Chinmay Chinara
USC-ID: 2527237452
E-mail: chinara@usc.edu

EE-569: HOMEWORK #3

Issue date: 02-26-2017 Due date: 03-26-2017

Table of Contents

Chinmay Chinara
USC-ID: 2527237452
E-mail: chinara@usc.edu

Problem-1: Texture Analysis and Segmentation

a. Texture Classification

I. Abstract and Motivation

Texture of an image defines the complete realm of what an image might look like. It is the heart of the image and the one which defines it. These textures are the ones that define how an image would look like. The intricate details they provide helps in classifying the image as well as segmenting the different components present in them. The spatial arrangement of the various intensity values is what the texture provides.

Task: There are twelve texture images as training data, denoted by Texture1-Texture12, which needs to be taken as input. They need to be categorized into four different parts (bark, grass, straw and sand) with three images of each types using two methods:

i. K-means clustering (unsupervised learning)
ii. Visual inspection (supervised learning)

The second part of the task is to use the above methods incorporated on the training data and use it to classify six other texture images, TextureA-TextureF, into the four classes bark, sand, grass and straw and finally answer the given set of questions given as a conclusion.

II. Approach and Procedures

Texture classification is a special type of machine learning problem that needs the application of the very famous K-means clustering to cluster the feature points by generating a codebook and extract neat features out of it. This method is called the unsupervised learning. The generic algorithm followed for K-means clustering is as under:

Step-1: Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
Step-2: Assign each object to the group that has the closest centroid.
Step-3: When all objects have been assigned, recalculate the positions of the K centroids.
Step-4: Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Theoretical Approach:

The idea to start the classification method is to extract features from texture image. One effective way of doing it is to filter the input texture images with a set of filters (called a filter bank) and compute the local energy from their output image. This s known as multichannel feature extraction method. As per the data given to us, a filter bank in our case will consist of 25 Laws' filters of size 5x5 using the tensor product of 1D kernels as shown in the table given below:

NAME	KERNEL
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5 (Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

To start off, we first generate a 5x5 Laws' filter from the 5 1D filters provided to us. Each of the 1D filter is combined with another filter by getting a tensor product to produce a 5x5 filter. These 5x5 filters are then applied to the given texture image with a certain number of steps as shown below:

1. **PRE-PROCESSING:** The input images are read and DC components is subtracted from each of them. The reason to do so is because we want to eliminate any high feature values that might come in the way of capturing good

feature from the image. The DC components have high energy values but very less information content. This is one critical step before feature extraction. The way we do is based on the formula below:

$$Img_{DC_{removed}} = Input_{Img} - (Mean\ of\ all\ pixels\ in\ each\ of\ the\ input\ images) \quad \rightarrow (1)$$

2. **FEATURE EXTRACTION:** The 25 Laws' filters were created as per the methods mentioned above and were applied (convolved) with each of the Texture1-Texture12 images. So now each of the 12 images had 25 images of their own which represented the 25D feature vector for each training image. Thus, each of the training image gave 25 output images that represented the filter response of the entire filter bank in terms of the training data. The above outputs of 12x25 were then normalized to find the energy of feature vector. The formula used was as under:

$$Feature_{Vector_{normalized}} = \frac{Feature_{Vector} - (Mean\ of\ all\ the\ data\ in\ the\ feature\ vector)}{Standard\ deviation\ of\ the\ feature\ vector} \quad \rightarrow (2)$$

   The normalized feature vector obtained gives a much better classification output as it always eliminates high values. This 12x25 normalized data is then used for performing classification / clustering.

3. **PCA FOR DIMENTIONALITY REDUCTION:** The Principal component analysis or PCA is used to reduce the dimensionality of the feature vectors obtained above. All the 25D feature vectors obtained above may not be important for classification purpose. The higher the dimensions, more is the time and space complexity. Dimensionality reduction is a process of deriving a set of features that reproduce most of the variability in a given dataset. PCA reduces the feature space by calculating the Eigen values and Eigen vectors of higher dimensional data and reduce it to n dimensions whose basis are the Eigen vectors corresponding to n largest Eigen values. Here n = 3. MATLAB's PCA function was used to reduce the 25D feature vector to 3D feature vector which could be represented in 3D space.

4. **CLUSTERING USING K-MEANS:** To classify and unknown set of textures using the texture types bark, grass, straw and sand, the known textures were first clustered into K points or centroids. So, the feature space is now known to consist of K centroids / clusters/ points. This was then passed into the K-means clustering algorithm as described above to get the best possible cluster centroids based on an iterative process till the Euclidean distance between that of Centroids and Data is the minimum and there is no change on the Centroids even on performing any further iterations.

Algorithm used to develop in C++:

Steps for classification of image without PCA (unsupervised):

Step-1: Read all the 12 images and store them in 2D arrays.
Step-2: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a 12x25 feature vector matrix.
Step-6: Normalize the 12x25 matrix obtained above using eqn-2 above.
Step-7: Perform the K-means clustering over the 12x25 data by randomly selecting the initial K centroids until several iterations till there is no more change in the value of centroids. This process was the unsupervised learning of classifying the data because the initial guess was random.

Step-8: Based on the Euclidean distance obtained for every centroid value, the min value's index indicates the class and type of texture it is.
Step-9: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.
Step-10: Step1-6 was followed for the 6 images that needs classification.
Step-11: Now classify the 6 images using the training data obtained above.
Step-12: The Euclidean distance of the centroids above with each normalized data from the 6 images was calculated.
Step-13: The min value's index indicates the class the texture is in.
Step-14: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.

Steps for classification of image with PCA (unsupervised):

Step-1: Read all the 12 images and store them in 2D arrays.
Step-2: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a 12x25 feature vector matrix.
Step-6: Normalize the 12x25 matrix obtained above using eqn-2 above.
Step-7: Save the 12x25 data in a .txt file and use MATLAB to perform PCA to get the 12x3 data. Save the 12x3 data to file again and load it back into your program by saving it in a 2D array.
Step-8: Perform the K-means clustering over the 12x3 data by randomly selecting the initial K centroids until several iterations till there is no more change in the value of centroids. This process was the unsupervised learning of classifying the data because the initial guess was random.
Step-9: Based on the Euclidean distance obtained for every centroid value, the min value's index indicates the class and type of texture it is.
Step-10: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.
Step-11: Step1-6 was followed for the 6 images that needs classification.
Step-12: Save the 6x25 data in a .txt file and use MATLAB to perform PCA to get the 6x3 data. Save the 6x3 data to file again and load it back into your program by saving it in a 2D array.
Step-13: Now classify the 6 images using the training data obtained above.
Step-14: The Euclidean distance of the centroids above with each normalized data from the 6 images was calculated.
Step-15: The min value's index indicates the class the texture is in.
Step-16: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.

Steps for classification of image without PCA (supervised):

Step-1: Read all the 12 images and store them in 2D arrays.
Step-2: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a 12x25 feature vector matrix.

Step-6: Normalize the 12x25 matrix obtained above using eqn-2 above.
Step-7: Perform the supervised clustering over the 12x25 data by assigning the initial K centroids with mean of 3 classes each across the 25 dimensions. This process was the supervised learning of classifying the data because the initial guess was random.
Step-8: Step1-6 was followed for the 6 images that needs classification.
Step-9: Now classify the 6 images using the training data obtained above.
Step-10: The Euclidean distance of the centroids above with each normalized data from the 6 images was calculated.
Step-11: The min value's index indicates the class the texture is in.
Step-12: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.

Steps for classification of image with PCA (supervised):

Step-1: Read all the 12 images and store them in 2D arrays.
Step-2: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a 12x25 feature vector matrix.
Step-6: Normalize the 12x25 matrix obtained above using eqn-2 above.
Step-7: Save the 12x25 data in a .txt file and use MATLAB to perform PCA to get the 12x3 data. Save the 12x3 data to file again and load it back into your program by saving it in a 2D array.
Step-8: Perform the supervised clustering over the 12x3 data by assigning the initial K centroids with mean of 3 classes each across the 3 dimensions. This process was the supervised learning of classifying the data because the initial guess was random.
Step-9: Step1-6 was followed for the 6 images that needs classification.
Step-10: Save the 6x25 data in a .txt file and use MATLAB to perform PCA to get the 6x3 data. Save the 6x3 data to file again and load it back into your program by saving it in a 2D array.
Step-11: Now classify the 6 images using the training data obtained above.
Step-12: The Euclidean distance of the centroids above with each normalized data from the 6 images was calculated.
Step-13: The min value's index indicates the class the texture is in.
Step-14: The index 0 was marked was Bark, index 1 was marked as Sand, index 2 was marked as Straw and index 3 was marked as Grass.

III. Experimental Results

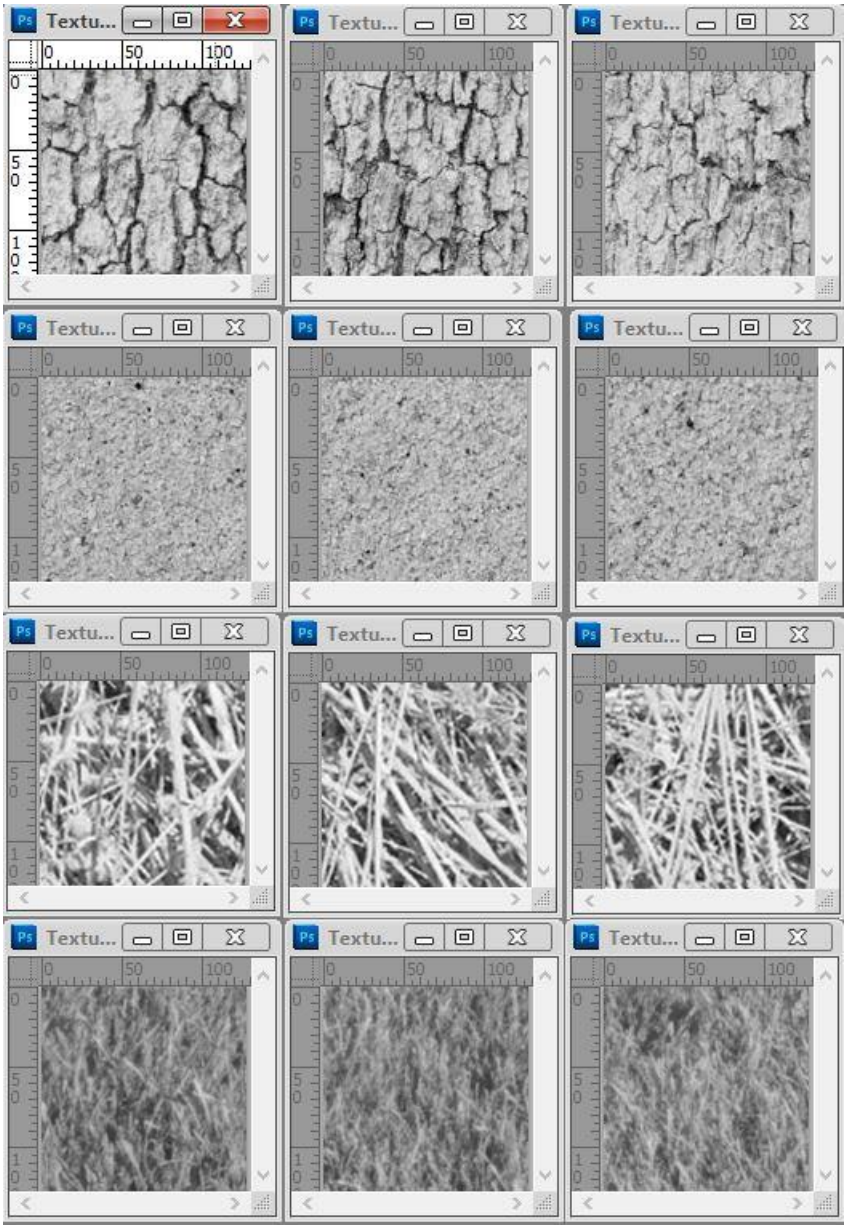

Figure-1.1: Textures 1 to 12
(Visual inspection tells 1st 3 are bark, next 3 are sand, next 3 are straw and next 3 are grass)

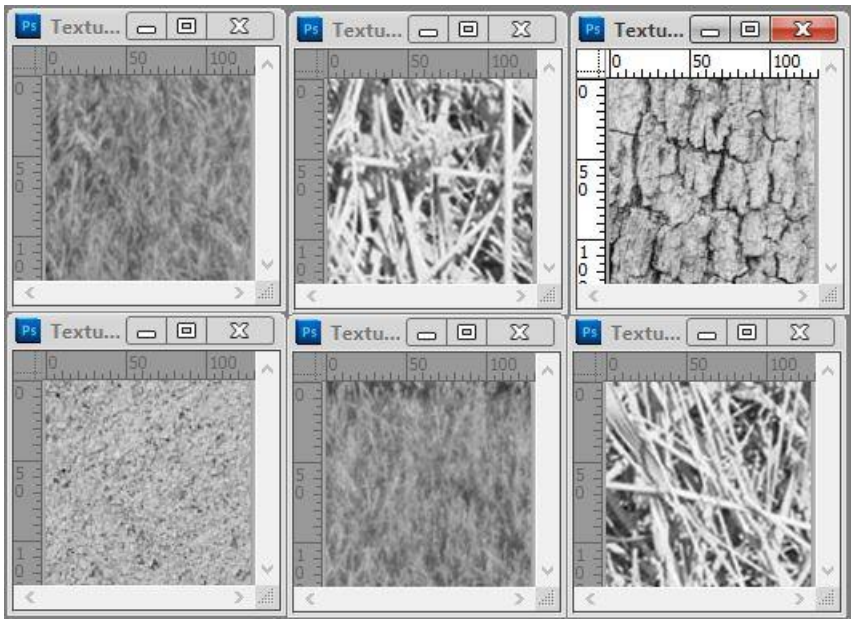

Figure-1.2: Textures A to F
(Visual inspection indicates A is grass, B is straw, C is bark, D is sand, E is grass and F is straw)

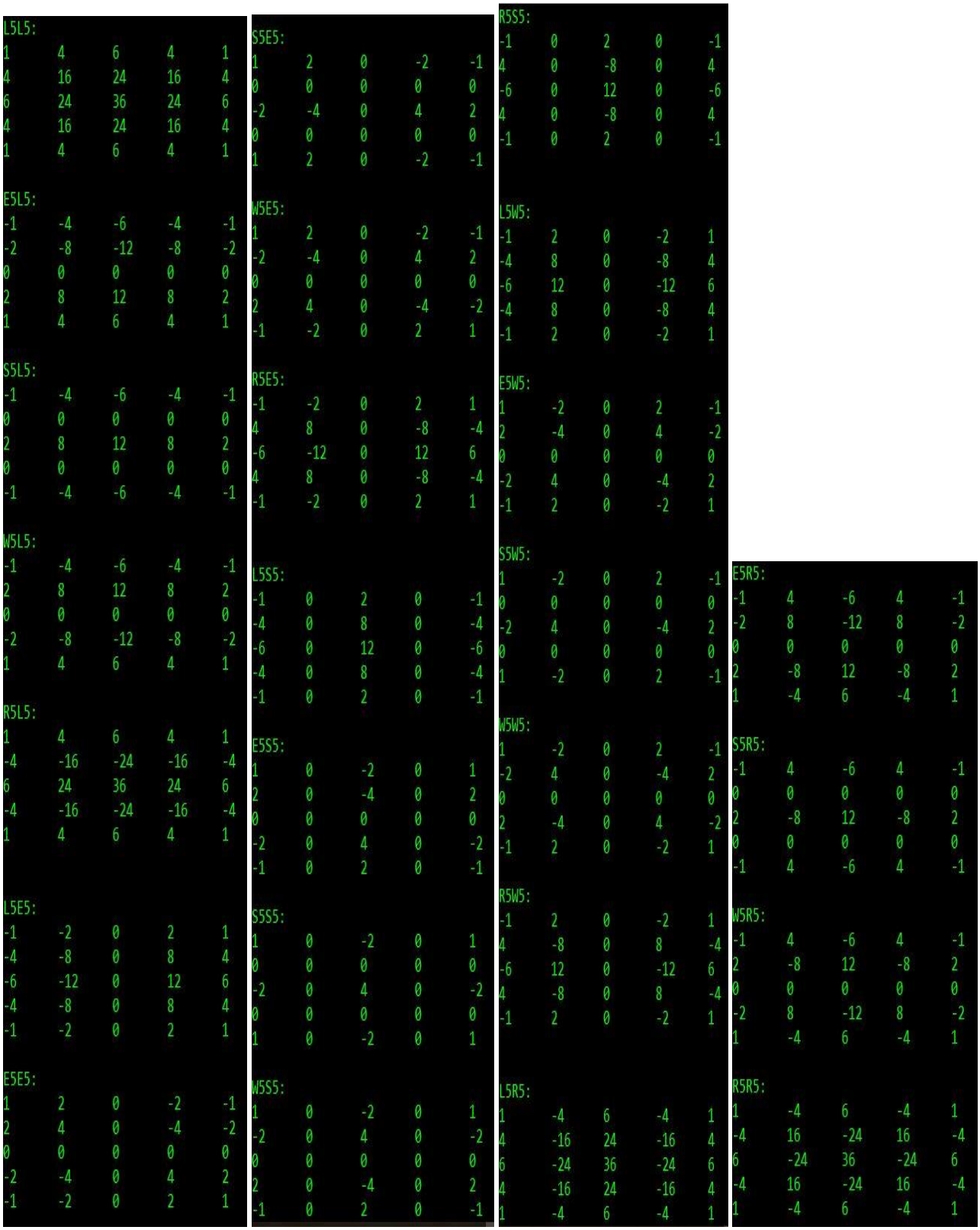

Figure-1.3: The 25 Laws' Filter

104516270.279800	2589913.099854	571487.761475	383100.858643	845340.045654	4706616.632324	169525.515625	43941.406738	32614.102539	74420.839355	
907009.408936	45929.644775	13590.613037	10745.505127	26036.420654	548890.909668	34733.931641	10874.957520	8876.541992	23254.116699	
1261334.807373	92011.498291	29320.363037	25182.710205	76906.444092						
89666093.811626	2505098.519897	629876.009644	479416.535522	1227723.501831	4848492.432739	181576.265747	53112.635864	44787.556763	120760.204224	
1208824.340698	59135.677124	18394.065308	16203.333374	44571.604370	849878.937622	50906.844849	16419.781372	14742.010864	42843.162231	
2056549.703003	152040.457397	51154.005737	46117.269897	151542.576050						
58868294.839600	2006116.943481	513215.270386	403694.307739	1045389.613159	3147585.405151	147339.951294	45242.675659	38327.229614	98009.721558	
883825.679565	46974.697388	15411.094604	13533.342896	35340.312378	674156.537964	40739.482544	13447.027222	11786.385864	33054.229370	
1709690.191284	124638.638794	41756.106323	36234.128052	115220.324097						
12107096.882996	1055248.600586	376568.198975	326908.958984	810327.762451	1015061.839844	93816.660889	34301.475586	30487.214600	78830.500000	
346663.325928	32409.166016	11724.328857	10315.266602	27851.118896	303838.645508	27782.753662	9730.601563	8475.455811	24556.133789	
877870.739014	78812.096680	26873.515381	23776.017578	78923.953857						
12967664.506153	1093323.643860	385867.230530	334175.626282	831676.236389	1074904.464661	100362.629944	36594.470520	32833.057678	84518.648254	
377520.326233	35169.020813	12750.832092	11487.323547	31107.955139	332654.359192	31217.963928	11138.513489	9893.040100	28136.167786	
962281.732483	90748.971985	32078.058655	29086.782532	92837.423889						
17241225.409409	1238774.197144	361193.668579	278487.774292	658565.232056	1278768.387817	102291.404175	32807.500122	27047.532104	66104.684692	
390124.819946	33363.464722	11388.510376	9618.385620	24263.855103	313852.034302	28221.269409	9712.349731	8197.475464	22211.456177	
889382.376587	84796.779175	28874.738892	24961.871948	77352.286743						
148020992.238956	3768523.781128	646271.627075	273814.719604	250887.247192	6632376.921753	296376.942017	58854.719604	25794.021118	24441.285034	
1414313.266724	77839.042847	16750.912231	7605.207886	7466.712036	675933.160034	41321.992798	9572.481323	4615.392212	5099.882690	
674211.665161	45648.660034	11396.443481	6308.911011	10724.876099						
148164273.006744	4566222.089478	734836.525269	318324.124634	322455.165894	7169905.774292	367436.832153	70751.623901	32170.755981	33025.688354	
1404203.698120	89169.068970	20140.077026	9887.166626	10683.030151	635605.904175	46129.010864	11514.753784	5994.669067	7380.833862	
610150.861206	49428.901978	13695.896362	8340.734009	16170.615112						
137543937.267944	4385353.906250	922019.676025	432214.766602	415842.475830	7348452.874268	309363.195313	68273.443604	34223.885742	35656.030518	
1628460.347900	79301.666992	18198.075439	9376.417969	10324.265869	723039.711182	42061.948242	10337.749268	5420.091797	6293.851807	
655241.589111	44777.835938	12011.160400	6886.133789	10779.210205						
28836472.085794	636226.136047	113884.955383	63494.365540	107188.878235	1547255.645569	56967.980530	11521.369202	6505.538147	11016.331116	
372445.429016	18672.466125	4320.308899	2529.648743	4469.216125	230458.631897	14337.005920	3677.449280	2285.469788	4560.567444	
410421.067688	29129.573547	8294.322571	5975.381165	15554.542297						
26433169.962639	596706.462280	113767.766235	64974.559937	108404.008423	1551749.015259	54971.141479	11353.538696	6570.198120	11304.993774	
376900.516235	17902.042358	4137.756470	2505.530640	4557.514282	235897.280884	13925.293823	3638.523071	2363.053589	4768.556274	
437643.145142	30596.604858	9233.979126	6864.390015	17229.783813						
29292877.963190	659376.524231	123117.083313	69752.926575	116805.833313	1454433.844788	60852.887756	12760.958069	7400.920959	12723.999084	
364103.020813	20485.114075	4842.788391	2939.047668	5393.100891	243258.854553	16000.571350	4151.358459	2656.760803	5453.149475	
468896.425110	33465.196106	9500.958313	6815.692200	17500.052063						

