins indexed as whole numbers.
- Compute frequency of each bin



- Ignore all bins below a threshold frequency
- The whole number points, multiplied by bandwidth, will be the seeds.

The Mean Shift procedure will be initialised only at these points.
The motivation behind this binning can be visualised here -->

Every point in the grid will also converge to same point as the nearest whole number point will converge (approximately!)

2.2. Radial Nearest Neighbours

To apply on the points within a distance of bandwidth (δ), we need to find all the points that are closer than a distance of δ.

Note: Distance is computed in 5 dimensional hyperspace of the image.

$$D = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (R_i - R_j)^2 + (G_i - G_j)^2 + (B_i - B_j)^2}$$

2.2.1. A Brute Force Approach

Compute distance to all pixels and pick all those pixels for which $D < \delta$. As can be clearly this involves redundant computations.

One might suggest to keep a distance matrix computed in pre-processing so that D_{ij} might be directly accessed. But this is memory inefficient as the matrix in question is really large.

2.2.2. A Better Approach

Apply the brute force method only on some selected pixels. We will choose such pixels by looking at their physical 2 dimensional coordinates

Let $\Phi_i = \{ pixel_j \mid x_j - x_i < \delta \ \ and \ \ y_j - y_i < \delta \}$. If $D \notin \Phi_i$ then definitely $D > \delta$ as $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < \delta$.

$\therefore$ It is enough to search for radial neighbours only in Φ_i.

2.2.3. A much better approach

Though the running time was greatly reduced by using the second approach, we searched for a better implementation and found one being used in *scikit-learn* libraries.

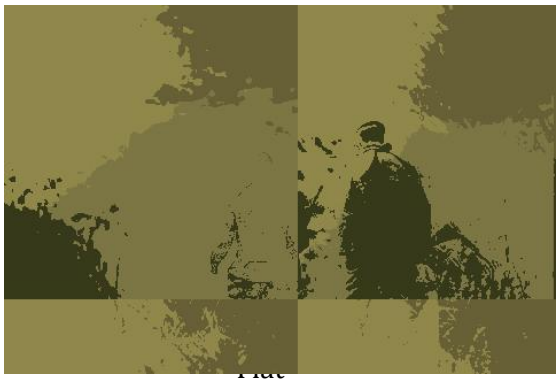
The implementation in study uses efficient data structures like Ball trees and KD trees to decrease memory requirements of storing distance, compromising on time to a reasonable extent.

We did not implement any tree structures but utilised the rich library of *scikit-learn* for computing radial nearest neighbours. The query times were logarithmic and the running time reduced significantly.

3. Results

The results reported here are computed using the second method of radial neighbour computation and binned estimator.

The choice is not very apparent in all images but it does matter in some of them.



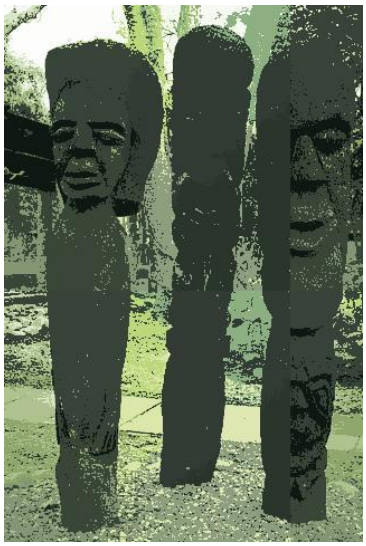
Flat

Gaussian

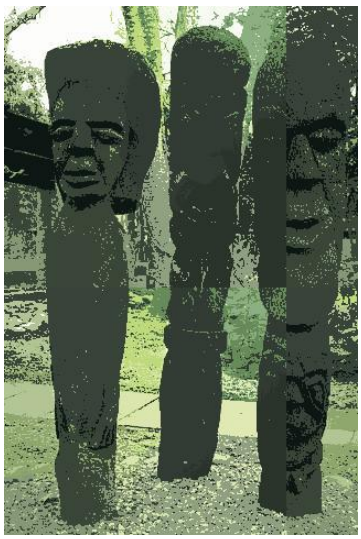
Both images are segmented using $\delta = 10$, the guassian seems to outperform