Figure-1.4: The 12x25 feature vectors for Texture1-Texture12 images

0.699628	0.353547	0.457348	0.699869	0.756398	0.506361	0.075570	0.199401	0.491035	0.561151	0.211830
-0.018178	0.176378	0.445095	0.505810	0.316257	0.202763	0.360782	0.469655	0.466004	0.691841	0.547309
0.473014	0.474551	0.435956								
0.416621	0.293355	0.691978	1.391916	1.776182	0.564991	0.192543	0.658685	1.478968	1.849312	0.846029
0.536292	1.065228	1.758213	1.905595	1.710592	1.539917	1.834973	2.028051	1.999419	2.293344	2.136763
2.068941	2.050573	2.048512								
-0.170309	-0.060766	0.223184	0.847837	1.289913	-0.137907	-0.139778	0.264567	0.954682	1.216885	0.163115
0.025700	0.513247	1.115831	1.208444	0.896554	0.699294	1.044613	1.242771	1.233149	1.594795	1.411214
1.382003	1.306538	1.263749								
-1.061461	-0.735586	-0.325925	0.296119	0.663023	-1.019170	-0.659311	-0.283354	0.318428	0.683733	-0.965616
-0.585850	-0.168967	0.341583	0.642857	-0.818954	-0.371951	0.056534	0.363090	0.567925	-0.080427	0.197814
0.294162	0.368651	0.479545								
-1.045061	-0.708565	-0.288558	0.348331	0.719958	-0.994440	-0.595771	-0.168524	0.508804	0.841854	-0.900776
-0.469975	0.020981	0.623572	0.888814	-0.685464	-0.087932	0.430853	0.739728	0.848167	0.089570	0.513880
0.674587	0.768462	0.780154								
-0.963617	-0.605340	-0.387707	-0.051797	0.258286	-0.910194	-0.577049	-0.358171	0.039281	0.329975	-0.874291
-0.545783	-0.231108	0.173918	0.371945	-0.772567	-0.335695	0.051681	0.289234	0.384386	-0.057243	0.356277
0.440441	0.457926	0.445588								
1.528720	1.189994	0.757863	-0.085374	-0.828957	1.302179	1.306875	0.946242	-0.062447	-0.828202	1.277819
1.321574	0.761173	-0.310439	-0.896582	0.904784	0.747455	0.014495	-0.662490	-0.955094	-0.490581	-0.680292
-0.837132	-0.946330	-0.993935								
1.531450	1.756111	1.113757	0.234435	-0.638091	1.524312	1.996630	1.542024	0.455055	-0.589569	1.256576
1.797278	1.388318	0.238585	-0.653684	0.717967	1.144893	0.530883	-0.296030	-0.776543	-0.619595	-0.580198
-0.669054	-0.793368	-0.876276								
1.329053	1.627751	1.865942	1.052762	-0.389035	1.598096	1.432929	1.417920	0.621676	-0.516450	1.727803
1.382984	1.028962	0.115702	-0.680778	1.123006	0.808633	0.217955	-0.448689	-0.861631	-0.528786	-0.703350
-0.792200	-0.902875	-0.992761								
-0.742641	-1.032961	-1.381503	-1.596567	-1.212188	-0.799242	-1.016989	-1.424154	-1.627799	-1.201395	-0.911440
-1.162601	-1.539038	-1.531585	-1.122954	-1.158889	-1.483626	-1.552806	-1.281527	-0.997311	-1.021836	-1.117686
-1.063881	-0.971439	-0.889587								
-0.788442	-1.061008	-1.381974	-1.585932	-1.208948	-0.797385	-1.036371	-1.432559	-1.622551	-1.193370	-0.902079
-1.194948	-1.572818	-1.537388	-1.116285	-1.133694	-1.517666	-1.563155	-1.260914	-0.981030	-0.967013	-1.078842
-0.995197	-0.904512	-0.853392								
-0.733943	-1.016532	-1.344405	-1.551598	-1.186541	-0.837600	-0.979279	-1.362077	-1.555134	-1.153924	-0.928970
-1.086495	-1.442356	-1.433086	-1.053182	-1.099592	-1.346085	-1.426808	-1.182879	-0.927441	-0.904071	-1.002887
-0.975683	-0.908178	-0.847553								

Figure-1.5: The 12x25 normalized feature vectors for Texture1-Texture12 images (Energy)

0.315313	0.195379	0.457503	0.979874	1.274164	0.311148	0.042779	0.374218	0.974895	1.209116	0.406991
0.181272	0.584951	1.106380	1.206616	0.974468	0.813991	1.080123	1.246826	1.232857	1.526660	1.365095
1.307986	1.277221	1.249405								
-1.023380	-0.683164	-0.334063	0.197551	0.547089	-0.974601	-0.610710	-0.270016	0.288838	0.618521	-0.913561
-0.533869	-0.126365	0.379691	0.634539	-0.758995	-0.265193	0.179689	0.464017	0.600159	-0.016033	0.355990
0.469730	0.531680	0.568429								
1.463074	1.524618	1.245854	0.400608	-0.618695	1.474862	1.578811	1.302062	0.338095	-0.644740	1.420733
1.500612	1.059484	0.014616	-0.743681	0.915252	0.900327	0.254444	-0.469070	-0.864423	-0.546321	-0.654613
-0.766129	-0.880857	-0.954324								
-0.755008	-1.036834	-1.369294	-1.578033	-1.202559	-0.811409	-1.010880	-1.406263	-1.601828	-1.182896	-0.914163
-1.148015	-1.518070	-1.500686	-1.097474	-1.130725	-1.449126	-1.514256	-1.241773	-0.968594	-0.964306	-1.066472
-1.011587	-0.928043	-0.863511								

Figure-1.6: The 4x25 normalized feature vectors for supervised learning of Texture1-Texture12 images (Energy)

2.11169782540309	-0.278479383483726	0.355039248498121
7.33260339302501	-2.29998115732306	1.10964926771877
4.19597761093209	-2.27715802049392	0.506491187262765
0.00388350538133244	-2.57767125079886	-1.14079870776889
0.982735785997355	-2.9250606543983	-1.07515933257984
-0.329342167283981	-2.27638197217334	-0.691050304248999
0.190063613543196	4.53546755026666	0.382488419327827
1.64196365175011	5.04316948863324	-0.50528044194721
1.66194525619898	5.02184622233876	-0.412947366622878
-6.05469903697591	-0.601160836214701	0.474764995924743
-6.01723381534303	-0.702738199554516	0.534291569090827
-5.71959616971981	-0.661852982618631	0.462510986301507

Figure-1.7: The 12x3 normalized feature vectors for Texture1-Texture12 images (Energy) after PCA

4.546760	-1.618540	0.657060
0.219092	-2.593038	-0.969003
1.164658	4.866828	-0.178580
-5.930510	-0.655251	0.490523

Figure-1.8: The 4x3 normalized feature vectors for supervised learning of Texture1-Texture12 images (Energy)

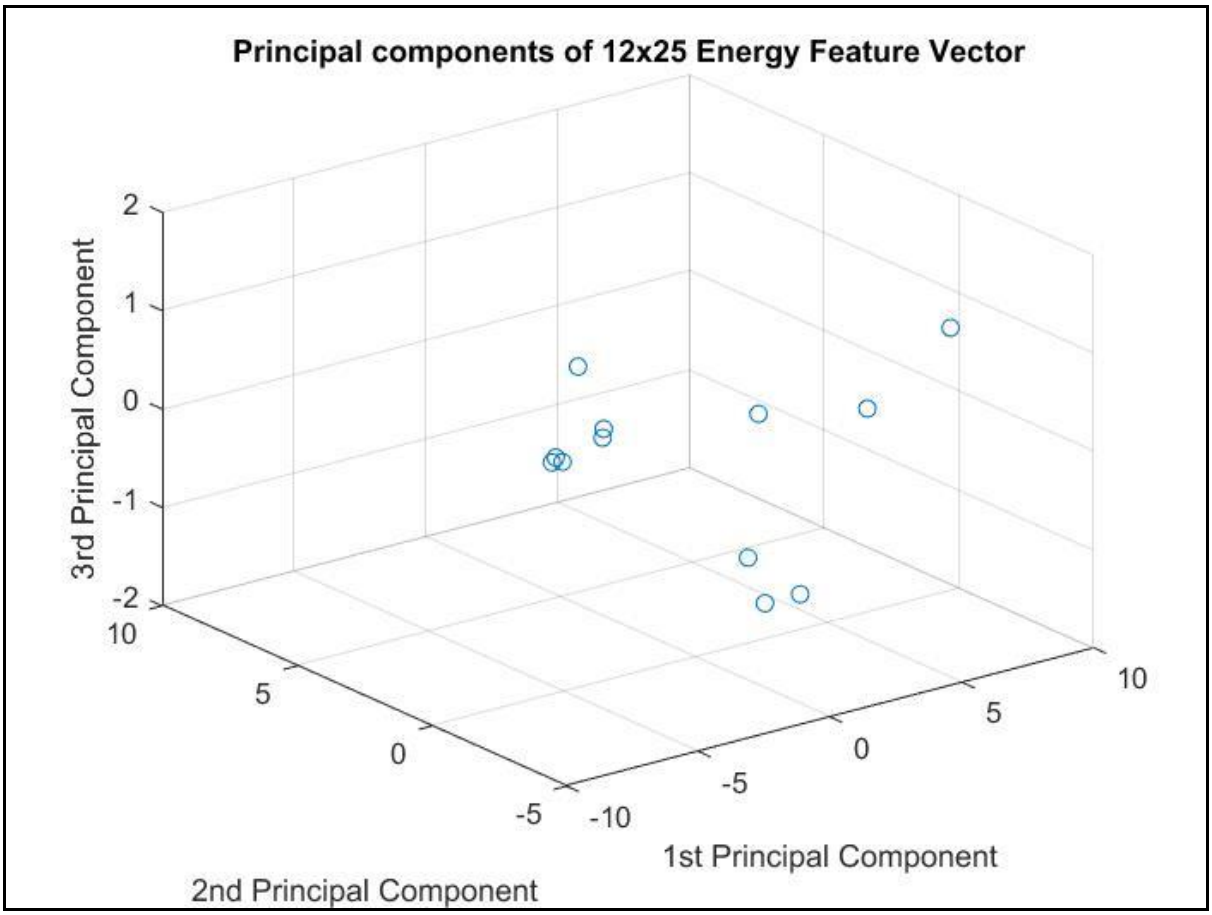

Figure-1.9: The 12x3 normalized feature vectors in feature space (Energy)

16826135.933231	451040.773315	83301.436646	45080.993042	75357.291138	948022.105591	41135.111694	8483.462036	4668.796265	7947.609497
242579.548950	13321.859253	3047.921021	1761.289917	3245.720825	159416.988403	10132.916382	2545.555786	1576.186890	3326.398560
301358.244263	21079.023315	5933.499146	4350.524292	12196.572388					
164603817.322420	3635509.883606	594229.586243	246956.033997	231600.443665	8166693.563293	291958.182922	52858.364075	22557.944641	21048.766418
1700236.319641	82161.970520	16290.462219	7155.550598	6877.483704	772752.650208	44820.738586	9680.161926	4522.195618	4936.275208
743161.381165	49132.844543	11874.812805	6455.354309	10496.498352					
79483096.088318	2308050.999268	604693.598389	470873.784424	1215872.559326	4345745.509521	171419.784912	51057.925537	43322.292725	113983.882568
1066031.156006	54562.467041	16973.673584	14770.869385	38915.361084	749415.170654	45833.397217	14402.391357	12438.865967	34703.106201
1821353.385498	131875.753174	42239.660889	36608.757080	120226.153076					
11893989.713255	1012861.406677	367925.949158	330628.015076	904758.344666	990216.057556	98839.078552	37758.435486	34385.181091	90225.211853
361381.960876	36358.022888	13856.110291	12514.646912	32633.458923	332720.328064	32791.272888	12235.127869	11060.422302	30472.826111
1005563.514587	99085.152771	36862.273376	34504.323669	108330.356384					
19319556.692839	377835.139221	69927.327942	41366.395081	72138.384583	923860.080383	32888.314514	6966.457336	4167.720764	7468.379211
236773.810364	11025.934143	2522.521301	1563.643127	3033.437317	156326.730774	8878.500061	2225.123352	1459.187561	3211.576477
290695.382629	19587.525940	5678.609192	4281.359924	11600.712708					
156455885.922981	4384963.321777	754057.769043	316902.512207	294554.039551	8094074.366455	369661.026611	69065.068604	29965.033447	28363.241455
1807809.094238	101475.966309	20334.663574	9273.328613	9343.543457	846534.591064	54606.973877	11982.371338	5852.605713	6640.700439
788201.021973	58309.069824	14866.575684	8617.432129	14402.549316					

Figure-1.10: The 6x25 feature vectors for TextureA-TextureF images

-0.894875	-1.016117	-1.252175	-1.266929	-0.892847	-0.921643	-1.004506	-1.266993	-1.266325	-0.884590	-0.992155
-1.087681	-1.321743	-1.220794	-0.857866	-1.168049	-1.284411	-1.321116	-1.071881	-0.791712	-1.016343	-1.036299
-0.936523	-0.814734	-0.704643								
1.387625	1.035312	0.692097	0.032097	-0.535478	1.323416	0.986981	0.657465	-0.042410	-0.570452	1.199457
0.963952	0.596847	-0.137439	-0.607199	0.917898	0.677340	0.175121	-0.381717	-0.670963	-0.158932	-0.345736
-0.528664	-0.664995	-0.739864								
0.072893	0.180165	0.731916	1.472964	1.715811	0.135074	0.029929	0.579383	1.378216	1.657936	0.245919
0.141408	0.695831	1.391982	1.604078	0.838528	0.734610	1.165445	1.472929	1.561708	1.933522	1.691034
1.555820	1.480139	1.533443								
-0.971055	-0.654193	-0.169073	0.570510	1.004211	-0.908520	-0.546347	0.002609	0.766769	1.088253	-0.813533
-0.401137	0.244156	0.938854	1.170496	-0.578647	-0.002980	0.710937	1.150000	1.244414	0.350313	0.883872
1.186673	1.330428	1.286994								
-0.856363	-1.063276	-1.303068	-1.290832	-0.900210	-0.929157	-1.069984	-1.332783	-1.300607	-0.896081	-1.000884
-1.156106	-1.397864	-1.260489	-0.872518	-1.178559	-1.355354	-1.388316	-1.099290	-0.800325	-1.037036	-1.073013
-0.954021	-0.819654	-0.716988								
1.261776	1.518108	1.300302	0.482189	-0.391487	1.300830	1.603927	1.360319	0.464358	-0.395066	1.361195
1.539564	1.182774	0.287885	-0.436990	1.168829	1.230795	0.657930	-0.070040	-0.543122	-0.071524	-0.119857
-0.323286	-0.511183	-0.658941								

Figure-1.11: The 6x25 normalized feature vectors for TextureA-TextureF images (Energy)

-5.27427906030382	-0.800529493831125	0.0855036354478724
0.723095271753327	3.60850079178643	0.271691969321562
5.46827344041608	-2.46867812710221	0.72871823896437
2.14402611869243	-3.59911624834686	-0.804361719372202
-5.43408251362269	-0.851672424713016	0.171336455958798
2.37296736799283	4.11149448156627	-0.452888976079672

Figure-1.12: The 6x3 normalized feature vectors for TextureA-TextureF images (Energy) after PCA

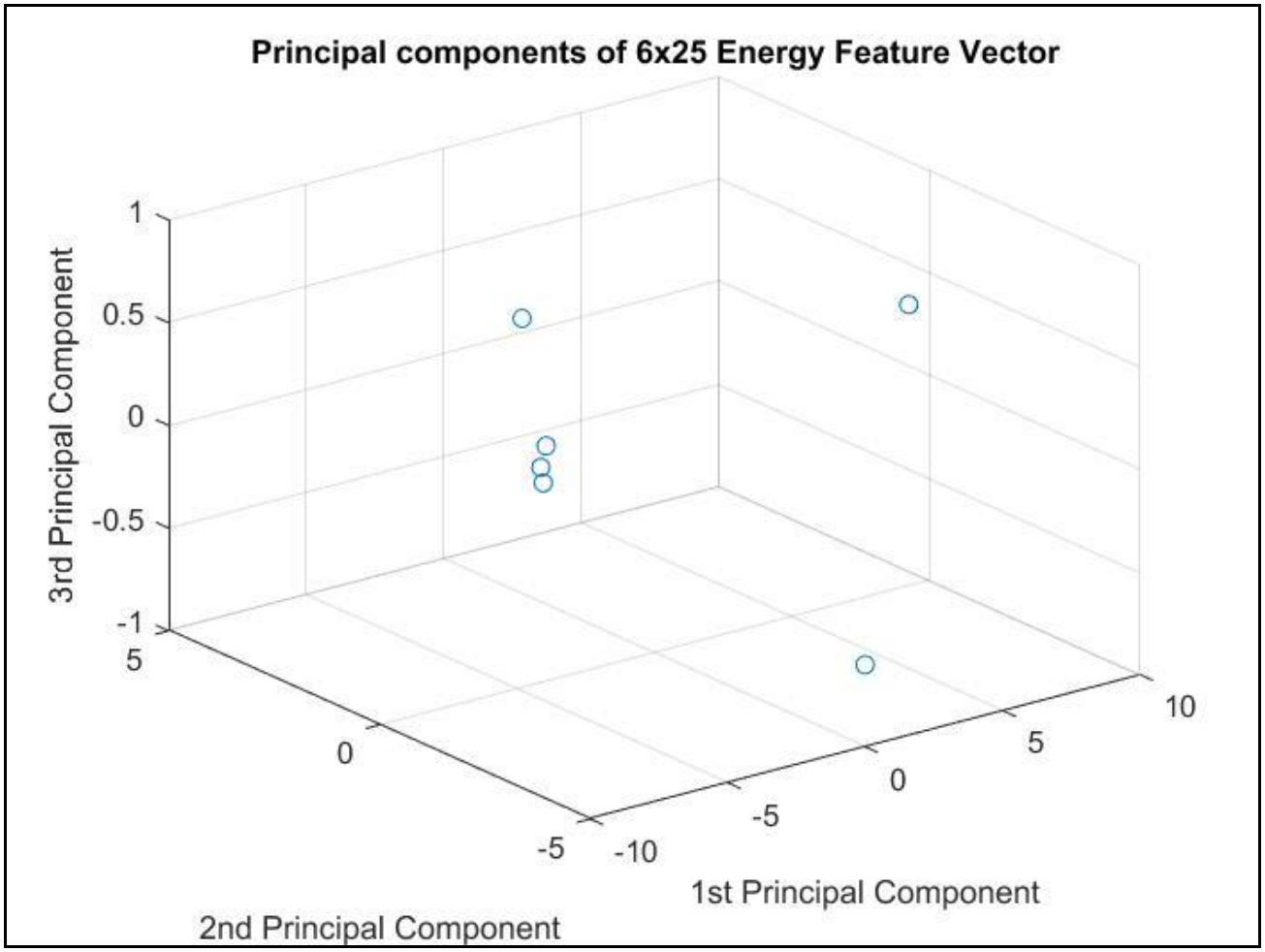

Figure-1.13: The 6x3 normalized feature vectors in feature space (Energy)

```
Texture 1 is classifed as Bark !
Texture 2 is classifed as Bark !
Texture 3 is classifed as Bark !
Texture 4 is classifed as Sand !
Texture 5 is classifed as Sand !
Texture 6 is classifed as Sand !
Texture 7 is classifed as Straw !
Texture 8 is classifed as Straw !
Texture 9 is classifed as Straw !
Texture 10 is classifed as Grass !
Texture 11 is classifed as Grass !
Texture 12 is classifed as Grass !

Iteration-->    20      completed !!!
```

Figure-1.14: Texture1-12 classification without PCA

```
Texture A is classifed as Grass !
Texture B is classifed as Straw !
Texture C is classifed as Bark !
Texture D is classifed as Sand !
Texture E is classifed as Grass !
Texture F is classifed as Straw !
```

Figure-1.15: TextureA-F classification without PCA (unsupervised)

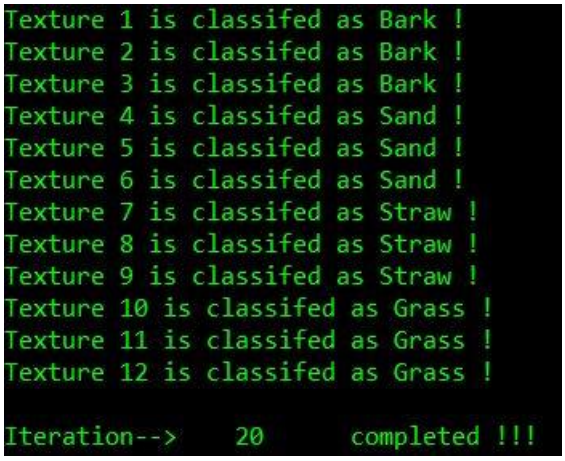

Figure-1.16: Texture1-12 classification with PCA

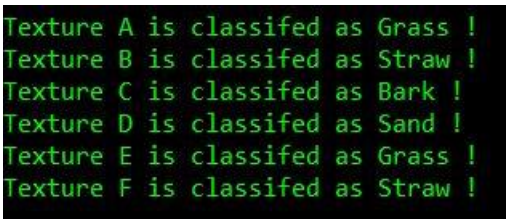

Figure-1.17: TextureA-F classification with PCA (unsupervised)

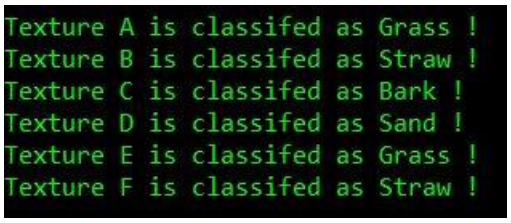

Figure-1.18: TextureA-F classification without PCA (supervised)

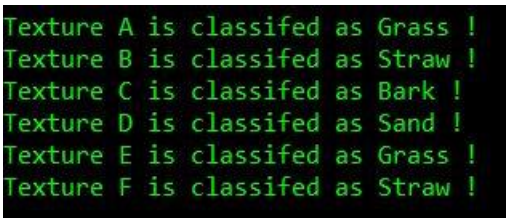

Figure-1.19: TextureA-F classification with PCA (supervised)

IV. Discussion

1. Boundary extension was done while applying Laws' filters based on the pixel replication method.
2. The discriminant power of features can be found by studying the number of data points overlapping after applying the filter over images. The features that have many overlapping points do not classify the data with high accuracy and have a very weak discriminant power. The features which have minimal amount of overlapping points can give a very good classification and has strong discriminant power.

   Another way to look at it is by studying the feature vector components. It can be seen for the above figures that the first three rows of the feature vectors in figure above represent the Bark, the next three rows represent the Sand, the next three rows represent the Straw and the next three rows represent the Grass. I picked up 1st row each from that of Bark and Sand and calculated the absolute difference between that of each of the 25 Filters applied. The data has been tabulated below:

Filter Bank Type	Bark (Texture-1) Feature Vectors	Sand (Texture-4) Feature Vectors	Absolute Difference
L5L5	0.699628	-1.061461	1.761089
E5L5	0.353547	-0.735586	1.089133
S5L5	0.457348	-0.325925	0.783273
W5L5	0.699869	0.296119	0.40375
R5L5	0.756398	0.663023	0.093375
L5E5	0.506361	-1.01917	1.525531
E5E5	0.07557	-0.659311	0.734881

S5E5	0.199401	-0.283354	0.482755
W5E5	0.491035	0.318428	0.172607
R5E5	0.561151	0.683733	0.122582
L5S5	0.21183	-0.965616	1.177446
E5S5	-0.018178	-0.58585	0.567672
S5S5	0.176378	-0.168967	0.345345
W5S5	0.445095	0.341583	0.103512
R5S5	0.50581	0.642857	0.137047
L5W5	0.316257	-0.818954	1.135211
E5W5	0.202763	-0.371951	0.574714
S5W5	0.360782	0.056534	0.304248
W5W5	0.469655	0.36309	0.106565
R5W5	0.466004	0.567925	0.101921
L5R5	0.691841	-0.080427	0.772268
E5R5	0.547309	0.197814	0.349495
S5R5	0.473014	0.294162	0.178852
W5R5	0.474551	0.368651	0.1059
R5R5	0.435956	0.479545	0.043589

After analyzing the various features above, the strongest feature was L5-L5 and the weakest feature was R5-R5. The more the absolute difference the better they are apart from each other and hence stronger is their discriminant power. The lesser the absolute difference, the more are they close to each other and hence weaker the discriminant power.

3. Plots are shown above in Figure-1.9 and Figure-1.13 above.
4. Feature reduction did not have a significant change in the output of the clustering algorithm. However, since the dimensional complexity reduced from 25D to 3D, the feature convergence for K-means in 3D was much faster as compared to 25D. This was done for various iterations and outputs were found to be consistent.
5. Both supervised and unsupervised classification gave a 100% classification output which can be seen in the figures above.

b. Texture Segmentation

I. Abstract and Motivation

Texture of an image defines the complete realm of what an image might look like. It is the heart of the image and the one which defines it. These textures are the ones that define how an image would look like. The intricate details they provide helps in classifying the image as well as segmenting the different components present in them. The spatial arrangement of the various intensity values is what the texture provides.

Task: Perform texture segmentation over a given image and discuss on the outputs.

II. Approach and Procedures

Texture segmentation is a special type of machine learning problem that needs the application of the very famous K-means clustering to cluster the feature points by generating a codebook and extract neat features out of it and finally segment the image with color codes to represent a segmented image. This method is called the unsupervised learning. The generic algorithm followed for K-means clustering is as under:

Step-1: Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

Step-2: Assign each object to the group that has the closest centroid.
Step-3: When all objects have been assigned, recalculate the positions of the K centroids.
Step-4: Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Theoretical Approach:

1. **Laws feature extraction:** Apply all 25 Laws filters to the input image and get 25 gray–scale images.
2. **Energy feature computation:** Use a window approach to computer the energy measure for each pixel based on the results from Step 1. You may try a couple of different window sizes. After this step, you will obtain 25-D energy feature vector for each pixel. Formula to be used is:

$$Energy = \frac{1}{(WindowSize * WindowSize)} \times \sum_{i=0,j=0}^{i=Height,j=Width} (Img(i,j) * Img(i,j)) \quad \rightarrow (3)$$

3. **Energy feature normalization:** All kernels have a zero-mean except for $L5^TL5$. The feature extracted by the filter $L5^TL5$ is not a useful feature for texture classification and segmentation. Use its energy to normal all other features at each pixel.
4. **Segmentation:** Use the k–means algorithm to perform segmentation on the composite texture images given in question based on the 25-D energy feature vectors.

Algorithm used to develop in C++:

<u>Steps for segmentation of image without PCA:</u>

Step-1: Read the image and store them in 3D array.
Step-2: Convert RGB to Gray using the formula:

$$Grayimage = 0.2989R + 0.5870G + 0.1140B \quad \rightarrow (4)$$

Step-3: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a (Image Height * Image Width) x 25 feature vector matrix.
Step-6: Calculate the energy using 13x13 window as per eqn-3 above and store it in a new 2D array.
Step-7: Normalize the energy matrix by $L5^TL5$ throughout.
Step-7: Perform the K-means clustering over the (Image Height * Image Width) x 25 data by intuitively selecting the initial K centroids until several iterations till there is no more change in the value of centroids.

The initial centroids selected were based on visual inspection of textures for window 13x13, 15x15, 31x31:

<u>For K=5:</u>

Black cat 2	(284, 87)
Orange cat 1 in-between black cats	(233, 155)
Orange cat 3	(475, 124)
Couch	(384, 265)

Background	(31, 63)

For K=3:

Orange cat 1 in-between black cats	(233, 155)
Couch	(384, 265)
Background	(31, 63)

For K=7:

Black cat 1	(38, 114)
Orange cat 1 in-between black cats	(233, 155)
Black cat 2	(284, 87)
Orange cat 2	(403, 170)
Orange cat 3	(475, 124)
Couch	(384, 265)
Background	(31, 63)

Step-8: Based on the Euclidean distance obtained for every centroid value, the min value's index indicates the class and type of texture it is.

Step-9: The new clusters were assigned based on the index value and after the final centroid calculation, based on the assignment, the new Centroid value was passed onto the input gray image.

Step-10: The new image showing segmentation was made using different color combination based on the value of the Centroid. Every pixel with any of the K centroid values was given one color. So, in total there were K colors seen in the final image.