Whereas in some cases there isn't any difference

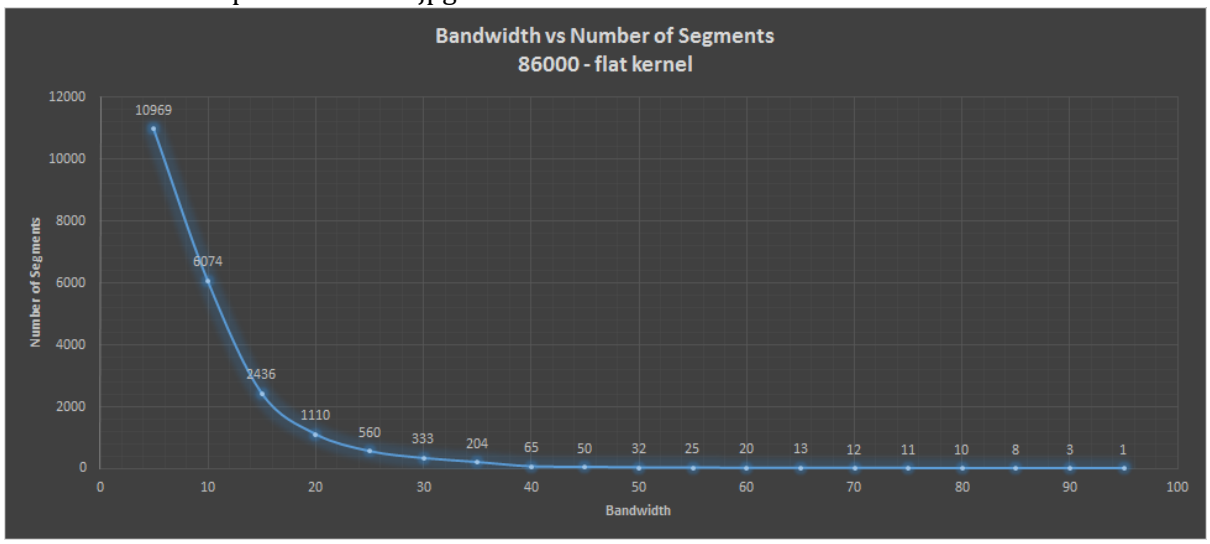
Flat ($\delta = 5, 40, 60$)

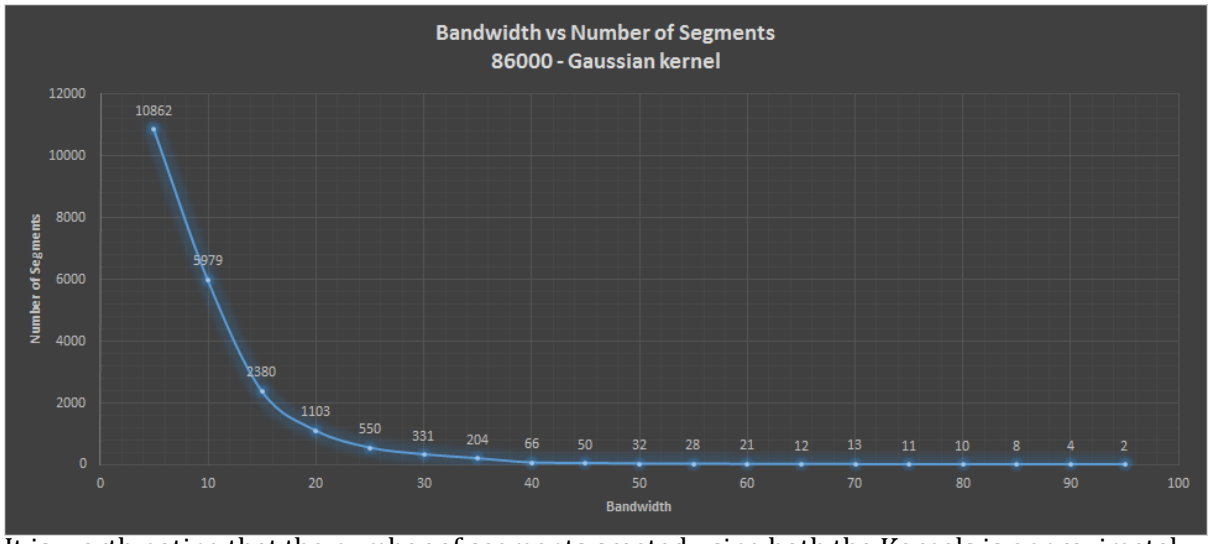


Gaussian ($\delta = 5{,}40{,}80$)