III. Experimental Results

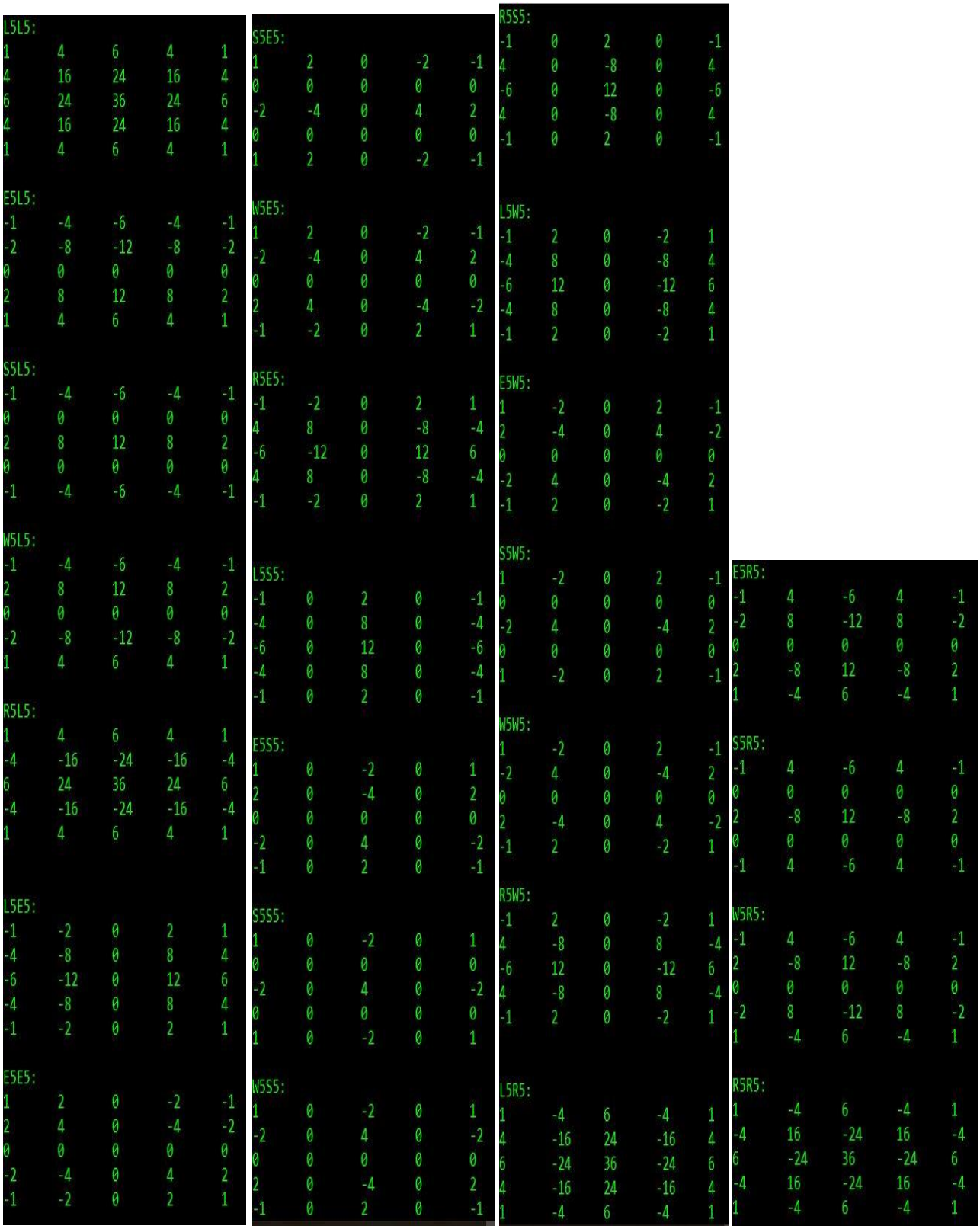

Figure-1.20: The 25 Laws' Filter

Figure-1.21: The input kitten

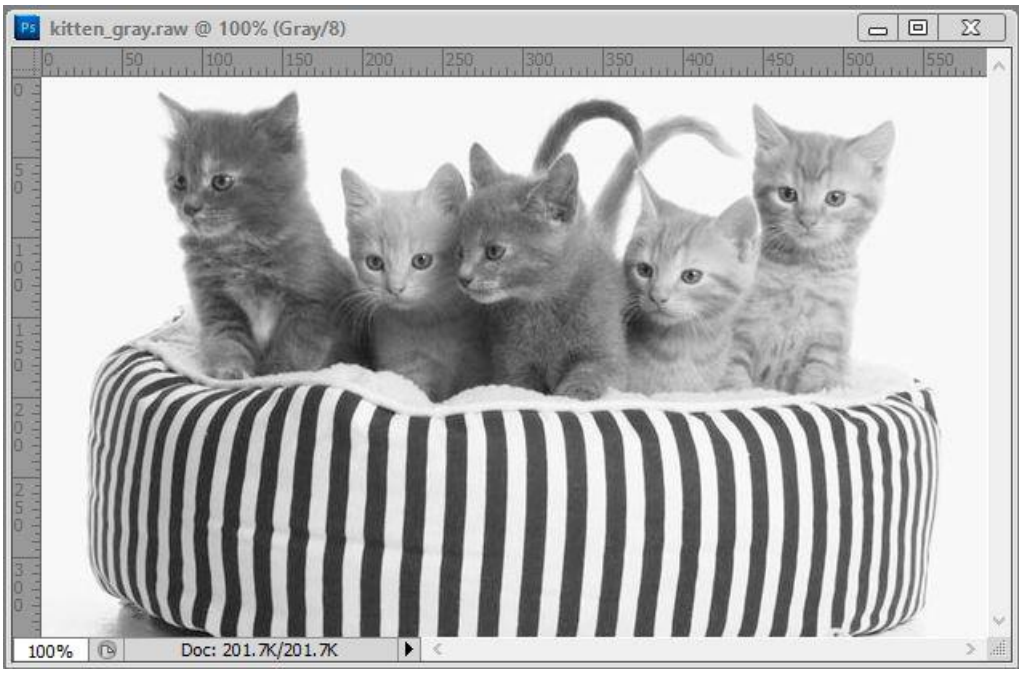

Figure-1.22: The grayscale kitten

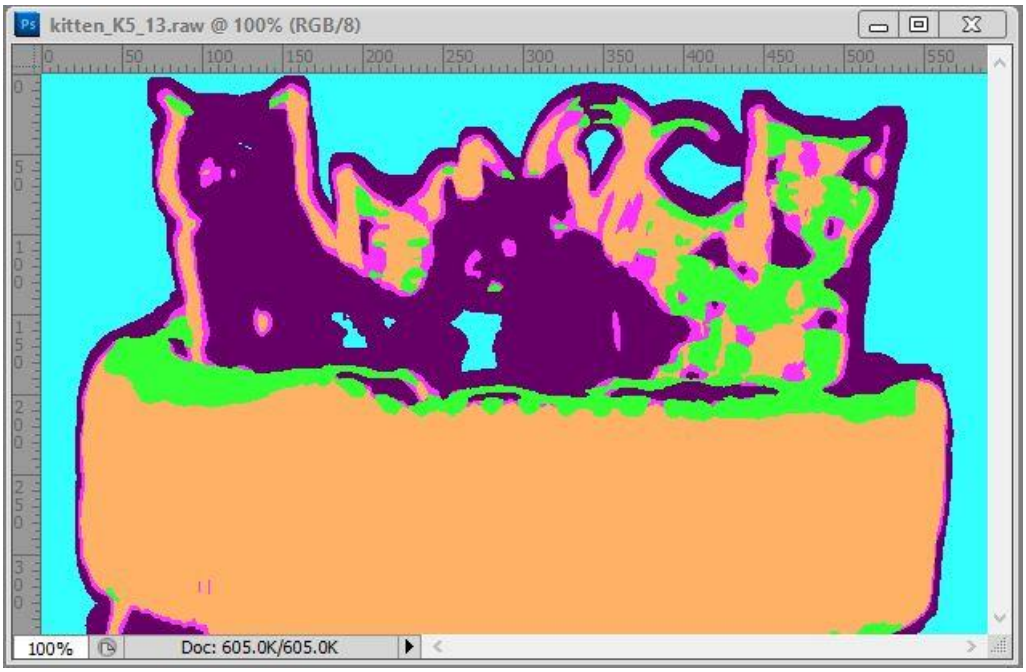

Figure-1.23: The segmented kitten for K = 5 and 13x13 window

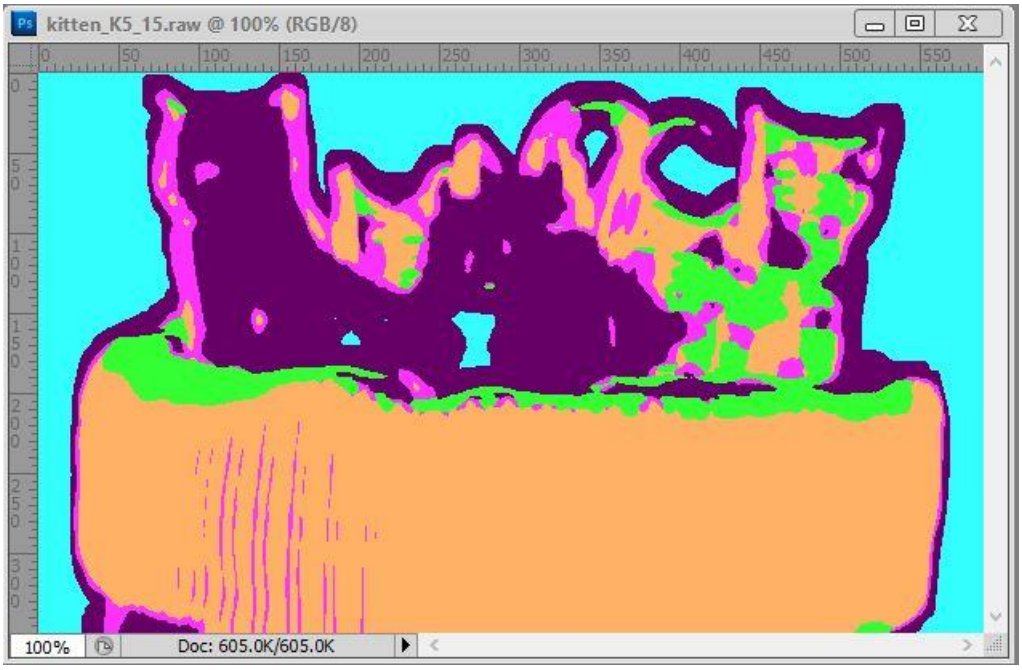

Figure-1.24: The segmented kitten for K = 5 and 15x15 window

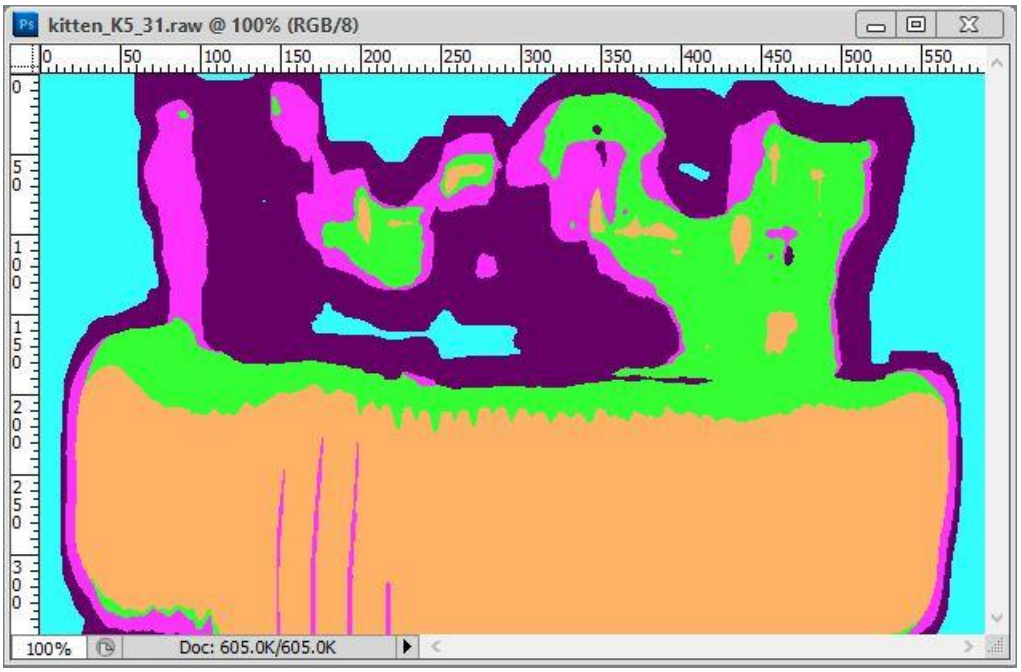

Figure-1.25: The segmented kitten for K = 5 and 31x31 window

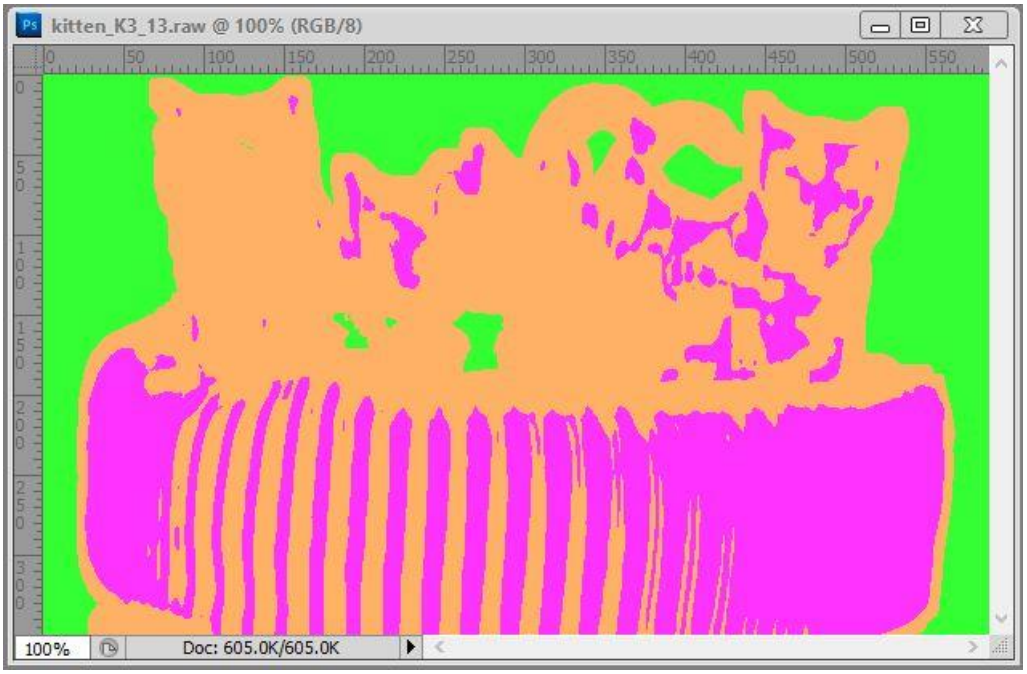

Figure-1.26: The segmented kitten for K = 3 and 13x13 window

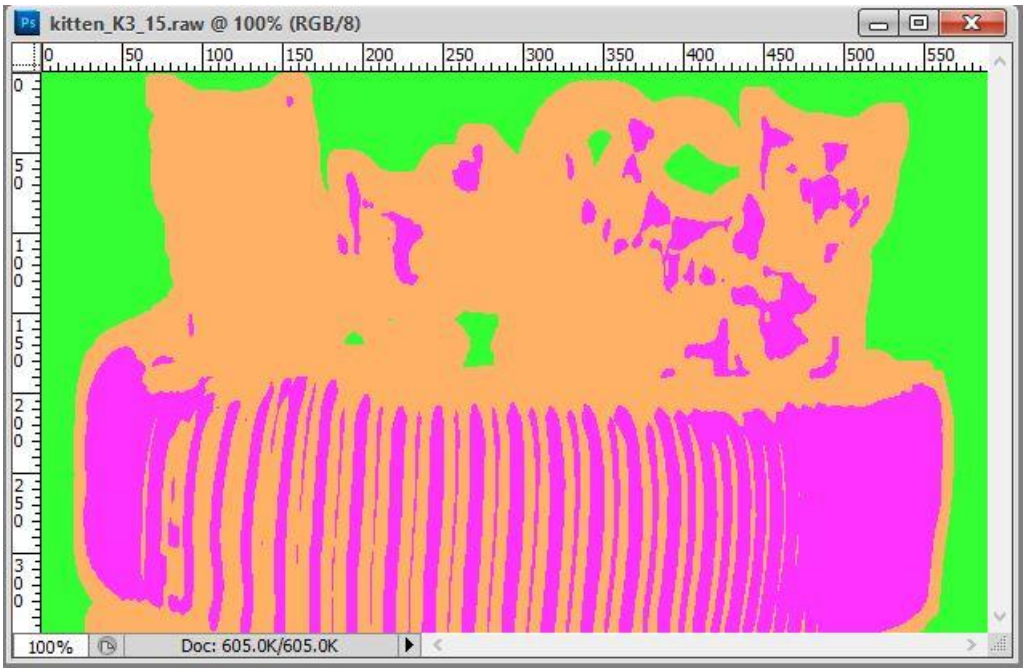

Figure-1.27: The segmented kitten for K = 3 and 15x15 window

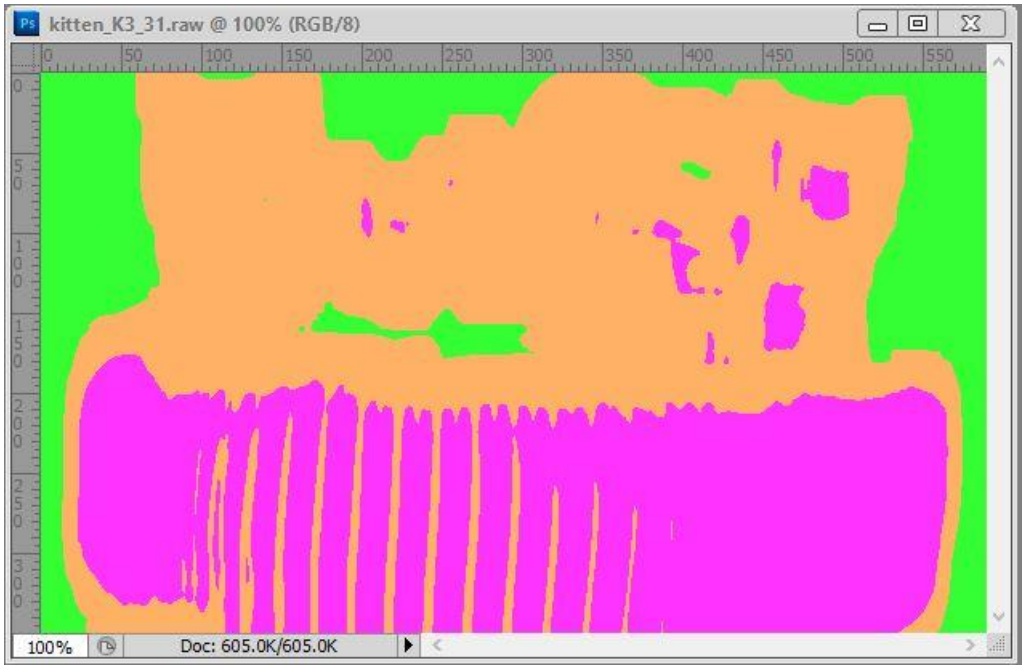

Figure-1.28: The segmented kitten for K = 3 and 31x31 window

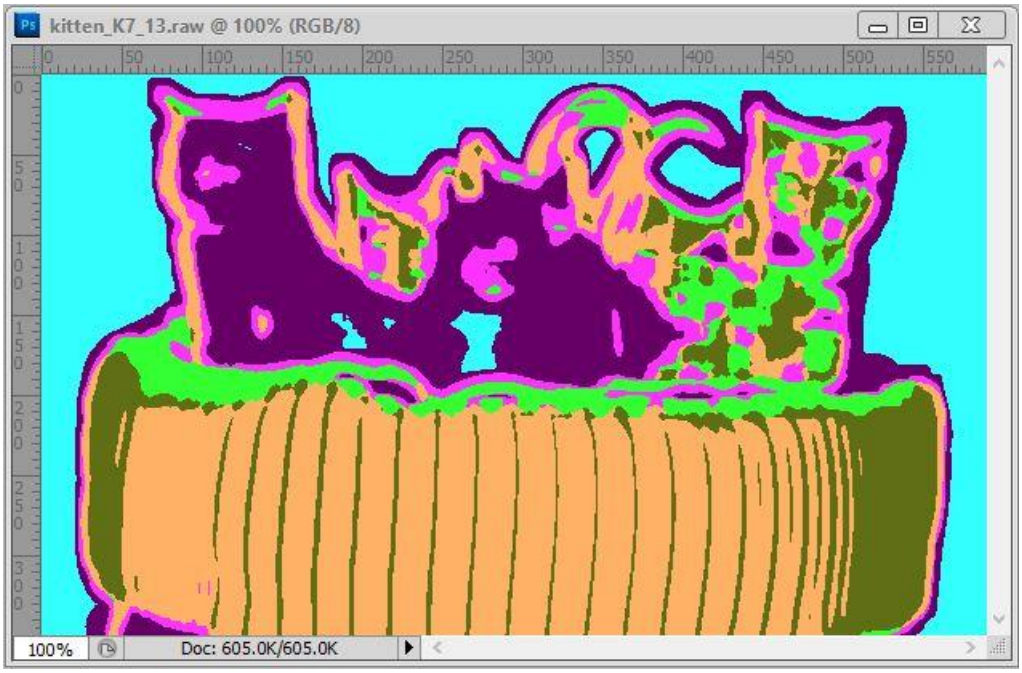

Figure-1.29: The segmented kitten for K = 7 and 13x13 window

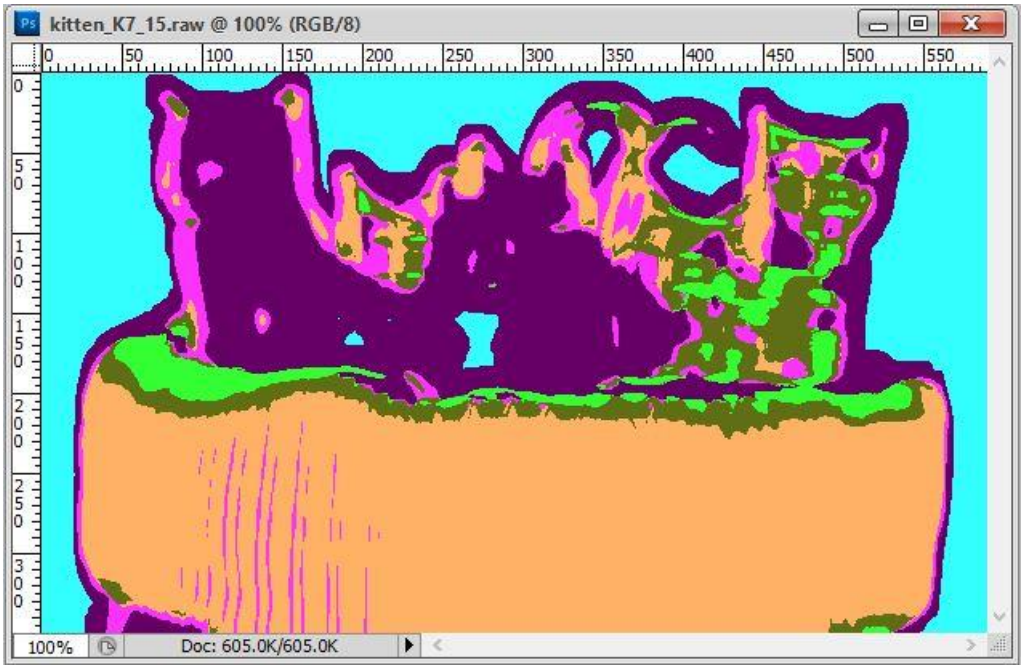

Figure-1.30: The segmented kitten for K = 7 and 15x15 window

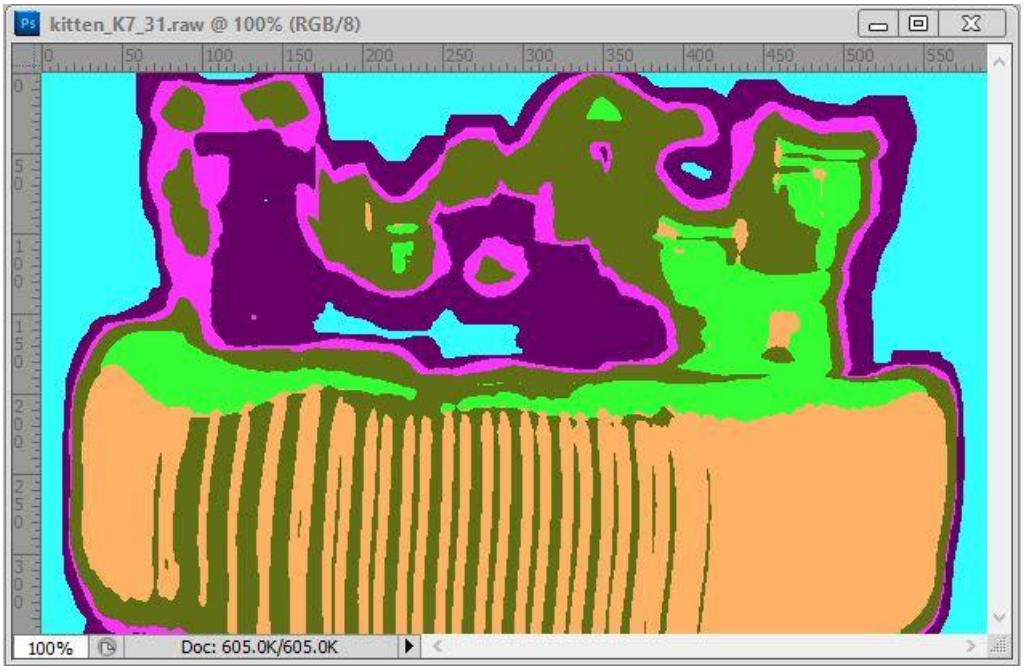

Figure-1.31: The segmented kitten for K = 7 and 31x31 window

IV. Discussion

I got a good segmentation output where a difference can be made that there are like five different textures in the image. I tested the output at different window sizes and different values K. For K=5 and window size 15x15 I got the best segmentation compared to the rest.

However, still the image lacks clarity. The clarity of the segmentation all depends on the initial value of K chosen. Better the value of K, better the segmentation. Generally, for segmentation, the larger the window size, the better the segmentation. But once a large enough window size is achieved, the output does not get affected on further increasing the window size.

Methods for improvement is discussed in the section below.

c. Further Improvement

I. Abstract and Motivation

Texture of an image defines the complete realm of what an image might look like. It is the heart of the image and the one which defines it. These textures are the ones that define how an image would look like. The intricate details they provide helps in classifying the image as well as segmenting the different components present in them. The spatial arrangement of the various intensity values is what the texture provides.

Task: Perform improvement of texture segmentation over the given image in above section and discuss on the outputs.

II. Approach and Procedures

Texture segmentation is a special type of machine learning problem that needs the application of the very famous K-means clustering to cluster the feature points by generating a codebook and extract neat features out of it and finally assign color cods to specific textures to get a final segmented output. This method is called the unsupervised learning. The generic algorithm followed for K-means clustering is as under:

Step-1: Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
Step-2: Assign each object to the group that has the closest centroid.
Step-3: When all objects have been assigned, recalculate the positions of the K centroids.
Step-4: Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Theoretical Approach:

1. **Laws feature extraction:** Apply all 25 Laws filters to the input image and get 25 gray–scale images.
2. **Energy feature computation:** Use a window approach to computer the energy measure for each pixel based on the results from Step 1. You may try a couple of different window sizes. After this step, you will obtain 25-D energy feature vector for each pixel. Formula to be used is:

$$Energy = \frac{1}{(WindowSize * WindowSize)} \times \sum_{i=0,j=0}^{i=Height,j=Width} (Img(i,j) * Img(i,j)) \quad \rightarrow (5)$$

3. **Energy feature normalization:** All kernels have a zero-mean except for $L5^TL5$. The feature extracted by the filter $L5^TL5$ is not a useful feature for texture classification and segmentation. Use its energy to normal all other features at each pixel.
4. **Segmentation:** Use the k–means algorithm to perform segmentation on the composite texture images given in question based on the 25-D energy feature vectors.
5. There are different ways to improve the segmentation as suggested in the question manual:
    - Consider the segmentation task in two spirals – the first spiral gives the coarse segmentation result while the second spiral offers fine-tuned segmentation results.
    - Develop a post-processing technique to merge small holes.
    - Enhance the boundary of two adjacent regions by focusing on the texture properties in these two regions only.
    - Adopt the PCA for feature reduction and, thus, cleaning.

Algorithm used to develop in C++:

<u>Steps for segmentation of image with PCA for improvement:</u>

Step-1: Read the image and store them in 3D array.
Step-2: Convert RGB to Gray using the formula:

$$Grayimage = 0.2989R + 0.5870G + 0.1140B \quad \rightarrow (6)$$

Step-3: Convert all the datatypes of the images to double for smooth manipulation of pixels.
Step-3: Find the Mean of each of the images as per eqn-1 above.
Step-4: Subtract the mean from each of the images to get the DC component removed image a store the data into a new 2d array.
Step-5: Create the 25Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a (Image Height * Image Width) x 25 feature vector matrix.
Step-6: Calculate the energy using 13x13 window as per eqn-3 above and store it in a new 2D array.
Step-7: Normalize the energy matrix by $L5^TL5$ throughout.
Step-8: Perform PCA and reduce the (Image Height * Image Width) x 25 data to (Image Height * Image Width) x 3 data.

Step-7: Perform the K-means clustering over the (Image Height * Image Width) x 3 data by intuitively selecting the initial K centroids until several iterations till there is no more change in the value of centroids.

The initial centroids selected were based on visual inspection of textures for window 15x15:

For K=5:

Black cat 2	(284, 87)
Orange cat 1 in-between black cats	(233, 155)
Orange cat 3	(475, 124)
Couch	(384, 265)
Background	(31, 63)

Step-8: Based on the Euclidean distance obtained for every centroid value, the min value's index indicates the class and type of texture it is.

Step-9: The new clusters were assigned based on the index value and after the final centroid calculation, based on the assignment, the new Centroid value was passed onto the input gray image.

Step-10: The new image showing segmentation was made using different color combination based on the value of the Centroid. Every pixel with any of the K centroid values was given one color. So, in total there were K colors seen in the final image.

III. Experimental Results

Figure-1.32: The input kitten

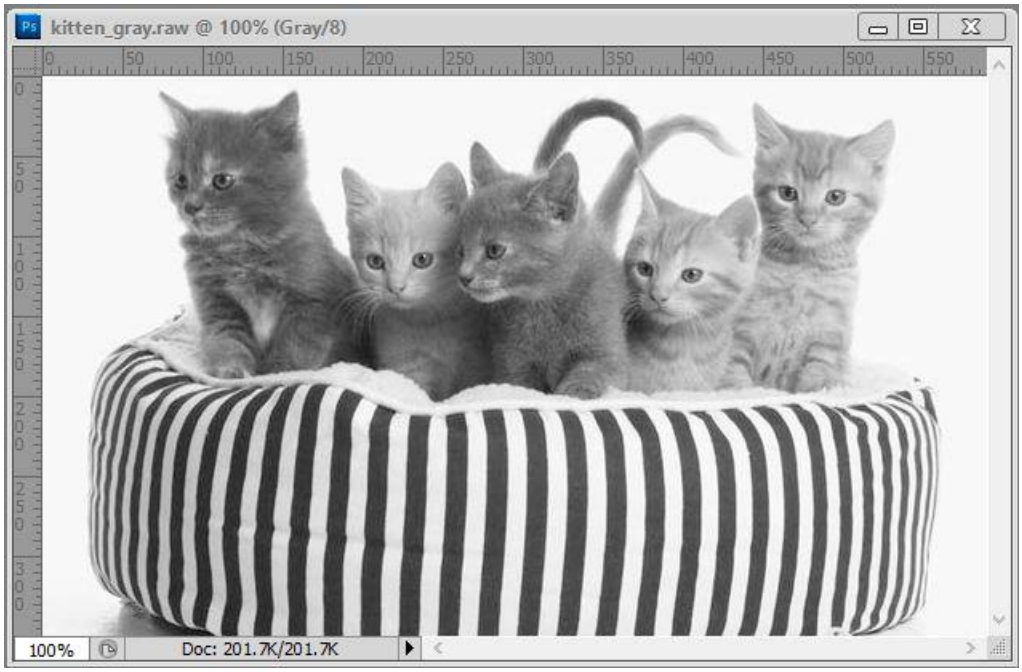

Figure-1.33: The grayscale kitten

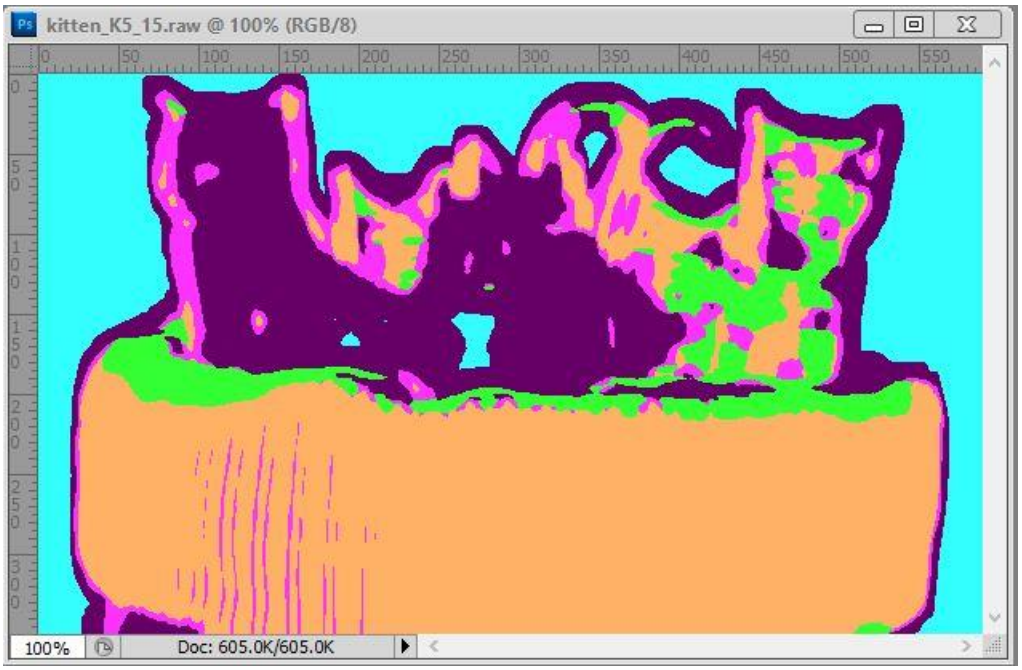

Figure-1.34: The segmented kitten for K = 5 and 15x15 window without PCA

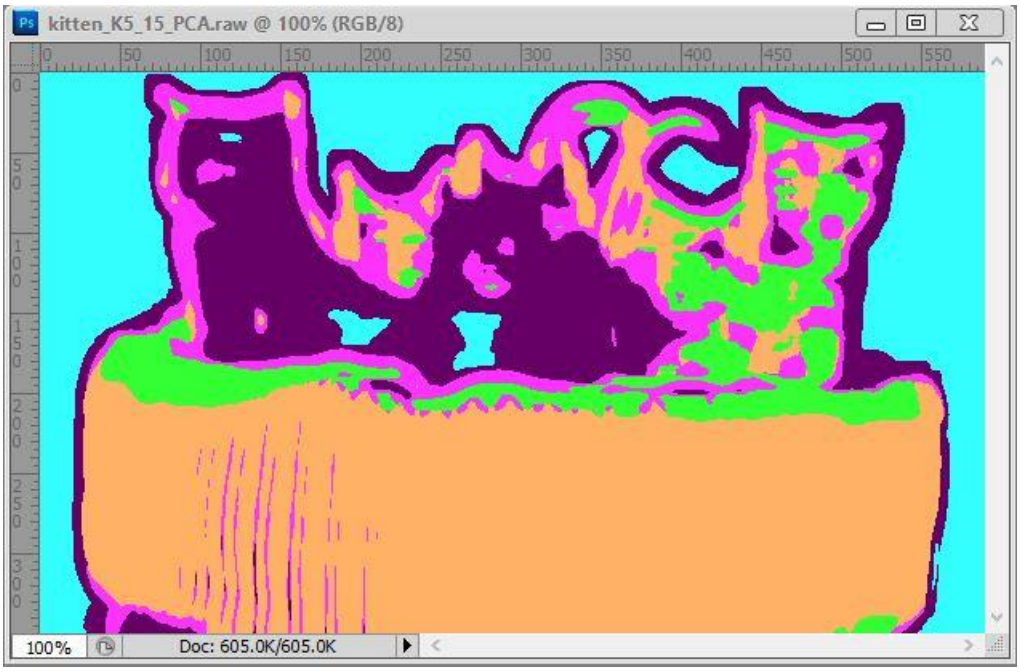

Figure-1.35: The segmented kitten for K = 5 and 15x15 window with PCA

IV. Discussion

As seen in the Theoretical approach section there are multiple ways by which we can improve the segmentation. For me the best result that I got without PCA in the earlier section was that with K = 5 and window size 15x15.

I tried using two methods i.e. the method of two spirals where the first spiral was expected to give coarse segmentation and the second spiral was expected to give fine segmentation but in my case the segmentation worsened even further on following this approach. The approach I took to perform a coarse segmentation was to first take K=3 and window size 15x15 and perform a coarse segmentation over the input image. Then a fine segmentation was performed over the obtained output from the coarse segmentation with K=5 and window size remaining the same i.e. 15x15 but then the output was even worse.

The second method that I used was that using PCA, where the output was much better than the one obtained in the method above as seen in the figures above. The PCA output still shows a comparatively good segmentation output where I can differentiate that there are four major kinds of textures in the image i.e. 2 black cats, 3 orange cats, 1 couch and 1 background.

However, still the image lacks clarity if we consider a target of fine segmentation. The clarity of the segmentation all depends on the initial value of K chosen. Better the value of K, better the segmentation. Generally, for segmentation, the larger the window size, the better the segmentation. But once a large enough window size is achieved, the output does not get affected on further increasing the window size.

Problem-2: Edge and Contour Detection

a. Canny Edge Detector

I. Abstract and Motivation

Canny edge detection is the most common edge detection algorithm used. Edge detection is one of the most fundamental procedure to be carried in Computer Vision and Image Processing applications. It is a pre-processing used widely is various IP/CV applications. We have various edge detection algorithms because till now we have not been able to come up with a single edge detection algorithm for every image.

Canny Edge detection is a multi-stage algorithm developed by John Canny. His main aim was to solve two main criterions in edge detection. Firstly, detection of all edges in image with very low error rate. Secondly, localize edge points such that distance between edges and centre of the true edge is minimum. He was wanted to ensure that detector has only single response to an edge. Canny detection achieves more reliability and involves simple computation.

Task: Perform Canny Edge Detection over a given set of images.

II. Approach and Procedures

Canny Edge detector helps in finding different range of edges in an image and it is a multiple-stage algorithm. In this first we apply Gaussian filter to remove the noise and then we perform Sobel edge detection followed by Non- maximum suppression. We have two thresholds in Canny edge detection – high and low thresholds. By choosing 2 thresholds, we have the better option of choosing the edge points. We do it as follows. If the point is above the maximum threshold, we classify it as edge point. If it is less than the minimum threshold, we classify it as non-edge point. If it is between the two and if it is connected to edge point, then we classify it as edge point otherwise it is classified as non- edge point.

Algorithm used to develop in OpenCV:

Step-1: Read the input .jpg image from file and convert it to a Mat data.
Step-2: Use the Gaussian Blur functionality of OpenCV to apply the Gaussian filter to the image.
Step-3: Use Canny function of OpenCV with different high and low thresholds to obtain the Canny Edge image.
Step-4: Write the Canny Edge image as .jpg to a file using imwrite() function of OpenCV.

III. Experimental Results

Figure-2.1: The boat image

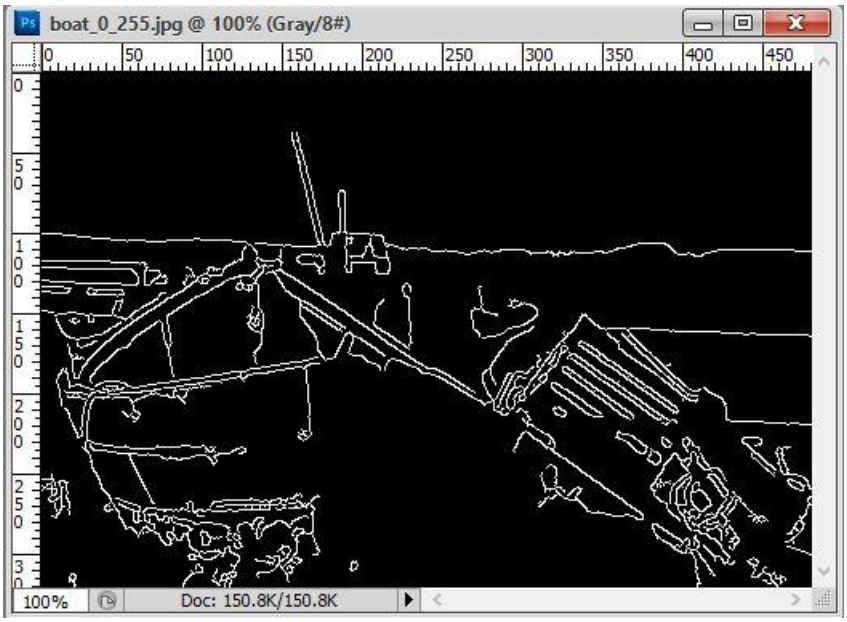

Figure-2.2: The Canny Edge boat image with low threshold = 0 and high threshold = 255 and sigma 1.5

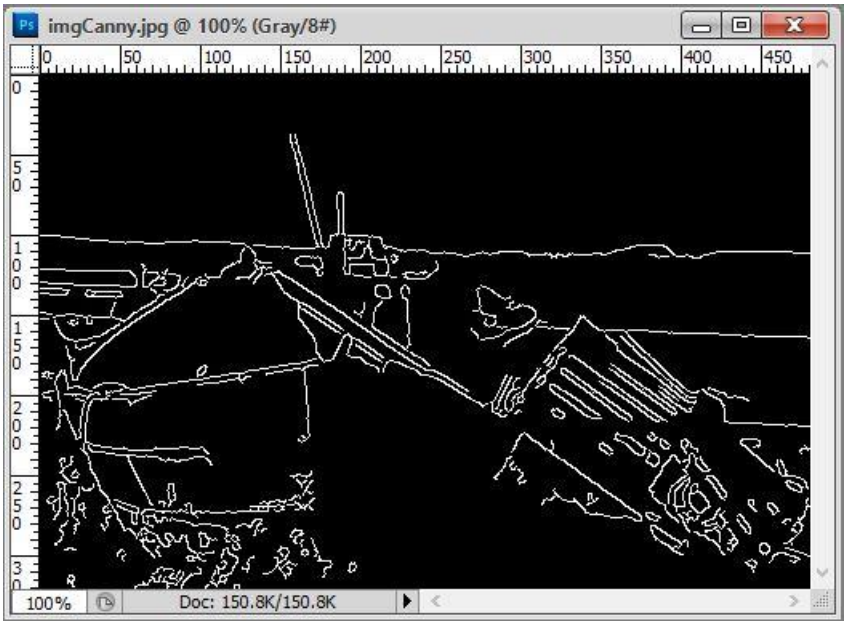

Figure-2.3: The Canny Edge boat image with low threshold = 85 and high threshold = 168 and sigma 1.5

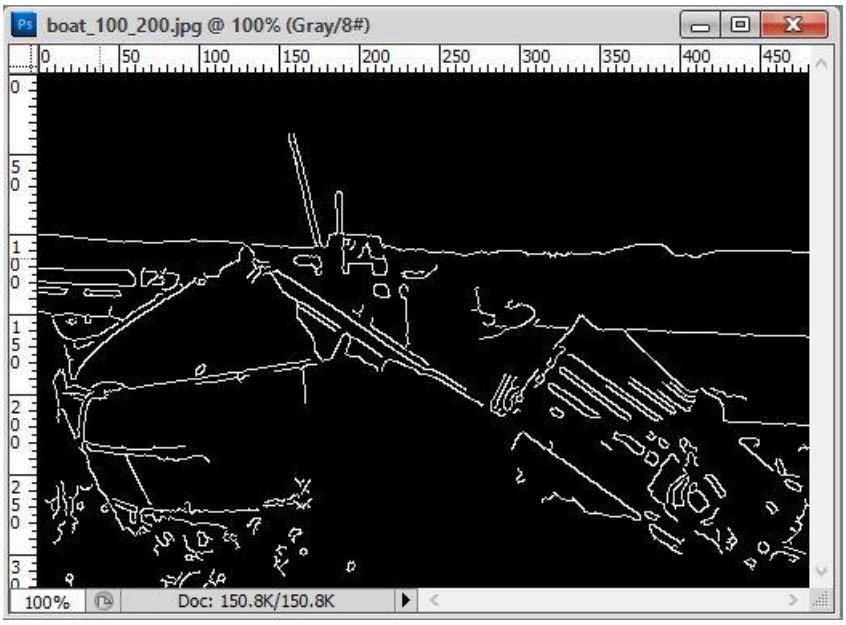

Figure-2.4: The Canny Edge boat image with low threshold = 100 and high threshold = 200 and sigma 1.5

Figure-2.5: The Canny Edge boat image with low threshold = 0 and high threshold = 255 and sigma 0.33

Figure-2.6: The Canny Edge boat image with low threshold = 85 and high threshold = 168 and sigma 0.33

Figure-2.7 The Canny Edge boat image with low threshold = 100 and high threshold = 200 and sigma 0.33

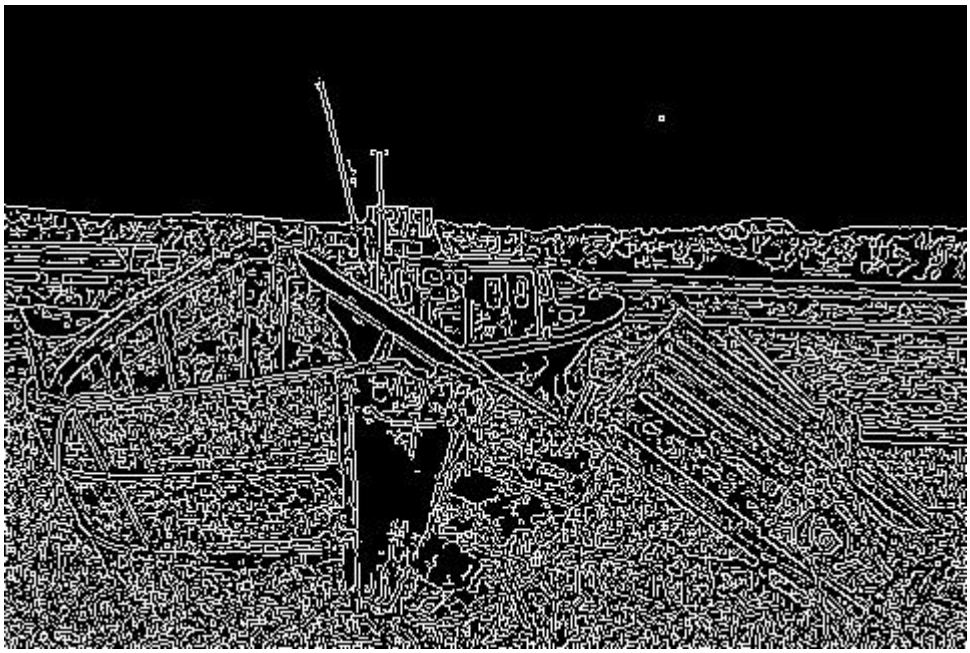

Figure-2.8: The Canny Edge boat image with low threshold = 10 and high threshold = 35 and sigma 0.33

Figure-2.9: The Canny Edge boat image with low threshold = 31 and high threshold = 212 and sigma 0.33

Figure-2.10: The Canny Edge boat image with low threshold = 55 and high threshold = 70 and sigma 0.33

Figure-2.11: The Canny Edge boat image with low threshold = 89 and high threshold = 143 and sigma 0.33

Figure-2.12: The Canny Edge boat image with low threshold = 156 and high threshold = 197 and sigma 0.33

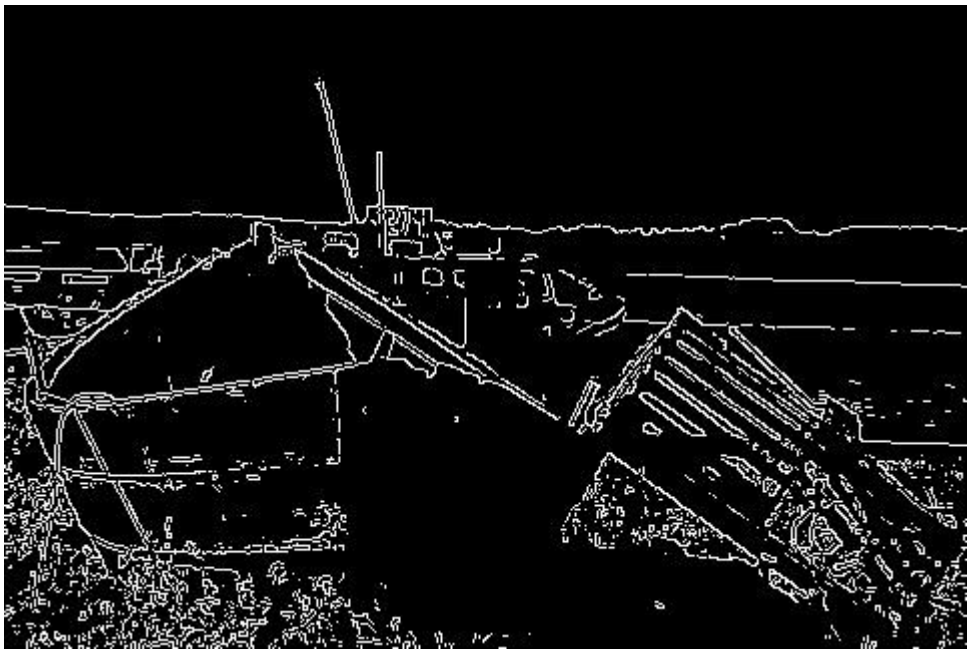

Figure-2.13: The Canny Edge boat image with low threshold = 234 and high threshold = 250 and sigma 0.33

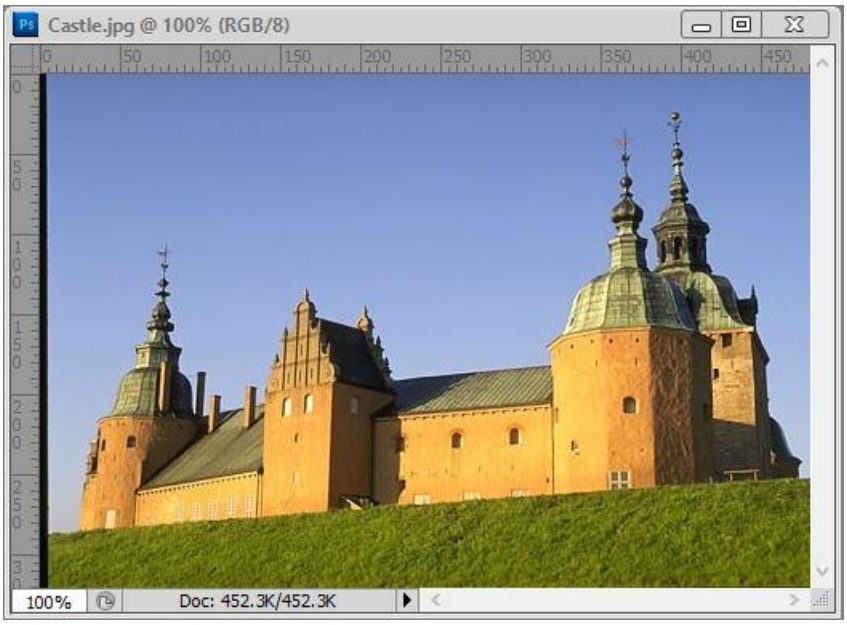

Figure-2.14: The castle image

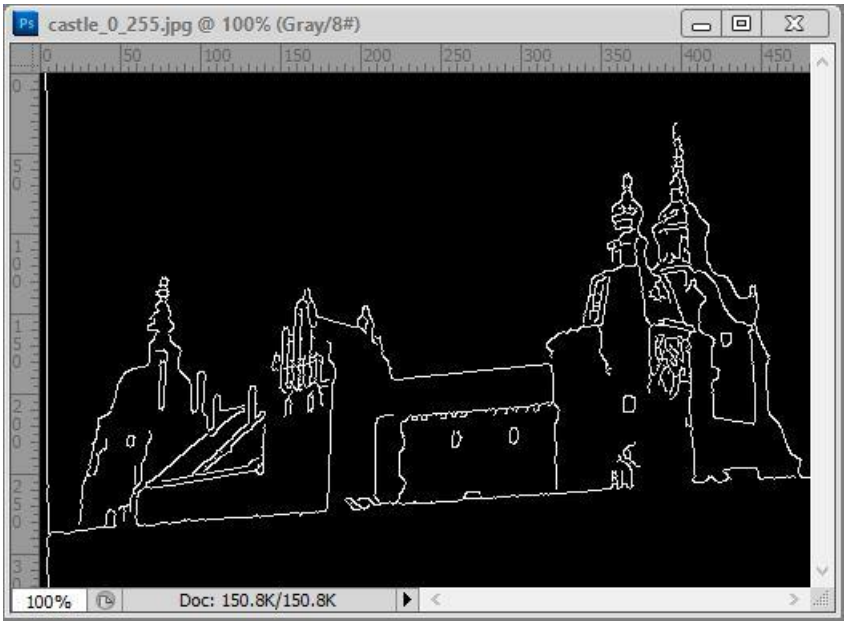

Figure-2.15: The Canny Edge castle image with low threshold = 0 and high threshold = 255 and sigma 1.5

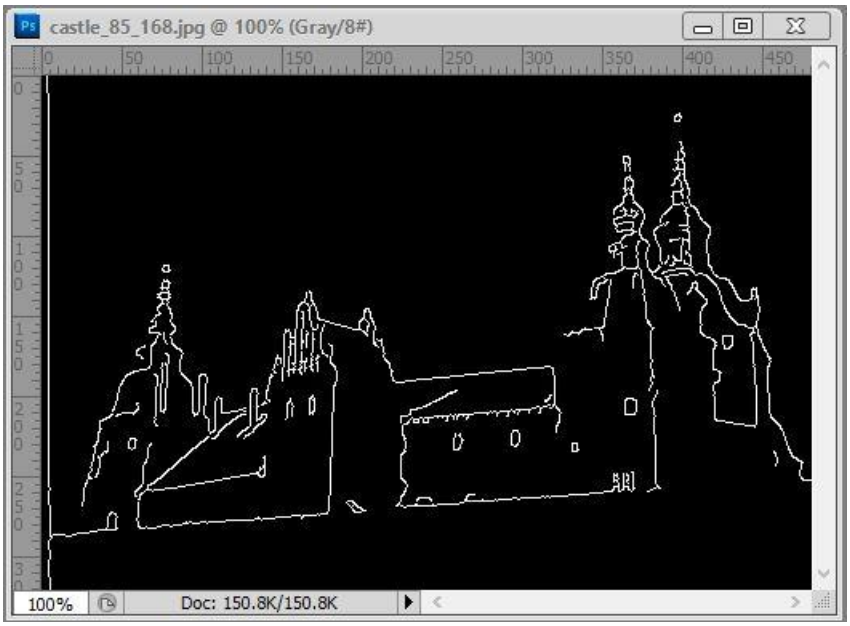

Figure-2.16: The Canny Edge castle image with low threshold = 85 and high threshold = 168 and sigma 1.5

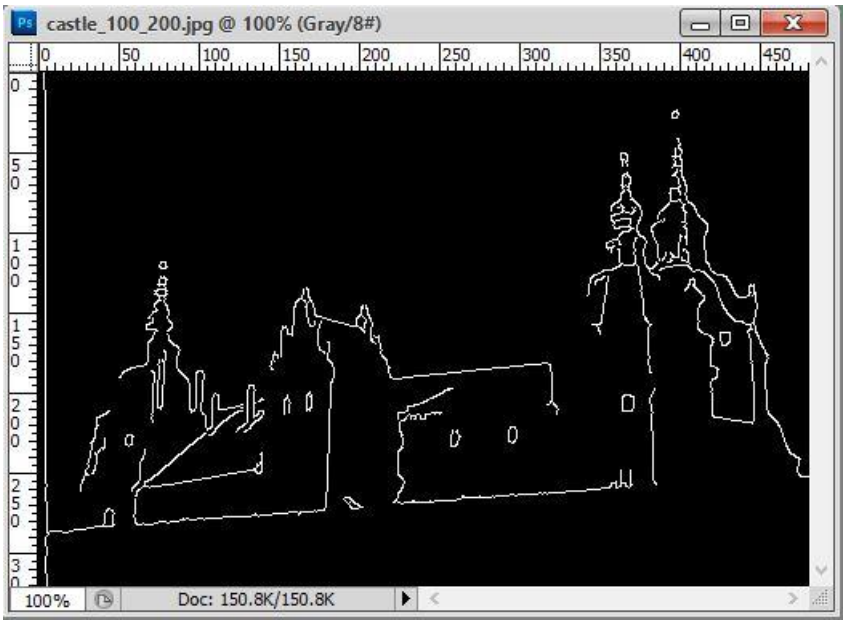

Figure-2.17: The Canny Edge castle image with low threshold = 100 and high threshold = 200 and sigma 1.5

Figure-2.18: The Canny Edge castle image with low threshold = 0 and high threshold = 255 and sigma 0.33

Figure-2.19: The Canny Edge castle image with low threshold = 85 and high threshold = 168 and sigma 0.33

Figure-2.20: The Canny Edge castle image with low threshold = 100 and high threshold = 200 and sigma 0.33

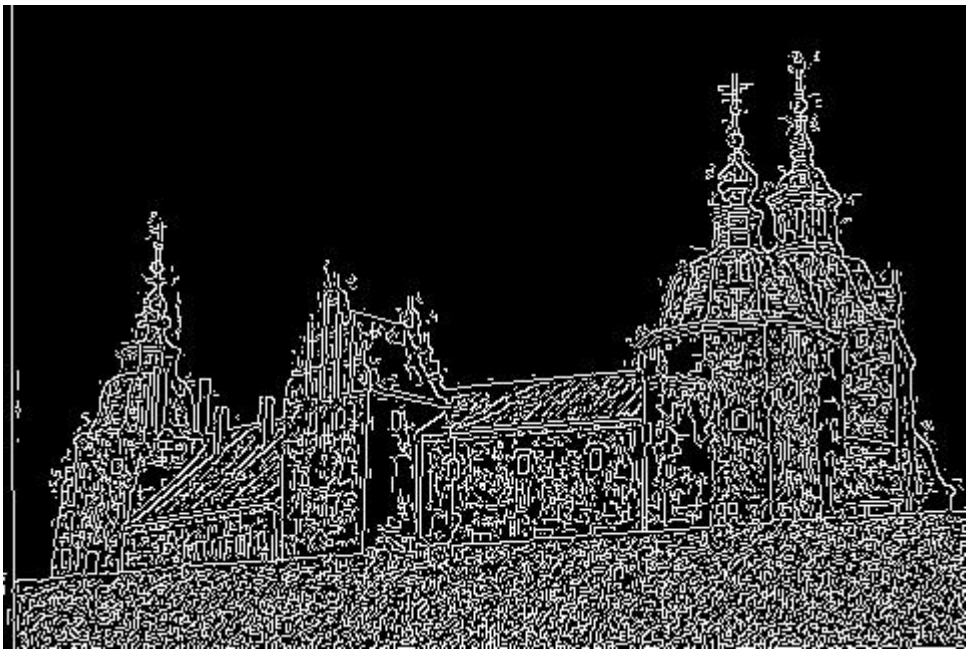

Figure-2.21: The Canny Edge castle image with low threshold = 10 and high threshold = 35 and sigma 0.33

Figure-2.22: The Canny Edge castle image with low threshold = 31 and high threshold = 212 and sigma 0.33

Figure-2.23: The Canny Edge castle image with low threshold = 55 and high threshold = 70 and sigma 0.33

Figure-2.24: The Canny Edge castle image with low threshold = 89 and high threshold = 143 and sigma 0.33

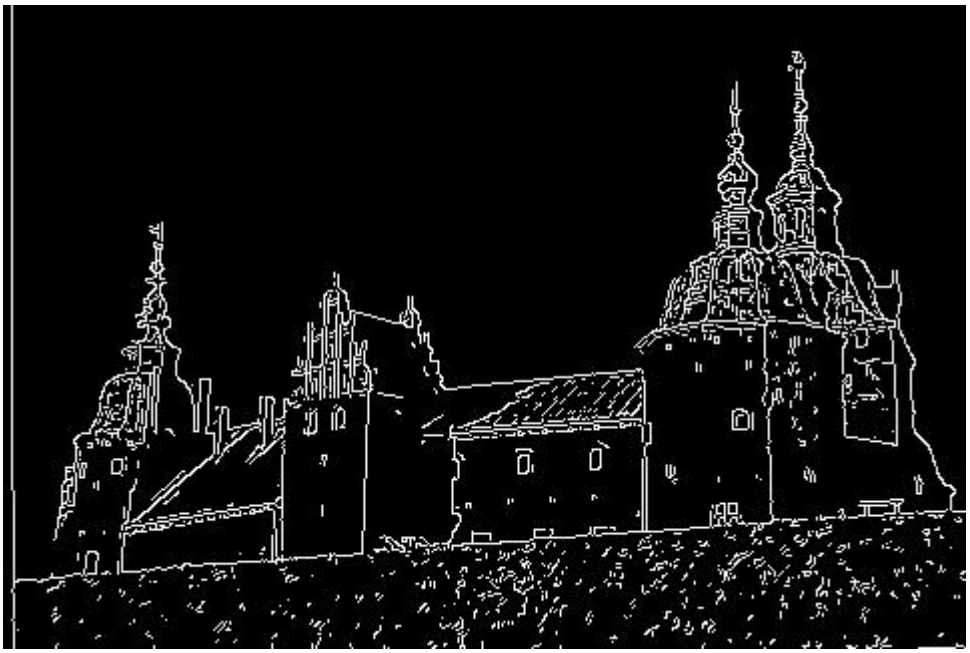

Figure-2.25: The Canny Edge castle image with low threshold = 156 and high threshold = 197 and sigma 0.33

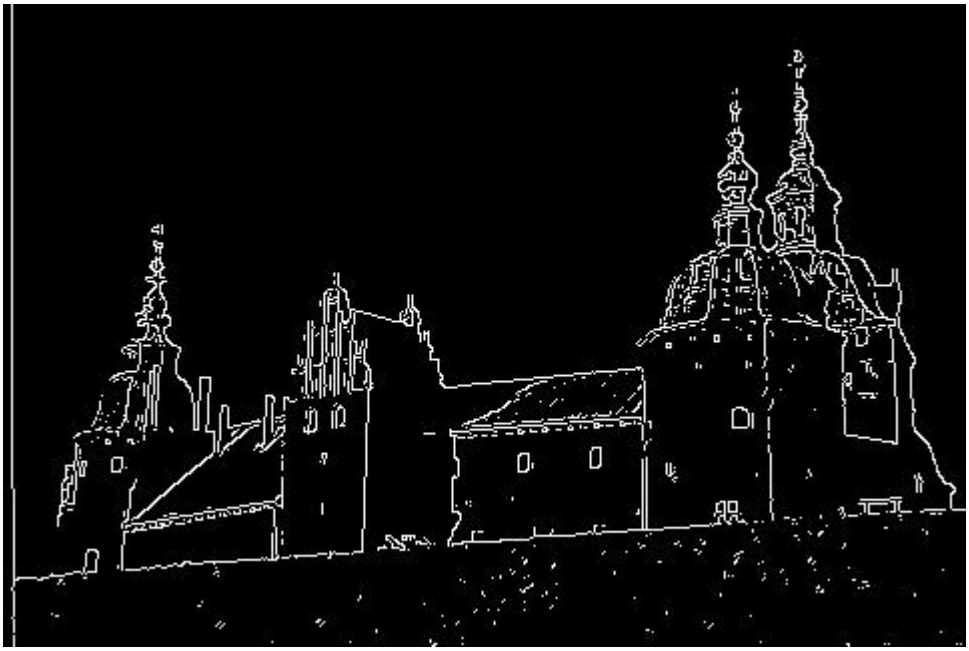

Figure-2.26: The Canny Edge castle image with low threshold = 234 and high threshold = 250 and sigma 0.33

IV. Discussion

1. Various thresholding inputs were given to the Canny Edge detector for both the images. The edge maps for low threshold and high threshold combinations are shown in the figures given above. On observing the best detection can be seen in the Castle image. Because of very bright texture towards the front of the boat, the edges across that region of the boat did not give a very good edge detection. The more the value of sigma, the poorer is the intricate edge detection. It is important to note that Canny edge detector cannot differentiate if the edge is from the object or texture of an object. Changing the parameters of the Canny detector, one cannot avoid the edge detected inside the object. Canny edge detector doesn't possess this layer of intelligence that prevents it from detecting uninformative edges inside the object. However, in this case for the shape and location of the castle can be differentiated from the edge map as the object but the boat cannot be because the boat gets camouflaged with the background and objects around it in the edge map. Sigma 0.33 gives a very good edge map and this value of sigma is a globally accepted one for good edge detection.
2. Yes, we can tune the sigma parameter to increase or decrease the accuracy of edge detection. Separating out the object might be something that we get once in a while on tuning sigma but yes, it's possible. The smaller the value of sigma, the better are the intricate detailed edges seen in the image. This has been proved above for sigma 0.33 and 1.5.
3. We have two thresholds in Canny edge detection – high and low thresholds. By choosing 2 thresholds, we have the better option of choosing the edge points. We do it as follows. If the point is above the maximum threshold, we classify it as edge point. If it is less than the minimum threshold, we classify it as non-edge point. If it is between the two and if it is connected to edge point, then we classify it as edge point otherwise it is classified as non- edge point. The gradient magnitude with values above the upper threshold are considered as strong edges. These edges are detected by the Canny edge detector. The gradient magnitudes that are less than the lower thresholds are rejected whereas the magnitudes between the lower and upper thresholds are taken only if they encounter a strong edge. So, by setting appropriate thresholds, we can filter out noisy edges and retain weak edges that are assumed to form a closed surface.

b. Contour Detection with Structured Edge

I. Abstract and Motivation

Contour Detection and Structured Edge are an integral part of image processing and computer vision logics. They are widely used for detecting fine edges. Structured forest is a state of the art algorithm that is used to compute the edges on a given image. Structured forest is a higher-level edge detection algorithm which is used for finely distinguishing between object and background. Object differentiation from background can very efficiently be done using this method. This edge detector considers a patch around the pixel and determines the likelihood of the pixel being an edge point or not. Edge patches are classified into sketch tokens using random forest classifiers. Structured learning is utilized to address problems associated with training a mapping function where the input or output space may be complex.

The Structured Edge Detection Algorithm is as under:

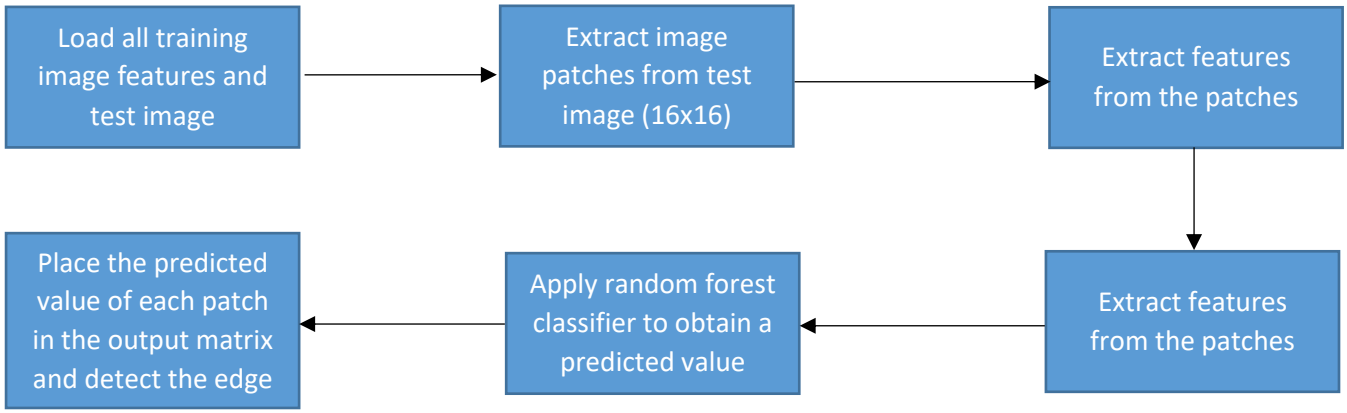

Figure-2.27: The flowchart for Structured Edge Algorithm

Initially the random forest is trained by taking a patch from the input and ground truth images and forming a structured output of whether a certain combination should be classified as an edge or not. After the random forest is trained by an appreciable number of samples, it can be tested. In the testing phase, only the test image is given for 16 segmentation and due to the speed of random forests, it computes the segmented output.