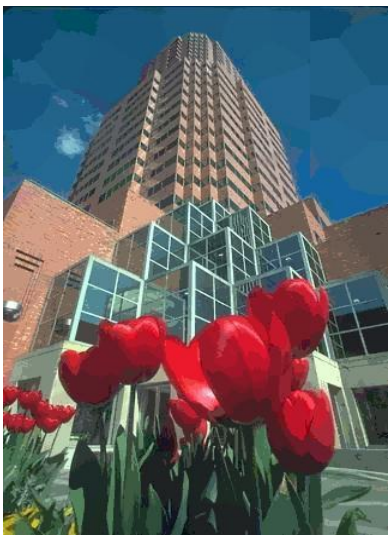


In general, the number of segments generated falls exponentially with increase in bandwidth as is observed in the plot for 86000.jpg





It is worth noting that the number of segments created using both the Kernels is approximately the same, in this case too. Also the *knee point doesn't differ with the choice of Kernel.*



Gaussian vs Flat

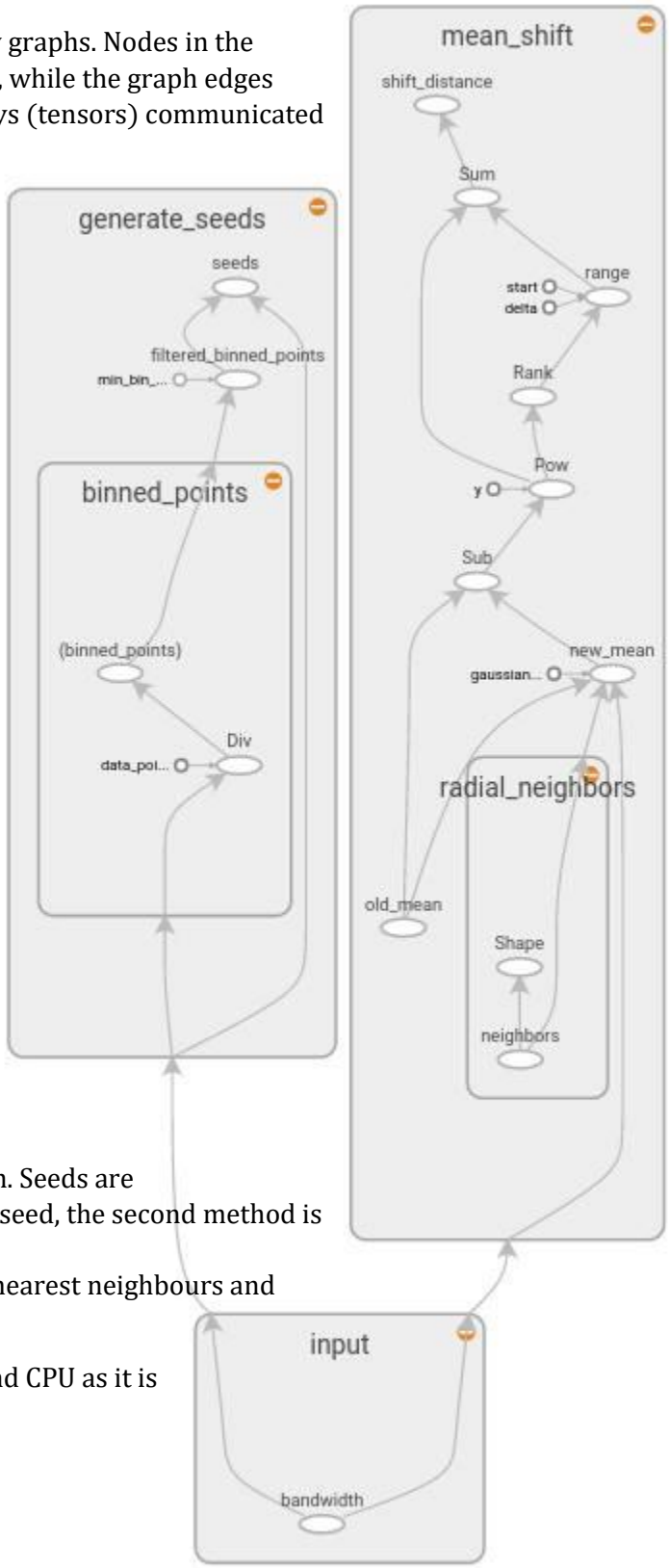
$\delta = 20$ (near Knee)

86000.jpg

The above algorithm involves matrix computations, which are efficiently implemented in Tensorflow.

All computations are done using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.

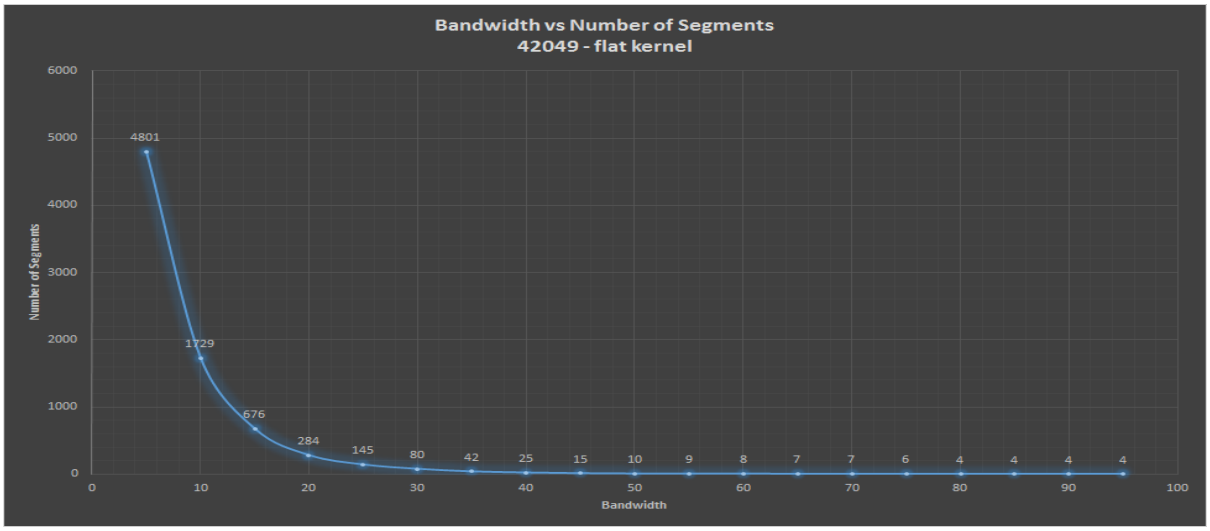
This is the data flow graph generated by our method.