A random forest is a neat approach and very efficient. They run on the divide and conquer logic that's the heart that helps to improve their performance. The main idea of ensemble approach is to put a set of weak learners to form a strong learner. The basic element of a random forest is a decision tree. The decision trees are weak learners that are put together to form a random forest.

A decision tree classifies an input that belongs to space A as an output that belongs to a space B. The input or output space can be complex in nature. For example, we can have a histogram as the output. In a decision tree, an input is entered at the root node. The input then traverses down the tree in a recursive manner till it reaches the leaf node. Each node in the decision tree has a binary split function with some parameters. This split function decides whether the test sample should progress towards the right child node or the left child node. Often, the split function is complex.

A set of such trees are trained independently to find the parameters in the split function that result in a good split of data. Splitting parameters are chosen to maximize the information gain.

Task: Structured Edge Detection over a given set of images.

II. Approach and Procedures

Algorithm used to develop in MATLAB:

Online available code was used for performing structured edge detection as asked in the question from https://github.com/pdollar/edges. From the list of files, edgesDemo.m file was used to perform the Structured Edge Detection algorithm. The logic used in the MATLAB code is as under:

Step-1: Set the training parameters and the model parameters for an image dataset BSDS500.
Step-2: Obtain the labels for these images.
Step-3: Extract lot of P patches.
Step-4: These patches are the training data.
Step-5: Compute gradient and color to extract features from the patches.
Step-6: Finally train the structured random forest.
Step-7: Then perform the edge detection by extracting patches from the image to be tested on.
Step-8: Then extract features from patches and applying classifiers (structured random forest).
Step-9: Place the predicted value into the output image to get a probability edge map.
Step-10: Set the threshold to the probability edge map to obtain the binary edge map.

III. Experimental Results

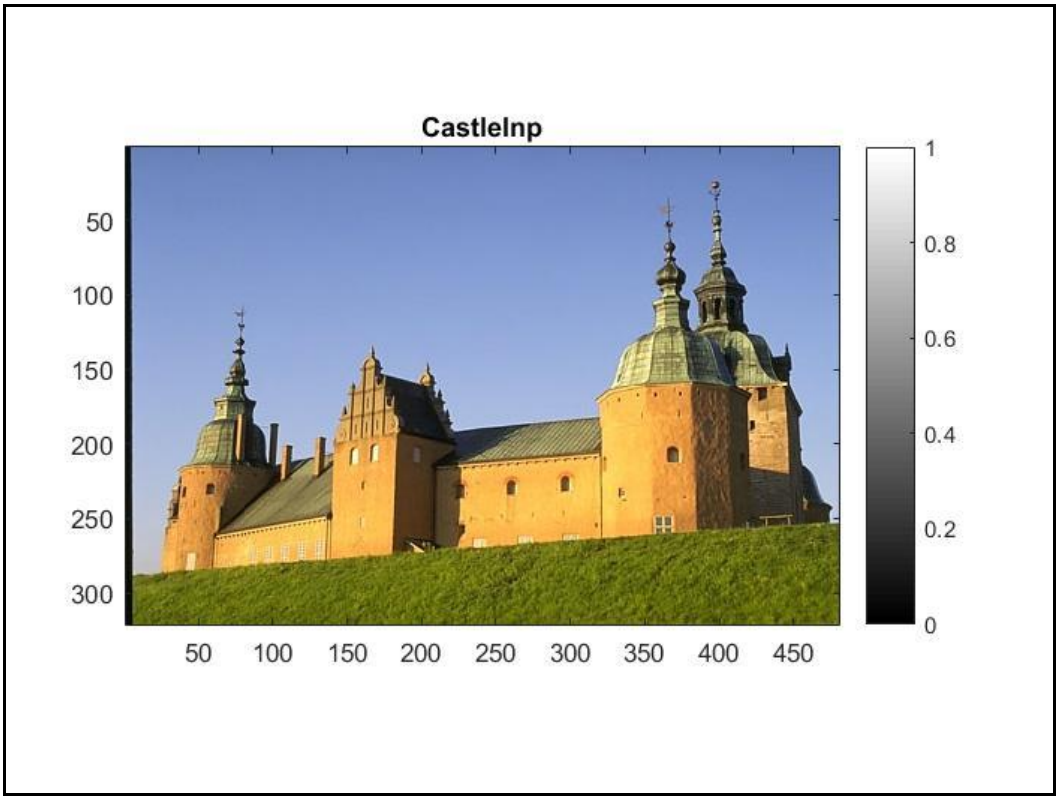

Figure-2.28: The Castle Image

Figure-2.29: Probability Map of Castle Image

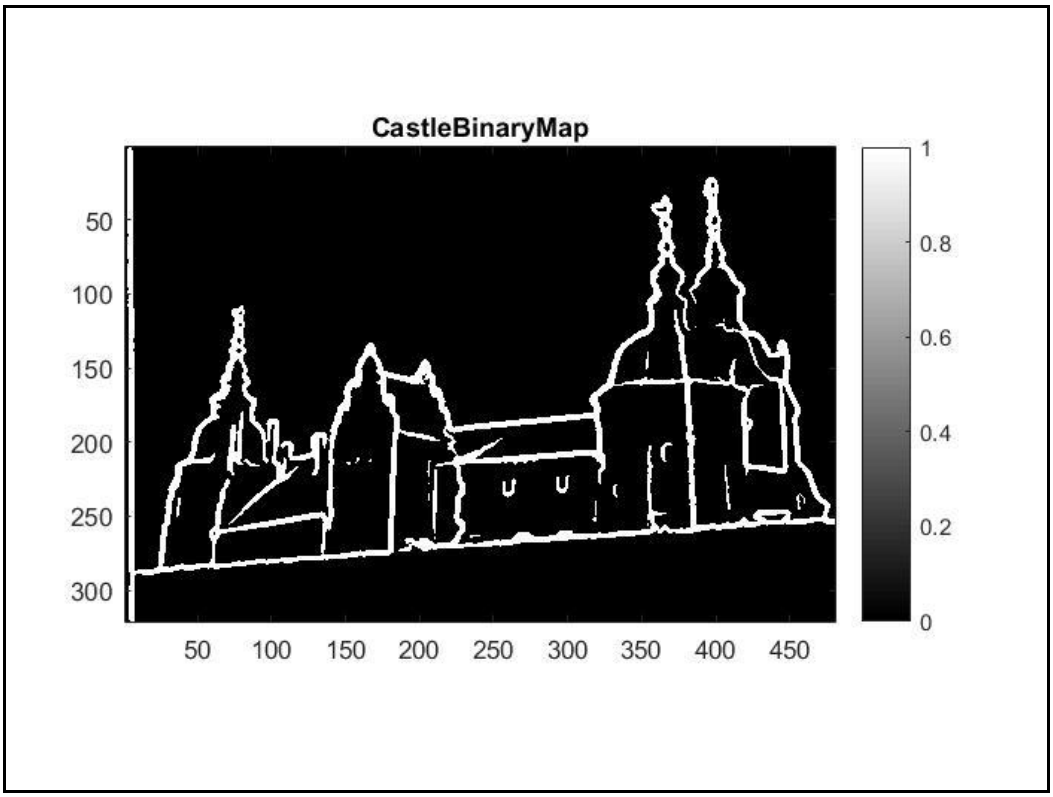

Figure-2.30: Binary Map of Castle Image (Probability Map > 0.2)

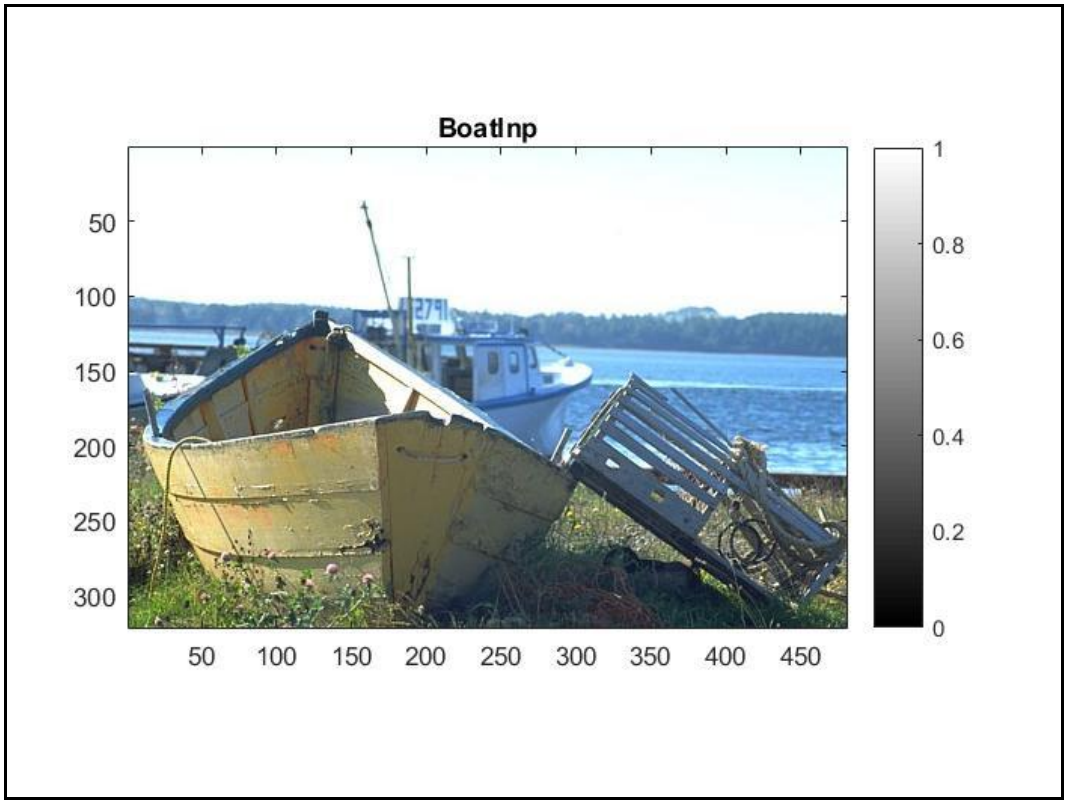

Figure-2.31: The Boat Image

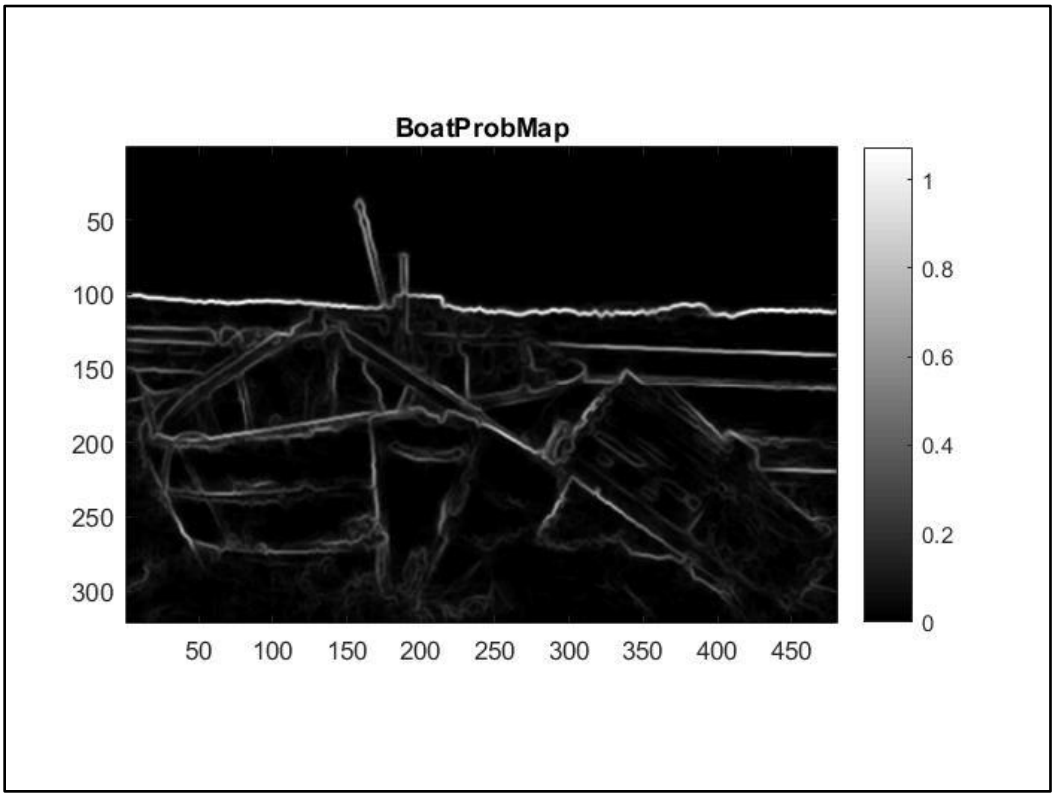

Figure-2.32: Probability Map of Boat Image

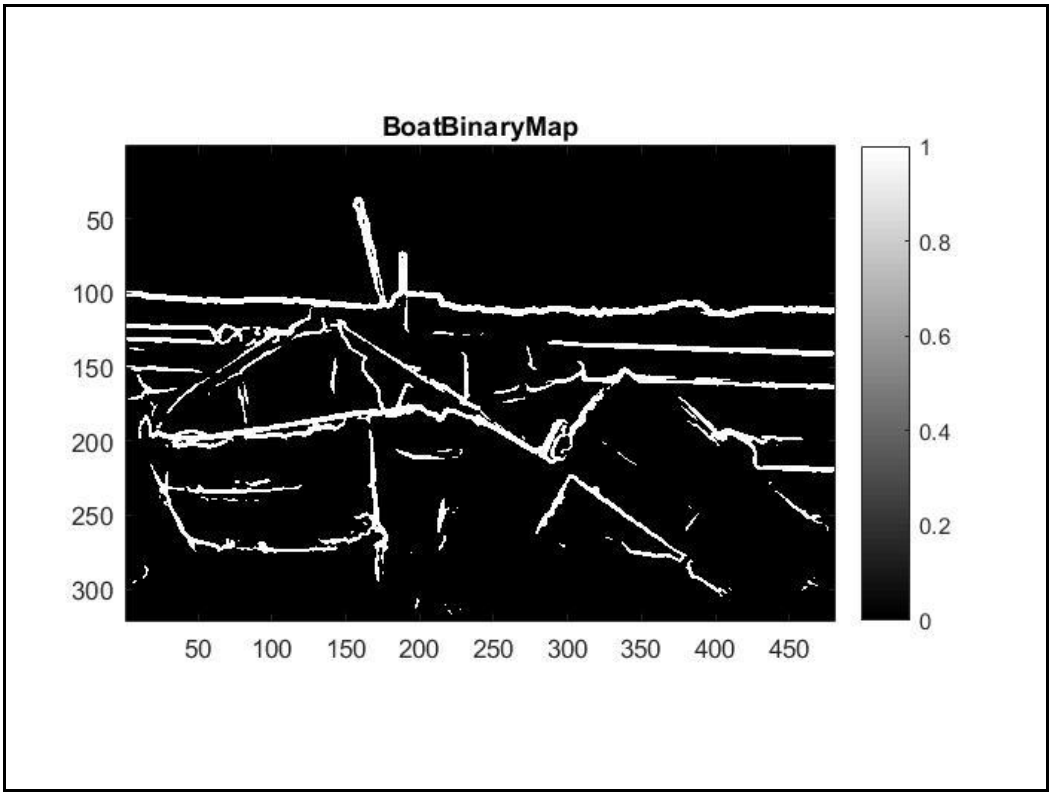

Figure-2.33: Binary Map of Boat Image (Probability Map > 0.2)

IV. Discussion

1. The Structured Edge Detector explanation along with the flowchart has been done above in the abstract and motivation section.
2. The process of decision tree construction and principle of RF classifier has been explained in the abstract and motivation section.
3. For Castle and Boat images, the threshold set to probability map was set to 0.2 for getting a binary map accordingly. The parameters of the model used are:

```
%% set detection parameters (can set after training)
model.opts.multiscale=0;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=4;            % max number threads for evaluation
model.opts.nms=0;                 % set to true to enable nms
```

The structed binary edge map was way thicker resulting coverage of less details of the edges as compared to Canny edge detector that gave a very fine edge detection which can be seen in the figures above. If we look at the probability edge map on the other hand, for structured edge we can differentiate between objects based on the probability edge map which is not something that is possible through a Canny edge detector output.

c. Performance Evaluation

I. Approach and Procedures

We can measure the performance of an edge detector using the recall and precision concepts. The F score is what gives the final evaluation metrics. These are generally used in machine learning applications to evaluate classifiers. Basically, the edges detected by the edge detection algorithms can never be perfectly correct compared to the ones detected by the

humans. The edges detected by humans are called ground truths. But we need to have multiple ground truths because everyone might not have the same perception for an edge. It changes from person to person. So, we take mean of the performance measure w.r.t. each ground truth. We compute mean of precision and recall. It is stated as under:

$$Precision, P = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad \rightarrow (7)$$

$$Recall, R = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad \rightarrow (8)$$

where,
True Positive → Edge pixel detected through algorithms (Canny, SE, etc) will map with ground truth
False Positive → Edge pixels in edge map correspond to non-edge pixels in ground truth
False Negative → Non-edge pixels in edge map correspond to true edge pixels in ground truth

We need not take true negative into consideration because it does not contribute to the performance. Therefore, we have a performance evaluation matrix called F-measure given by:

$$F = 2 * \frac{P \times R}{P + R} \quad \rightarrow (9)$$

In MATLAB, performance evaluation is done by:

```
[thrs, cntR, sumR, cntP, sumP, V] = edgesEvalImg(BinaryMap, <ground truth mat file>);
```

where,
thrs → the threshold values
P → cntP./sumP
R → cntR./sumR
F → as per eqn-9 above

II. Experimental Results

Table-2.1: Table for P, R, F for Castle Image with threshold 0.20

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.4827	0.9524	0.6407
Ground Truth 2	0.4801	0.9607	0.6410
Ground Truth 3	0.6848	0.9015	0.7783
Ground Truth 4	0.7915	0.8543	0.8217
Ground Truth 5	0.6709	0.9105	0.7725
Ground Truth 6	0.7535	0.8720	0.8084
Mean	**0.6439**	**0.9085**	**0.7437**

Table-2.2: Table for P, R, F for Castle Image with threshold 0.19

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.4769	0.9584	0.6369
Ground Truth 2	0.4755	0.9673	0.6376
Ground Truth 3	0.6794	0.9109	0.7783
Ground Truth 4	0.7888	0.8671	0.8261
Ground Truth 5	0.6666	0.9214	0.7735
Ground Truth 6	0.7480	0.8816	0.8093
Mean	**0.6392**	**0.9177**	**0.7436**

Table-2.3: Table for P, R, F for Castle Image with threshold 0.30

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.5445	0.9075	0.6806
Ground Truth 2	0.5445	0.9186	0.6837
Ground Truth 3	0.7122	0.7920	0.7500
Ground Truth 4	0.8231	0.7504	0.7851
Ground Truth 5	0.6994	0.8017	0.7471
Ground Truth 6	0.7729	0.7555	0.7641
Mean	**0.6827**	**0.8209**	**0.7351**

Table-2.4: Table for P, R, F for Boat Image with threshold 0.20

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.5634	0.4472	0.4986
Ground Truth 2	0.5187	0.6780	0.5877
Ground Truth 3	0.5771	0.4457	0.5029
Ground Truth 4	0.5894	0.6475	0.6171
Ground Truth 5	0.4495	0.6652	0.5365
Ground Truth 6	0.5858	0.6320	0.6080
Mean	**0.5473**	**0.5859**	**0.5595**

Table-2.5: Table for P, R, F for Boat Image with threshold 0.19

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.5518	0.4537	0.4980
Ground Truth 2	0.5041	0.6827	0.5800
Ground Truth 3	0.5658	0.4527	0.5029
Ground Truth 4	0.5745	0.6538	0.6116
Ground Truth 5	0.4419	0.6774	0.5349
Ground Truth 6	0.5743	0.6417	0.6061
Mean	**0.5362**	**0.5936**	**0.5555**

Table-2.6: Table for P, R, F for Boat Image with threshold 0.30

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.6534	0.3277	0.4362
Ground Truth 2	0.6225	0.5149	0.5636
Ground Truth 3	0.6597	0.3224	0.4331
Ground Truth 4	0.6979	0.4851	0.5723
Ground Truth 5	0.5249	0.4915	0.5077
Ground Truth 6	0.6801	0.4642	0.5518
Mean	**0.6397**	**0.4343**	**0.5107**

III. Discussion

Table-2.7: Table for P, R, F for Castle Image Canny Edge with low threshold = 50 and high threshold = 70 at sigma = 0.33:

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.0429	0.2721	0.0742
Ground Truth 2	0.0338	0.2356	0.0591
Ground Truth 3	0.0466	0.1677	0.0729

Ground Truth 4	0.0856	0.2470	0.1271
Ground Truth 5	0.0582	0.2386	0.0935
Ground Truth 6	0.0851	0.2734	0.1297
Mean	**0.0587**	**0.2391**	**0.0928**

Table-2.8: Table for P, R, F for Castle Image Canny Edge with low threshold = 31 and high threshold = 212 at sigma = 0.33:

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.1055	0.1381	0.1196
Ground Truth 2	0.0698	0.0952	0.0806
Ground Truth 3	0.0832	0.0648	0.0729
Ground Truth 4	0.1022	0.0700	0.0831
Ground Truth 5	0.1080	0.0923	0.0996
Ground Truth 6	0.1447	0.0995	0.1176
Mean	**0.1022**	**0.0933**	**0.0956**

Table-2.9: Table for P, R, F for Boat Image Canny Edge with low threshold = 50 and high threshold = 70 at sigma = 0.33:

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.0736	0.2691	0.1156
Ground Truth 2	0.0316	0.2180	0.0551
Ground Truth 3	0.0670	0.2444	0.1052
Ground Truth 4	0.0431	0.2410	0.0731
Ground Truth 5	0.0197	0.1579	0.0351
Ground Truth 6	0.0254	0.1408	0.0431
Mean	**0.0434**	**0.2119**	**0.0712**

Table-2.10: Table for P, R, F for Boat Image Canny Edge with low threshold = 31 and high threshold = 212 at sigma = 0.33:

Ground Truth Image	Precision (P)	Recall (R)	F-measure (F)
Ground Truth 1	0.1066	0.1450	0.1229
Ground Truth 2	0.0286	0.0794	0.0421
Ground Truth 3	0.1036	0.1385	0.1185
Ground Truth 4	0.0681	0.1373	0.0910
Ground Truth 5	0.0231	0.0716	0.0350
Ground Truth 6	0.0283	0.0596	0.0384
Mean	**0.0597**	**0.1053**	**0.0746**

1. The experimental results above give the required tabulation of the various Precision, Recall and F-Score at different thresholds. In F-measure, we evaluate the performance of edge maps based on ground truth (GT) developed by people. Based on the F-measure table values given in above section and the Canny edge ones just above, we can see that Structured edge has more F-Measure value compared to Canny Edge detection. So, this conveys that structured edge detection beats canny edge detection in terms of F-measure and visual appearance too.
2. The F-measure is image dependent. The Castle image will have more F-Measure than the Boat image because of the less background cluttering in the Castle image. In Boat image, apart from the boat there are lot of other objects surrounding it that occludes the shape of the boat. But for castle image, the object is distinctively separated because of less number of objects present in the background.