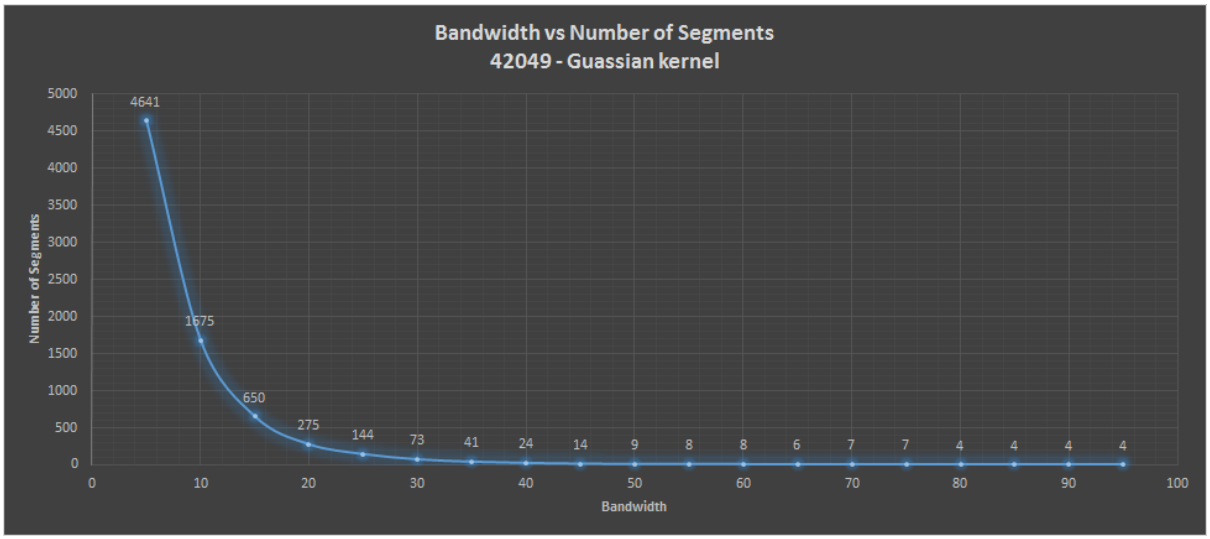


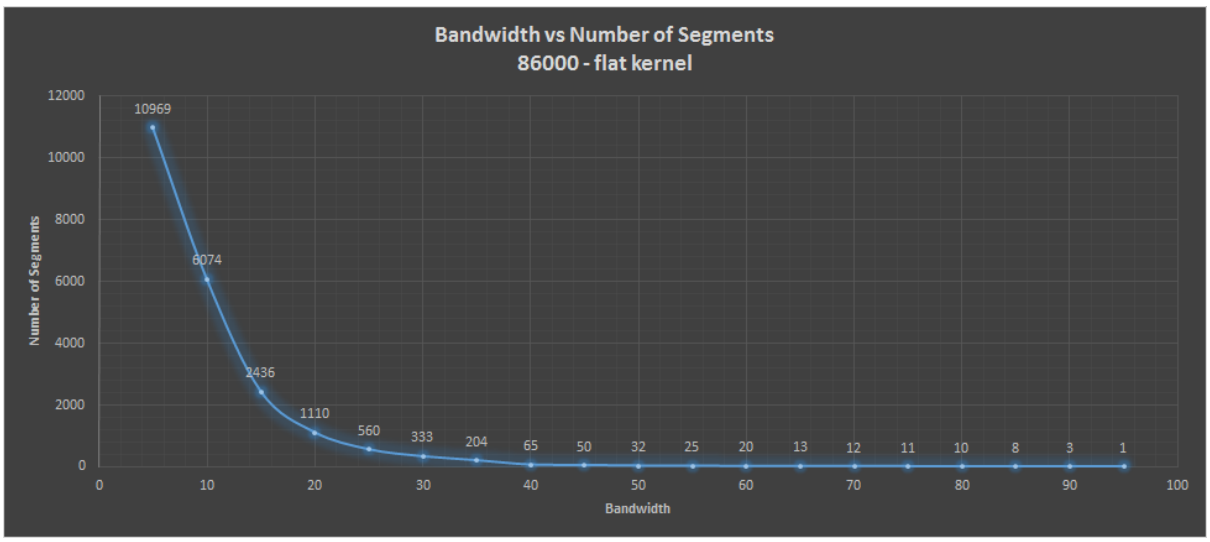
Every node is some function we've written. Seeds are generated by the first graph and on every seed, the second method is applied to find converging means.
Note the subgraphs for generating radial nearest neighbours and binned points.

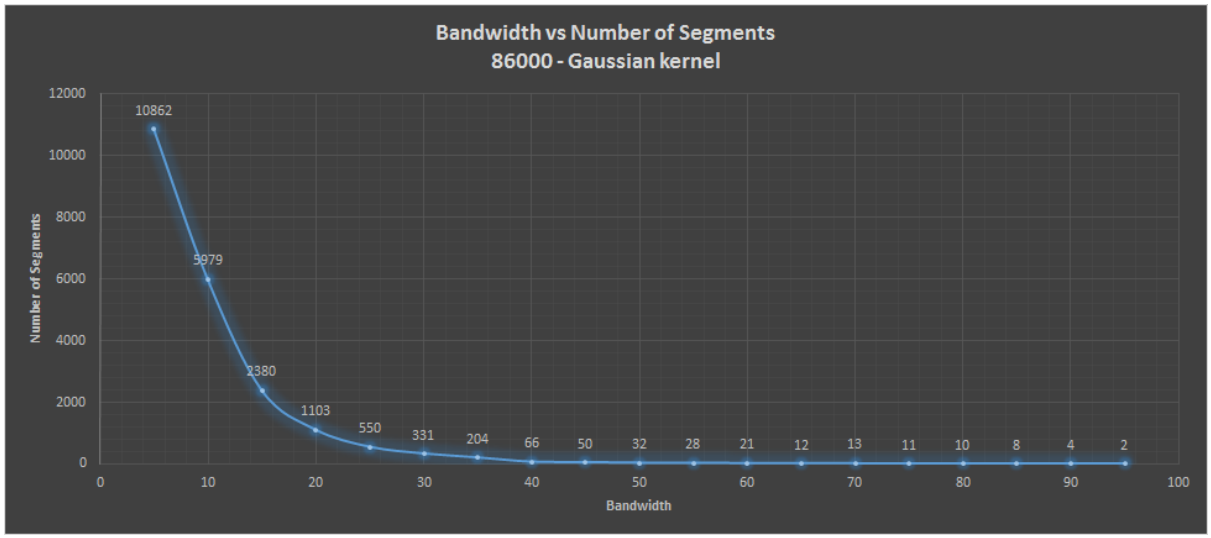
By doing so we can easily between GPU and CPU as it is already internalised in Tensorflow.

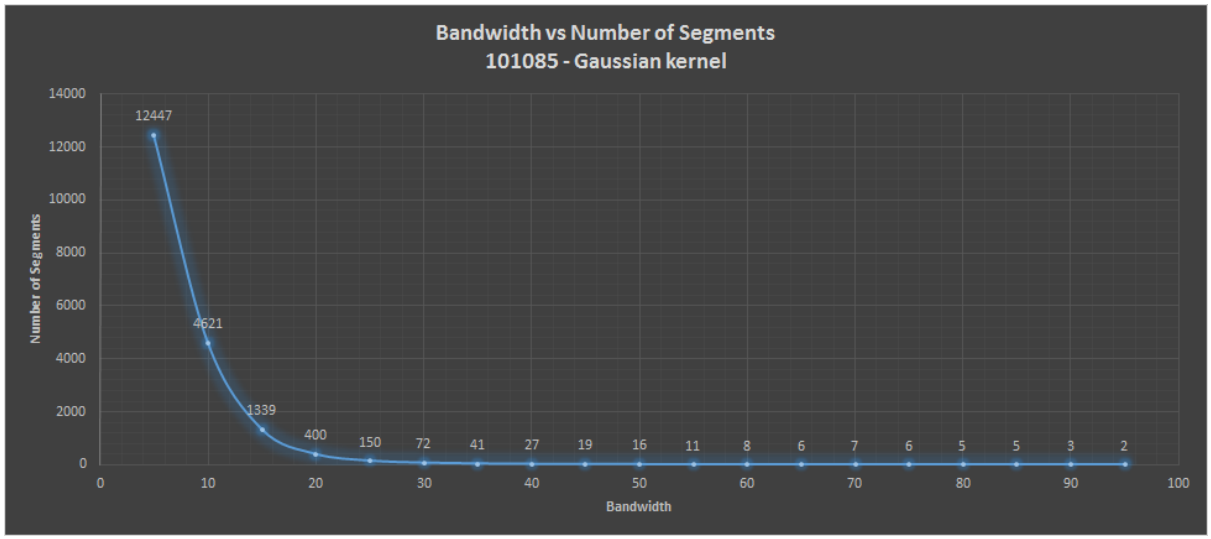
B. All Plots

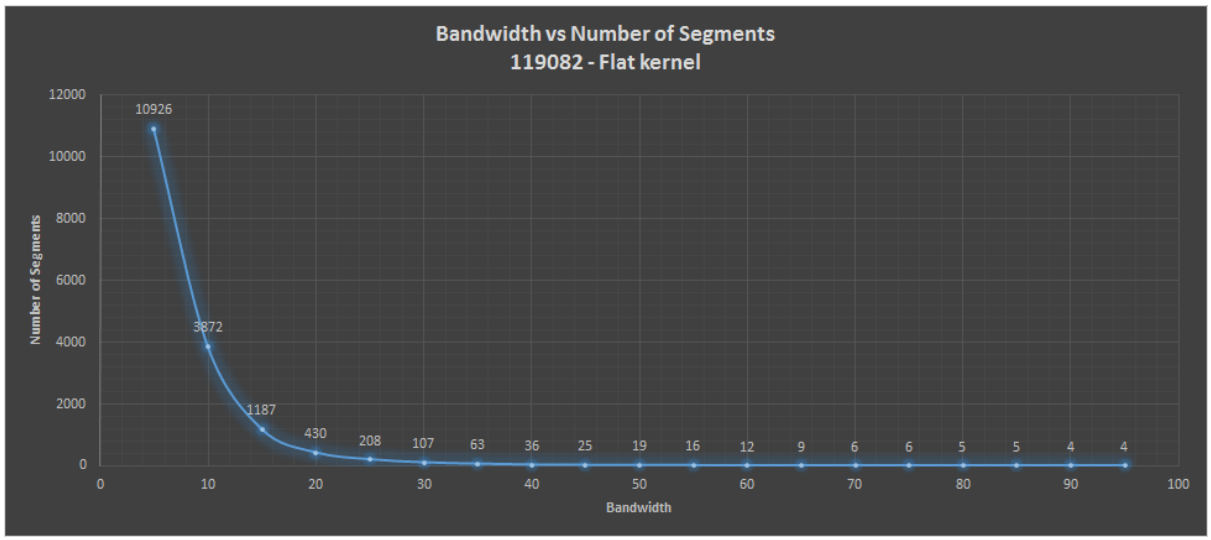

Bandwidth vs Number of Segments
42049 - flat kernel
Number of Segments
Bandwidth
4801
1729
676
284
145
80
42
25
15
10
9
8
7
7
6
4
4
4
4