3. The F-Measure is a measure of test accuracy in statistics. It value is directly affected by both precision and recall. If any one of these metrics is low, then it results in a bad F-Measure. Additionally, it is interesting to note that there exists an inverse relation between precision and recall. If recall increases, then precision gets low and vice versa. To maximize the product of the two which follows an inverse relation, we need to have them to be equal. To prove this statement, the following steps are followed:
   Given,
   P+R = Constant (C)
   ⇨ P = R - C
   Now, F = 2 * P * R / (P+R)
   ⇨ F = 2 * (R - C) * R / C
   Maximizing F i.e.
   Performing derivative of F w.r.t. R, we get –
   ⇨ (2 * 2 * R - 2 * C) / C = 0 9equate to zero as it is a maximization problem)
   ⇨ 4 * R = 2 * C
   ⇨ R = C / 2
   ⇨ P = C / 2
   ⇨ P = R
   ⇨ Precision = Recall
   ⇨ **F-Measure reaches maximum when Precision is equal to Recall → (PROVED)**

Problem-3: Salient Point Descriptors and Image Matching

a. Extraction and Description of Salient Points

I. Abstract and Motivation

Salient point extraction is very much required in image processing and computer vision tasks specifically such as object recognition, video tracking, navigation, gesture recognition and image stitching, we need a means to extract meaningful features that can describe the image. These kinds of feature extraction requirements are where SIFT and SURF come into the picture. These extracted features are required to be scale invariant and rotation invariant to provide robustness. SIFT and SURF are such algorithms that extract important local features in an image that are scale and rotation invariant.

Task: In this problem, we implement SIFT and SURF algorithms to compare their performance and efficiency. We also utilize these features to perform object matching using Brute Force matching technique. It is not the best kind of matching that can be expected and its output is not the best but the detailed discussion on the same and its pros and cons are discussed below.

Object categorization and localization is sometime required to enhance an image. The biggest challenges in object classification are introduced due to problems such as different camera positions, illumination differences, internal parameters and variation within the object which has not been solved till date and is a big area of research that is still going on.

II. Approach and Procedures

Theoretical Approach:

- **SIFT (Scale Invariant Feature Transform)**

SIFT was proposed by David G Lowe. To implement this, OpenCV was used that contains SIFT detectors and extractors. The overview of SIFT algorithm is as under:

➔ Scale Space using Gaussian Kernel construction.
➔ Now, we compute DoG (Difference of Gaussian) – an approximation of LoG (Laplacian of Gaussian)

Given a Gaussian blurred image,

$$L(x, y, \sigma) = G(x, y, \sigma) \times I(x, y) \quad \rightarrow (10)$$

where,

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} exp^{\frac{-(x^2+y^2)}{\sigma^2}} \quad \rightarrow (11)$$

Now, we have difference of Gaussian blurred images at scales σ and kσ .

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \quad \rightarrow (12)$$

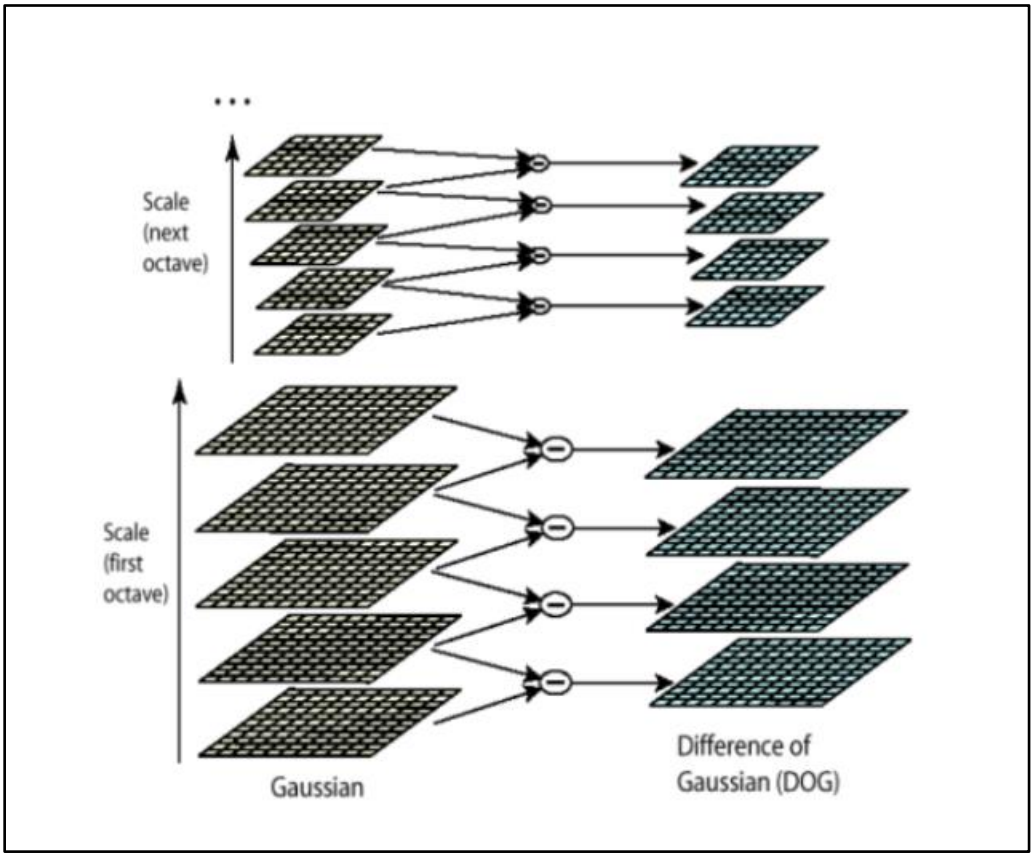

Figure-3.1: Blurred images at different scales and DoG computation (SIFT)

- ➔ We now must locate the extrema of DoG. This can be done by scanning through each DoG image and then identifying the maximum and the minimum.
- ➔ Taylor Series expansion to localize the sub-pixels needs to be used.
- ➔ Filtering out low contrast points using scale space values at previous locations and filter edge responses using Hessian Matrix.
- ➔ Assignment of key point orientations by creation of histogram of local gradient directions.
- ➔ Building the key point descriptors, wherein each key point has a location, orientation and scale.
- ➔ Find blurred image of nearest scale and then sample the points around the key point. Now, rotate the gradients & co-ordinates by the orientation computed previously.
- ➔ Divide the region into sub-regions and create a histogram for each sub-region with several bins.

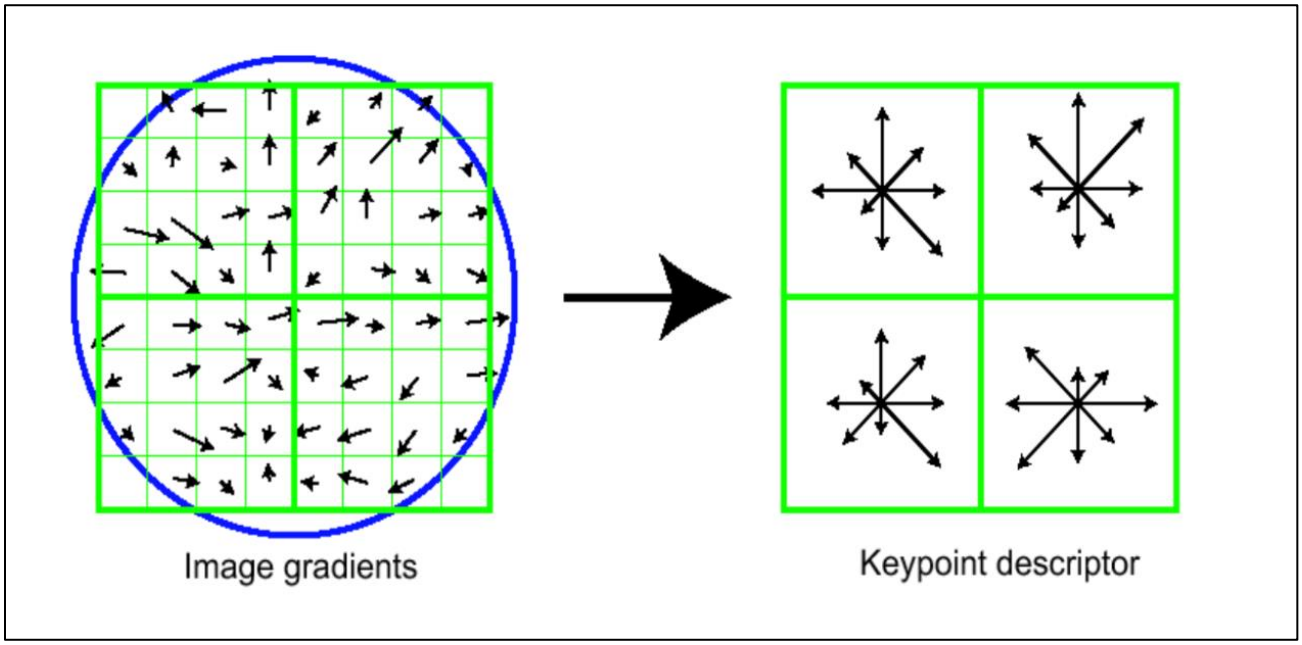

Figure-3.2: Gradient orientation and normalization tricks (SIFT)
Source: http://3.bp.blogspot.com/-2Lw3DxApZrw/VKBKMeAwTnI/AAAAAAAANyU/7IQxfszsclc/s1600/sift_pic.png

- ➔ Once we select the key point orientation, the feature descriptor is then calculated as a set of orientation histograms on 4 × 4 pixel neighbor hoods.
- ➔ The orientation data is obtained from the Gaussian image which is closest in scale to the key point's scale.
- ➔ Histograms contain 8 bins, and descriptors contains array of 4 histograms around the key point and this leads to a SIFT feature vector with (4 × 4 × 8 =) 128 elements. This vector is then normalized to nullify the changes in illumination.
- ➔ Using SIFT algorithm, we obtain scale and rotation invariance, view point variance by using scale space and also illumination changes.
  For programming this, we use OpenCV built-in class for SIFT as shown below in the algorithms section.

- **SURF (Speeded Up Robust Features):**

In computer vision, speeded up robust features (SURF) is a patented local feature detector and descriptor. It is widely used in the areas of computer vision and image processing. SURF is faster than SIFT.

To detect interest points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a precomputed integral image. SURF descriptors have been used to locate and recognize objects, people or faces, to reconstruct 3D scenes, to track objects and to extract points of interest. SURF was first presented by Herbert Bay, et al., at the 2006 European Conference on Computer Vision.

We create integral images in SURF because of the fast computation of mask convolutions is independent of the size of the filter. Integral image is of the same size of the image to be analyzed and the value of integral image at any location can be determined as the sum of the intensity values lesser than or equal to the location where we evaluate the integral image. Once the integral image is created, we follow the steps below:

- ➔ First step is finding the interest points in the image. We compute the Determinant of Hessian Matrix – (Product of Eigen Values of Hessian Matrix). We use LoG (second order derivative filter) as the derivative filter in the Hessian matrix. Now, using the integral image, we obtain the sum of intensities for squares present in Hessian box filters – multiplied by the weight factor and then addition of Net Resultant Sum (NRS). Evaluation of filter size is done using a threshold, wherein we control the number of interest points which are to be detected.

- Second Step is finding the major interest points. To obtain the main features, we use a 3 x 3 Non-maximal suppression below and above the scale space of each octave. Now, since the scale space is scaled into large and irregular shaped pieces, we need to interpolate the interest points to obtain a correct scale.
- Now, comes the feature direction. We find this using Haar Transform. We first, get the pixels available in a circular range radius of six times our scale space. Now, we calculate x and y Haar transform for each point. Choose a direction of max weight to obtain the direction of feature.
- Final step is generating the 64-dimensional feature vectors consisting of sum of dx, dy, $|dx|$, $|dy|$ for 16 sub-regions which are created from square description window. dx and dy are Haar wavelet responses in Horizontal & Vertical directions. These are computed in the direction of rotation. We now have the SURF descriptors (feature vectors).

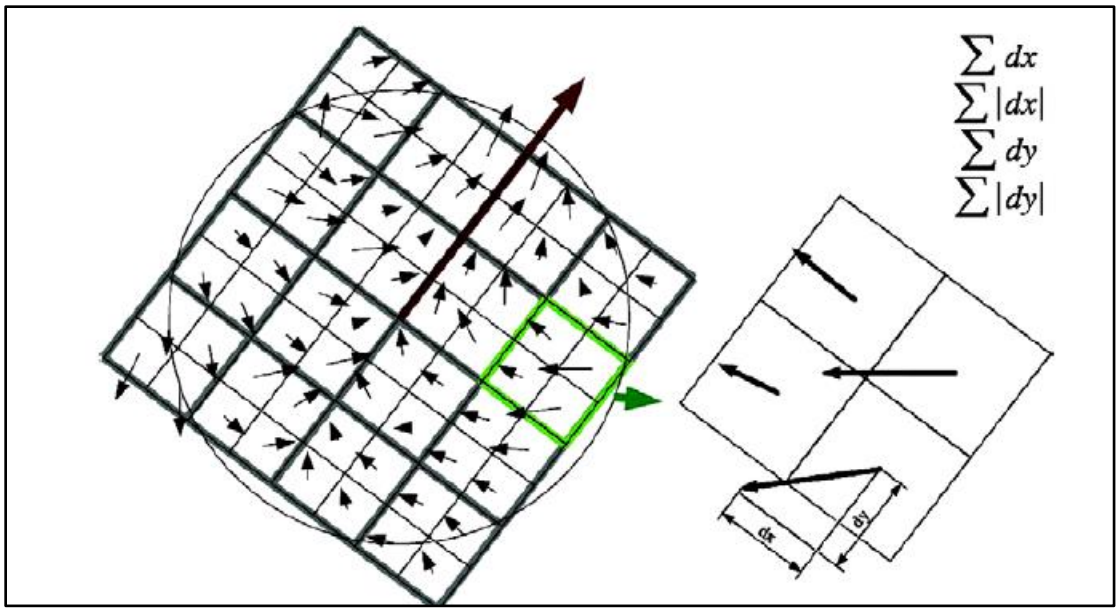

Figure-3.3: SURF descriptor
Source: https://courses.cs.washington.edu/courses/cse576/13sp/projects/project1/artifacts/woodrc/images/image014.jpg

Algorithm used to develop SIFT to extract salient points in OpenCV:
Step-1: Read the images of SUV and TRUCK.
Step-2: Detect the key points using SIFT detector.
Step-3: For the detection procedure, define vectors of key points for each image.
Step-4: Use the detector class of OpenCV to detect the key points in each image.
Step-5: Use the drawKeypoints() function of OpenCV to draw key points on each of the images and finally display the image with the key points overlaid on it.

Algorithm used to develop SURF to extract salient points in OpenCV:
Step-1: Read the images of SUV and TRUCK.
Step-2: Detect the key points using SURF detector.
Step-3: For the detection procedure, define vectors of key points for each image.
Step-4: Use the detector class of OpenCV to detect the key points in each image.
Step-5: Use the drawKeypoints() function of OpenCV to draw key points on each of the images and finally display the image with the key points overlaid on it.

III. Experimental Results

Figure-3.4: SUV SIFT feature points with Hessian parameter as 400

Figure-3.5: TRUCK SIFT feature points with Hessian parameter as 400

Figure-3.6: SUV SURF feature points with Hessian parameter as 400

Figure-3.7: SUV SURF feature points with Hessian parameter as 400

IV. Discussion

The discussion states the performance and efficiency of SIFT and SURF according to their weakness and strengths. Main advantage of SURF is that it is faster in computation than SIFT. Though here we have a smaller dataset, there is not much difference in computational times. It will make a huge impact with huge datasets. SIFT achieves strong features (number of features) compared to SURF. One of the biggest advantages of SIFT is that it is invariant to scaling, rotation, change in illumination, noise and small changes in the point of view. We can also say that SIFT provides more features with respect to bus parts such as name at the front bottom, windows and features of the environment which are not that important and not considered by SURF. SIFT is sometimes more used than SURF because of its ability to provide more features even though it is slow than SURF. In SIFT, key point angle is calculated within a square of neighbors and then we find different angles. In SURF, direction is taken as the Haar wavelet response with greatest normal within a sector. SIFT descriptors are

computed by 128-D gradient histogram. SURF descriptors record the Haar wavelet response with 64-D. One last thing to discuss is that, SURF performs better in illuminance change and in resisting blurring because its descriptors are calculated based on original image using is integral image, whereas, In SIFT its descriptor are computed based on Gaussian Pyramids.

b. Image Matching

I. Approach and Procedures

Theoretical Approach:

In the above section, we saw the importance of good features and extraction of SIFT and SURF features. For finding relevant matches among images is the underlying reason why we extract features from an image. Image matching is done by following a set of steps in addition to the steps above as mentioned below:

After SIFT or SURF feature extraction using OpenCV, we have used Brute Force matcher after using key points matching.

Algorithm used to develop in OpenCV for Image Matching SIFT:

Step-1: Read the images of all the combinations of images two at a time.

Step-2: Detect and compute the key points as well as the descriptors using SIFT detector and Hessian parameter set to 20.

Step-3: Use the Brute Force Matcher to match the key points and finally using the drawMatches() function of OpenCV the lines connecting the key points are drawn.

Step-4: Finally display the image with the key points and matching lines overlaid on it.

Algorithm used to develop in OpenCV for Image Matching SURF:

Step-1: Read the images of all the combinations of images two at a time .

Step-2: Detect and compute the key points as well as the descriptors using SURF detector and Hessian parameter as 10000.

Step-3: Use the Brute Force Matcher to match the key points and finally using the drawMatches() function of OpenCV the lines connecting the key points are drawn.

Step-4: Finally display the image with the key points and matching lines overlaid on it.

II. Experimental Results

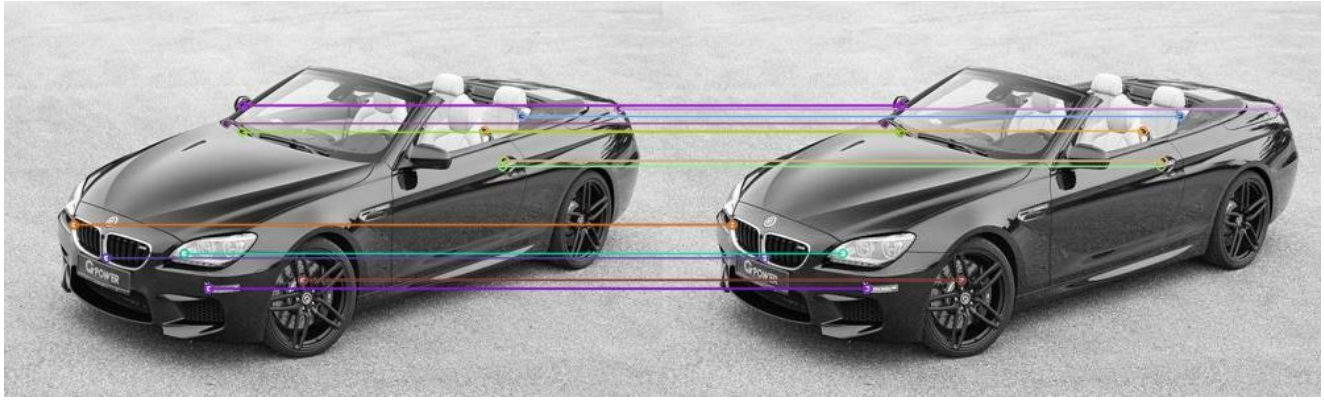

Figure-3.8: SIFT image matching with Hessian parameter as 20 (Convertible1 with Convertible1 image using Brute Force Matcher)

Figure-3.9: SIFT image matching with Hessian parameter as 20 (Convertible2 with Convertible2 image using Brute Force Matcher)

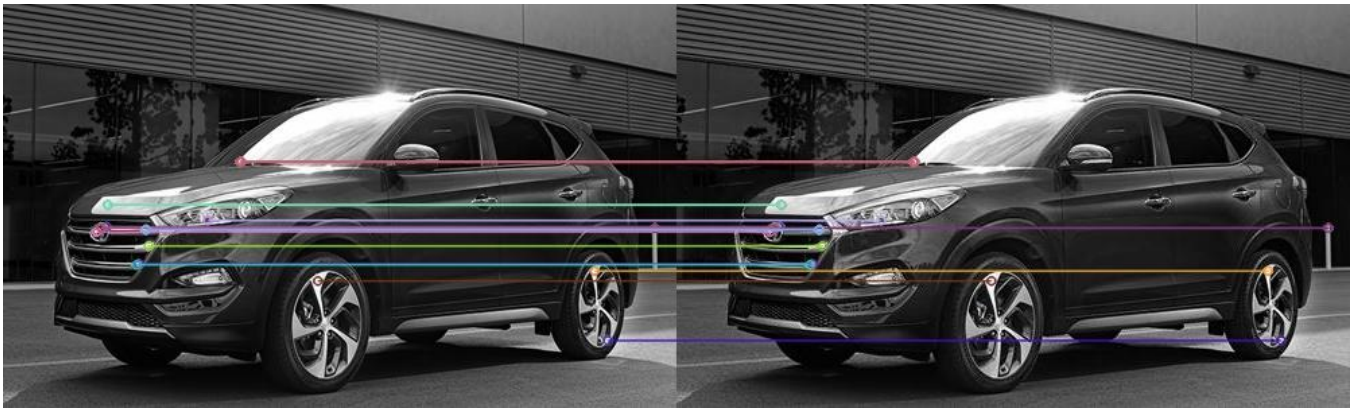

Figure-3.10: SIFT image matching with Hessian parameter as 20 (SUV with SUV image using Brute Force Matcher)

Figure-3.11: SIFT image matching with Hessian parameter as 20 (Truck with Truck image using Brute Force Matcher)

Figure-3.12: SIFT image matching with Hessian parameter as 20 (Convertibe1 with Convertible2 image using Brute Force Matcher)

Figure-3.13: SIFT image matching with Hessian parameter as 20 (Convertible1 with SUV image using Brute Force Matcher)

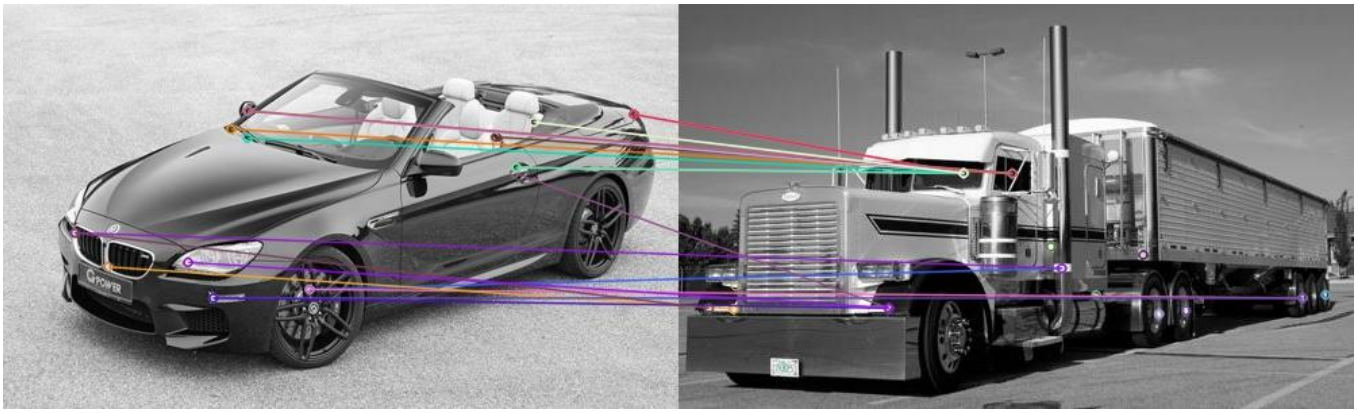

Figure-3.14: SIFT image matching with Hessian parameter as 20 (Convertible1 with Truck image using Brute Force Matcher)

Figure-3.15: SIFT image matching with Hessian parameter as 20 (Convertible2 with SUV image using Brute Force

Figure-3.16: SIFT image matching with Hessian parameter as 20 (Convertible2 with Truck image using Brute Force Matcher)

Figure-3.17: SIFT image matching with Hessian parameter as 20 (SUV with Truck image using Brute Force Matcher)