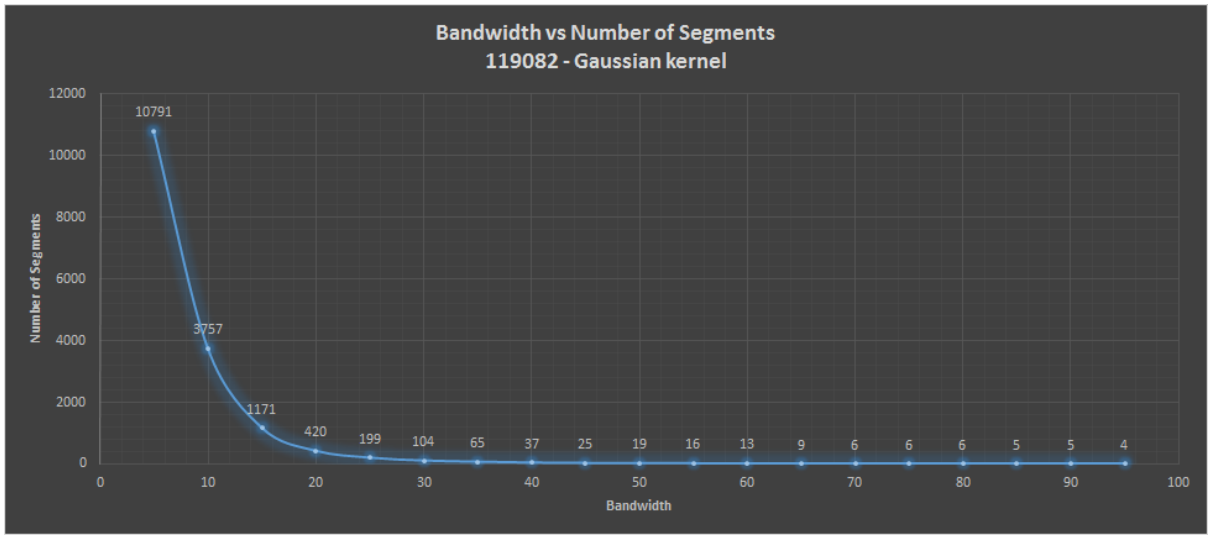

Bandwidth vs Number of Segments
42049 - Guassian kernel
Number of Segments
Bandwidth
4641
1675
650
275
144
73
41
24
14
9
8
8
6
7
7
4
4
4
4

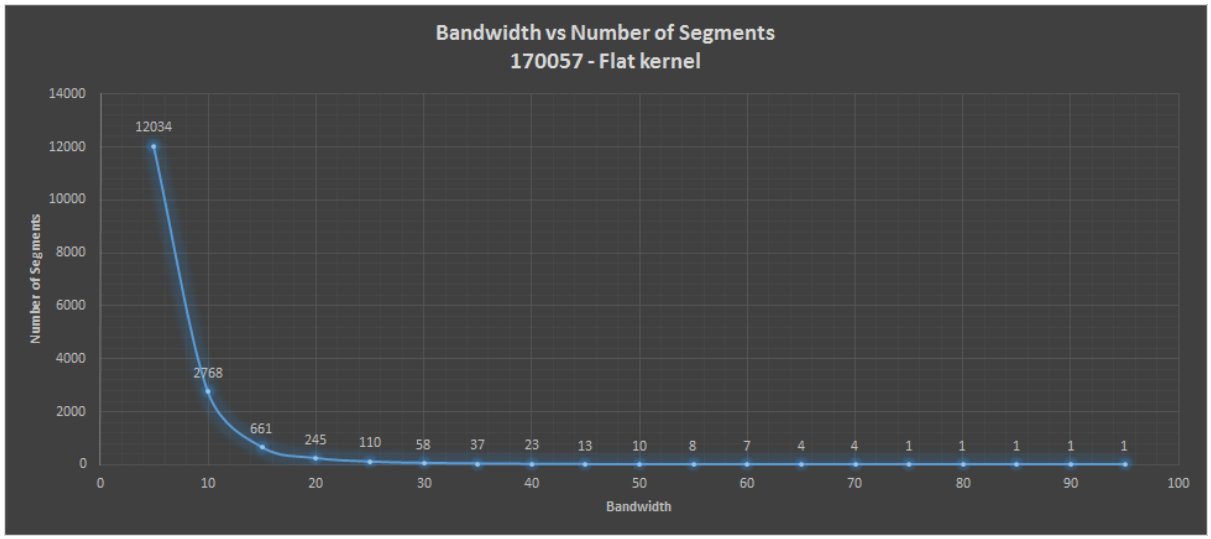

Bandwidth vs Number of Segments
86000 - flat kernel
Number of Segments
Bandwidth
10969
6074
2436
1110
560
333
204
65
50
32
25
20
13
12
11
10
8
3
1

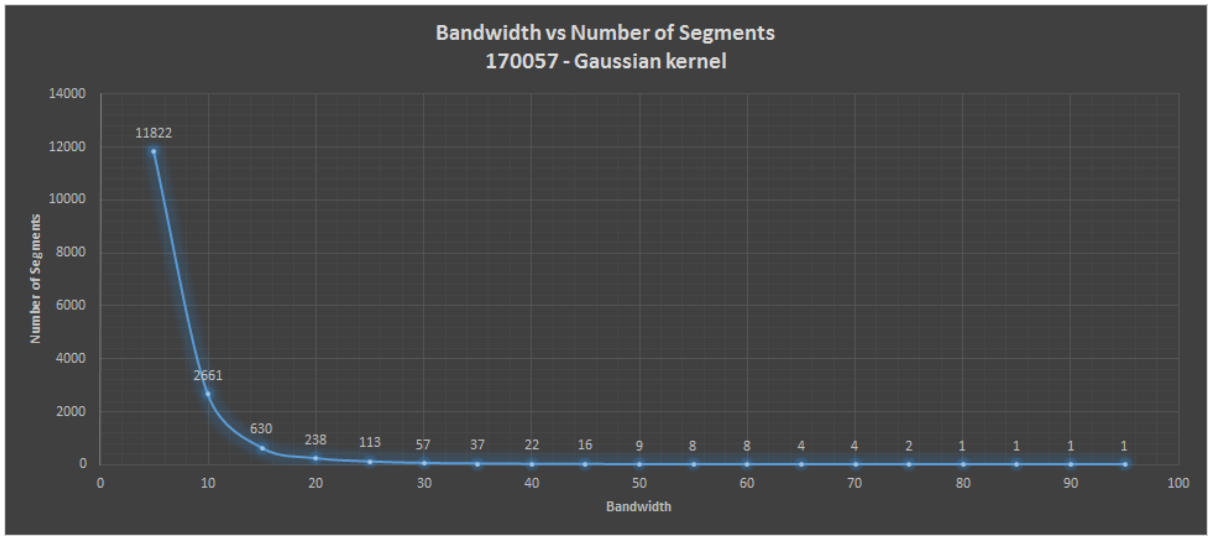

Bandwidth vs Number of Segments
86000 - Gaussian kernel
Number of Segments
Bandwidth
10862
5979
2380
1103
550
331
204
66
50
32
28
21
12
13
11
10
8
4
2


Bandwidth vs Number of Segments
101085 - Gaussian kernel
Number of Segments
Bandwidth
12447
4621
1339
400
150
72
41
27
19
16
11
8
6
7
6
5
5
3
2


Bandwidth vs Number of Segments
119082 - Flat kernel
Number of Segments
Bandwidth
10926
3872
1187
430
208
107
63
36
25
19
16
12
9
6
6
5
5
4
4


Bandwidth vs Number of Segments
119082 - Gaussian kernel
Number of Segments
Bandwidth
10791
3757
1171
420
199
104
65
37
25
19
16
13
9
6
6
6
5
5
4


Bandwidth vs Number of Segments
170057 - Flat kernel
Number of Segments
Bandwidth
12034
2768
661
245
110
58
37
23
13
10
8
7
4
4
1
1
1
1
1


Bandwidth vs Number of Segments
170057 - Gaussian kernel
Number of Segments
Bandwidth
11822
2661
630
238
113
57
37
22
16
9
8
8
4
4
2
1
1
1
1

Links

- Code Repository : https://github.com/manikantareddyd/TF_Mean_Shift_Image_Segmentation

- Mean Shift: A Robust Approach Towards Feature Space Analysis
- Tensorflow
- Nearest Neighbors, Scikits-learn