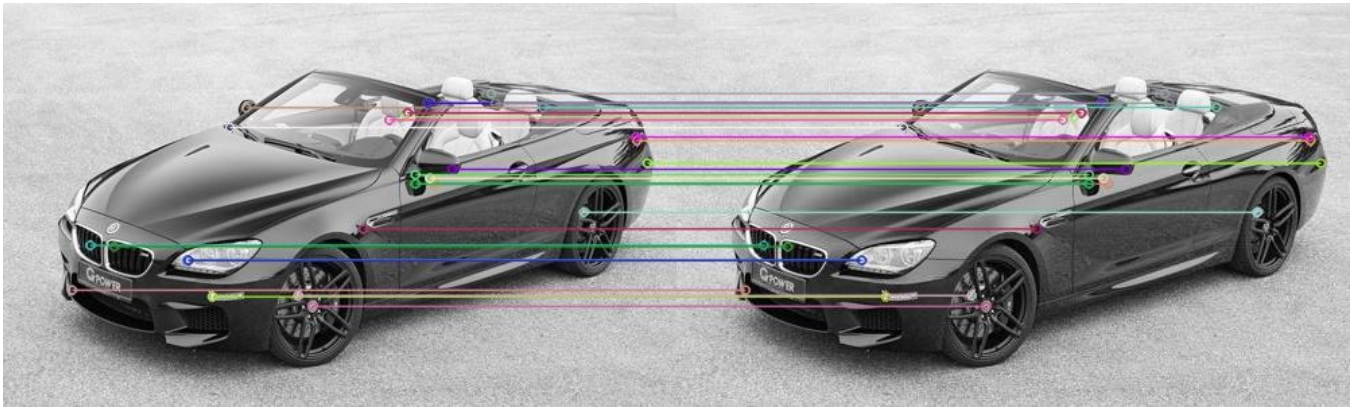

Figure-3.18: SURF image matching with Hessian parameter as 10000 (Convertible1 with Convertible1 image using Brute Force

Figure-3.19: SURF image matching with Hessian parameter as 10000 (Convertible2 with Convertible2 image using Brute Force

Figure-3.20: SURF image matching with Hessian parameter as 10000 (SUV with SUV image using Brute Force Matcher)

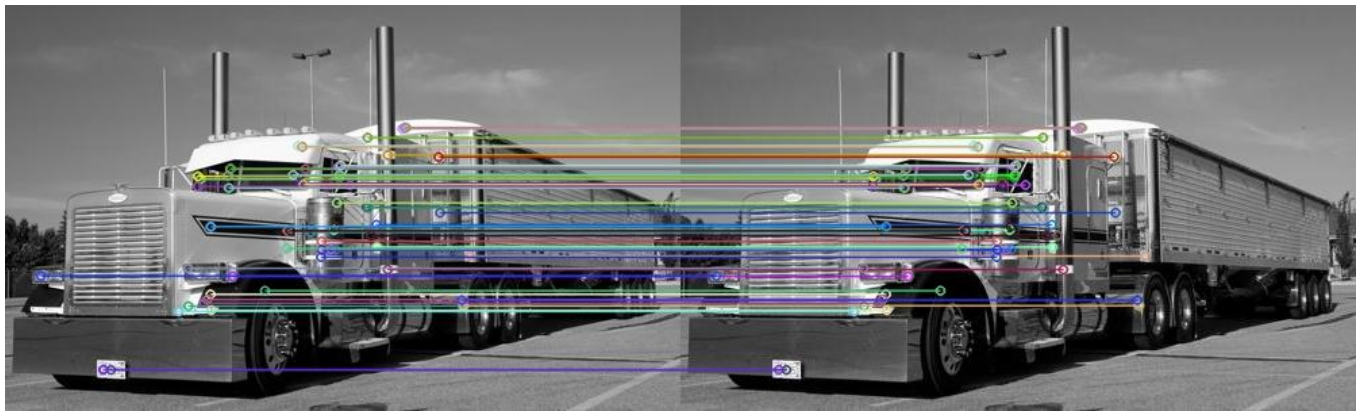

Figure-3.21: SURF image matching with Hessian parameter as 10000 (Truck with Truck image using Brute Force Matcher)

Figure-3.22: SURF image matching with Hessian parameter as 10000 (Convertible1 with Convertible2 image using Brute Force

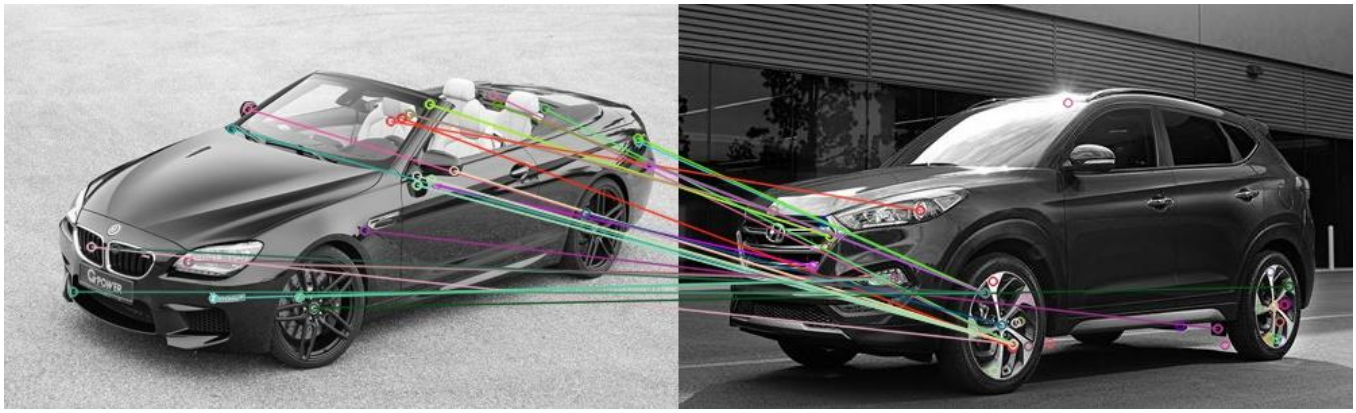

Figure-3.23: SURF image matching with Hessian parameter as 10000 (Convertible1 with SUV image using Brute Force Matcher)

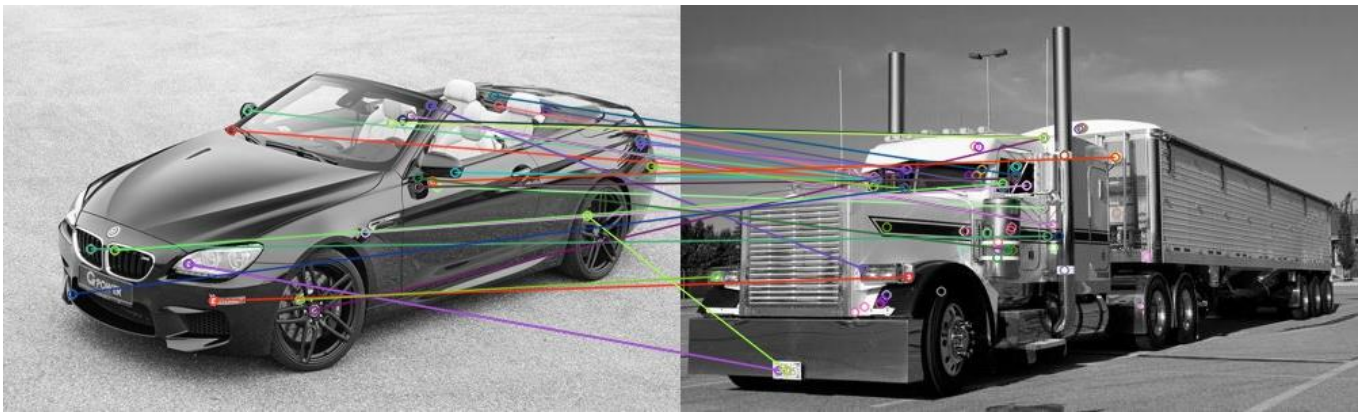

Figure-3.24: SURF image matching with Hessian parameter as 10000 (Convertible1 with Truck image using Brute Force Matcher)

Figure-3.25: SURF image matching with Hessian parameter as 10000 (Convertible2 with SUV image using Brute Force Matcher)

Figure-3.26: SURF image matching with Hessian parameter as 10000 (Convertible2 with Truck image using Brute Force Matcher)

Figure-3.27: SURF image matching with Hessian parameter as 10000 (SUV with Truck image using Brute Force Matcher)

III. Discussion

Discussing on the matching of the various images, the following conclusions can be drawn.

For SIFT and SURF, when we match the same image with each other then, the salient points match and 100% matching is observed with each key point of first image matches to the corresponding key point in the second image.

For SIFT and SURF, when we match different image combinations, there are a lot of correct features matched between them. There are obviously few features which are not matched correctly, but overall, we have a lot of correct feature matches.

For SIFT and SURF, when Convertible1 is matched with SUV, there are lots of features like points on the tyre cap of former matches with that of the latter but there are points where the formers key point matches to some point in the latter's background. When Convertible1 is matched with Truck, there are points like the tyre caps match, the windshields match but some points like the windshield of former matches with the number plate of latter which is a failed matching. So, overall there are cases of pass as well as failure.

There are a lot of factors that define these types of failures being observed which are listed as under:

- The dimensions of the objects being matched may be different due to which the key points may be quite apart which may lead to incorrect matching. One way out is to use bilinear interpolation to resize the image because the algorithm is invariant to scaling.
- The orientation of the images being matched may be different. If one is at some viewing angle and the other is at a different orientation, then there can be a mismatch in the key points matching.
- Brute Force matcher too could be a big issue. Since the Brute Force matcher computes a match for every point, noisy matches could be filtered out. To do this, we have calculated the maximum and minimum values of distance between each match. Later we only considered the matches that were less than twice the minimum value. Hence this way we can filter out matches that were not relevant. Again, this type of match filtering may give errors when an object is being compared with different objects. This is because the Brute force method tries to match irrelevant points and in doing so it sets up the minimum distance value. This can be solved by using a FLANN based matcher as it uses the nearest neighbor's technique.

c. Bag of Words

I. Abstract and Motivation

Bag of Words algorithm is based on the idea of document classification and OCR use it on a broader scale. It can be used for image classification in small systems wherein user defines the number of clusters. Bag of words is the vector of occurrence of count of words and is expressed as a histogram of each word. Training of Bag of Words is done to cluster the

training images into several clusters. The calculated features of test images can be compared to words in a dictionary to get histogram.

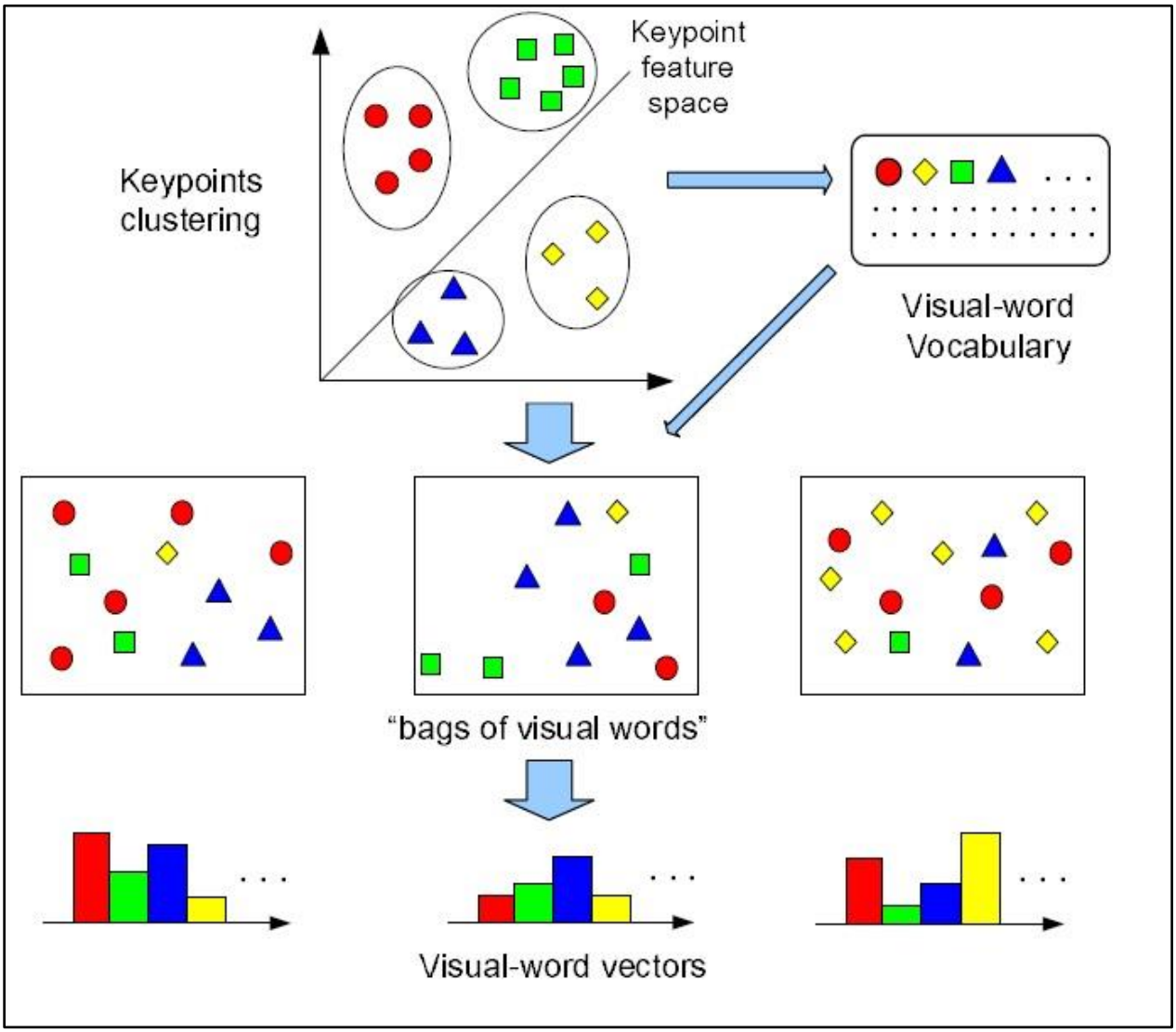

Figure-3.28: Bag of Words algorithm

II. Approach and Procedures

Theoretical Approach:

- ➔ We first, extract the local features of the image using SIFT.
- ➔ Placement of all the features into a single set.
- ➔ K- means clustering algorithm is applied over the set of the feature vectors we obtained in above step to find centroid co-ordinates and we assign a code word to each centroid obtained. This results in the construction of our vocabulary of length K.
- ➔ Finally, once we obtain feature vectors based on global identities, we classify them. Based on the size of training data we used K- nearest algorithm for smaller training data set.

Algorithm used to develop in OpenCV:

Step-1: Read the four images SUV, Truck, Convertible1 and Convertible2 and store it in a Mat data.
Step-2: Define a Hessian parameter with a value between 100 to 800.

Step-3: Detect and compute the key points as well as the descriptors using SIFT detector and Hessian parameter defined above.

Step-4: Create a BOWMeansTrainer for 8 bins which is the Bag of Words.

Step-5: Add SUV, Truck and Convertible1 descriptors to the Bag of Words and create a vocabulary / codebook and pass the codebook through K-means clustering to get the final vocabulary.

Step-6: The dimensions of the vocabulary would be now 8x128 and that of descriptors would be <Hessian parameter>x128.

Step-7: Find the Euclidean distance of each of the Hessian parameter points for each descriptor from the 8 vocabularies. Whichever of the 8 values gives the minimum value, store that index and increment the respective histogram data index value by 1.

Step-8: Save these data to .txt files and plot the data in MATLAB which will give the Histogram of each of the descriptors.

Step-9: The question now asks to what does the Convertible2 match to.

Step-10: For this find the error in each of the histograms from SUV, Truck and Convertible one with that of Convertible2.

Step-11: Error calculation can be done by subtracting individual bin values of Convertible2 from each of SUV, Truck and Convertible1 and then taking a mean of the data.

Step-12: Now there will be three error values obtained, one representing Convertible2/SUV, second Convertible2/Truck and the third Convertible2/Convertible1.

Step-13: Whichever error is the least indicates the highest match which is the desired output.

Step-14: Repeat the entire process with different Hessian parameters ranging from 100 to 800.

III. Experimental Results

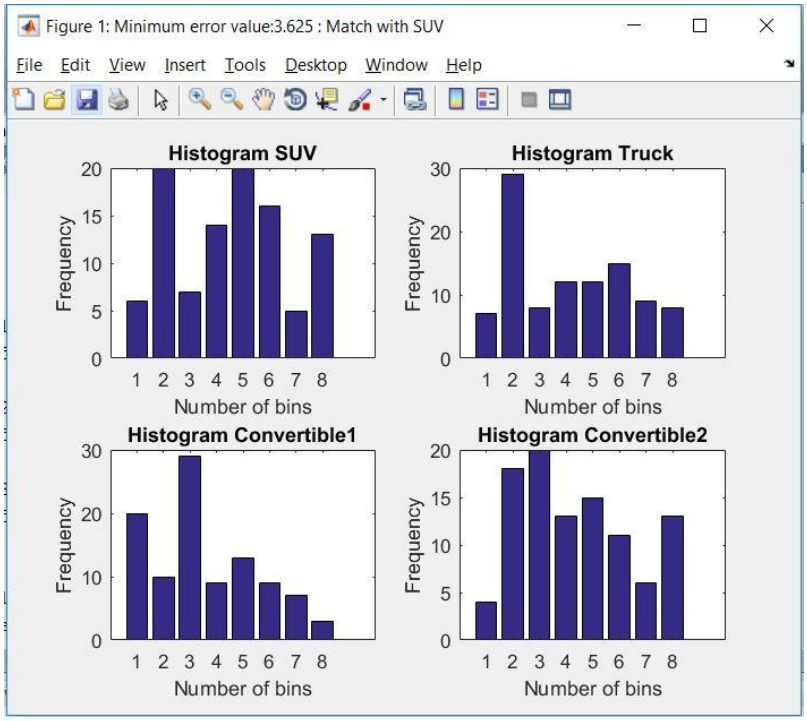

Figure-3.29: Convertible2 matching SUV based on histogram for Hessian parameter = 100 (check title bar of figure window for error value and details of match)

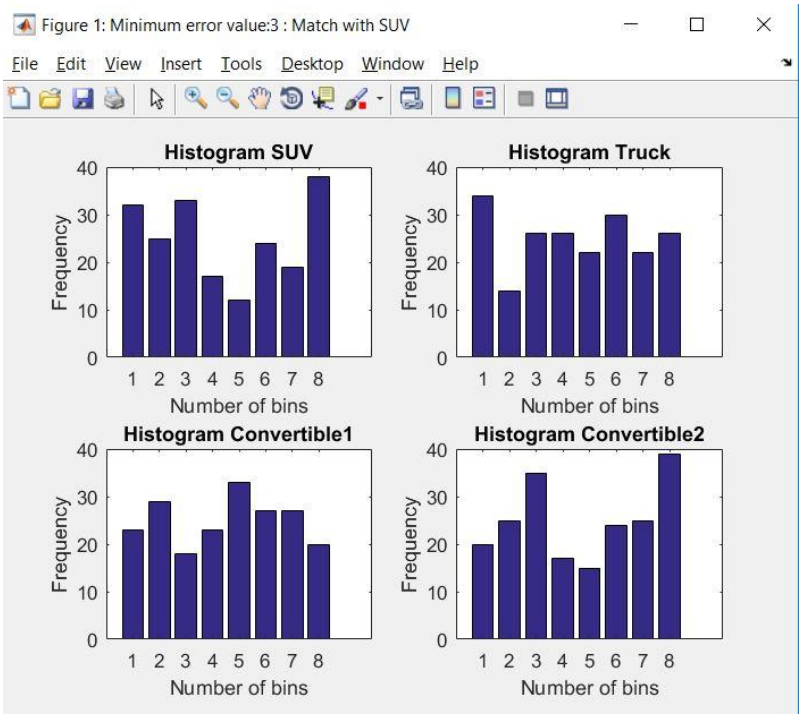

Figure-3.30: Convertible2 matching SUV based on histogram for Hessian parameter = 200 (check title bar of figure window for error value and details of match)

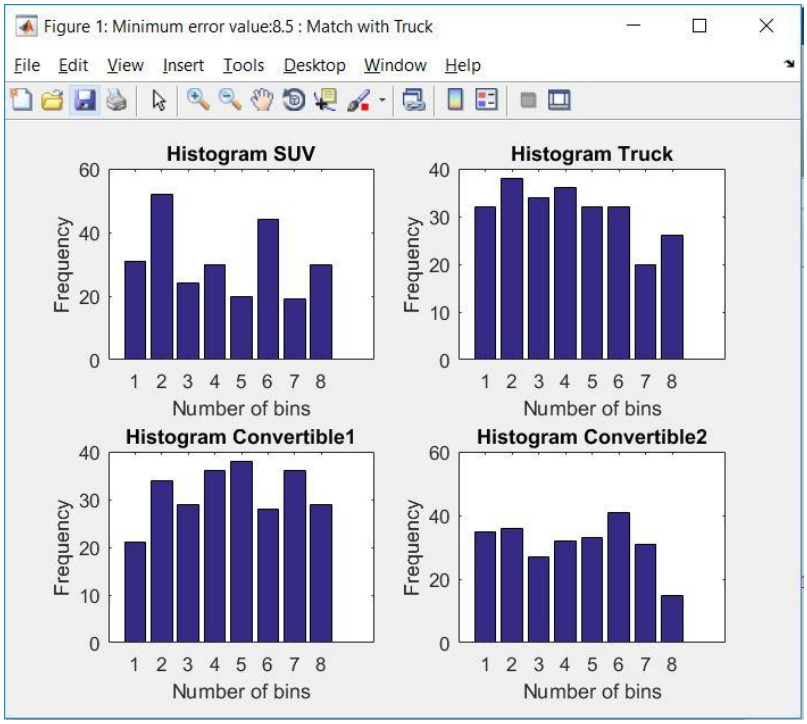

Figure-3.31: Convertible2 matching Truck based on histogram for Hessian parameter = 250 (check title bar of figure window for error value and details of match)

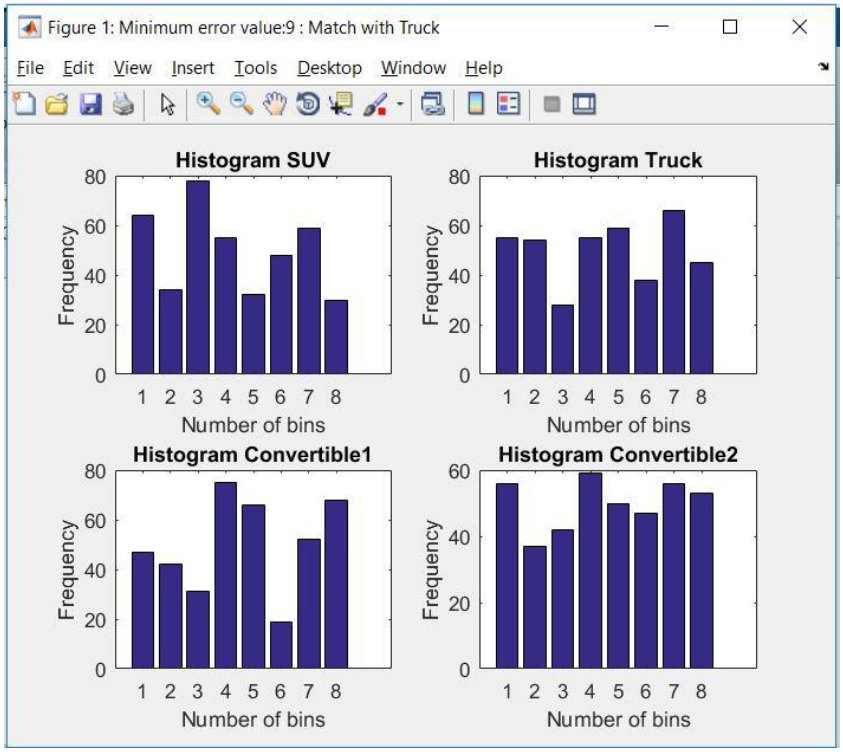

Figure-3.32: Convertible2 matching Truck based on histogram for Hessian parameter = 400 (check title bar of figure window for error value and details of match)

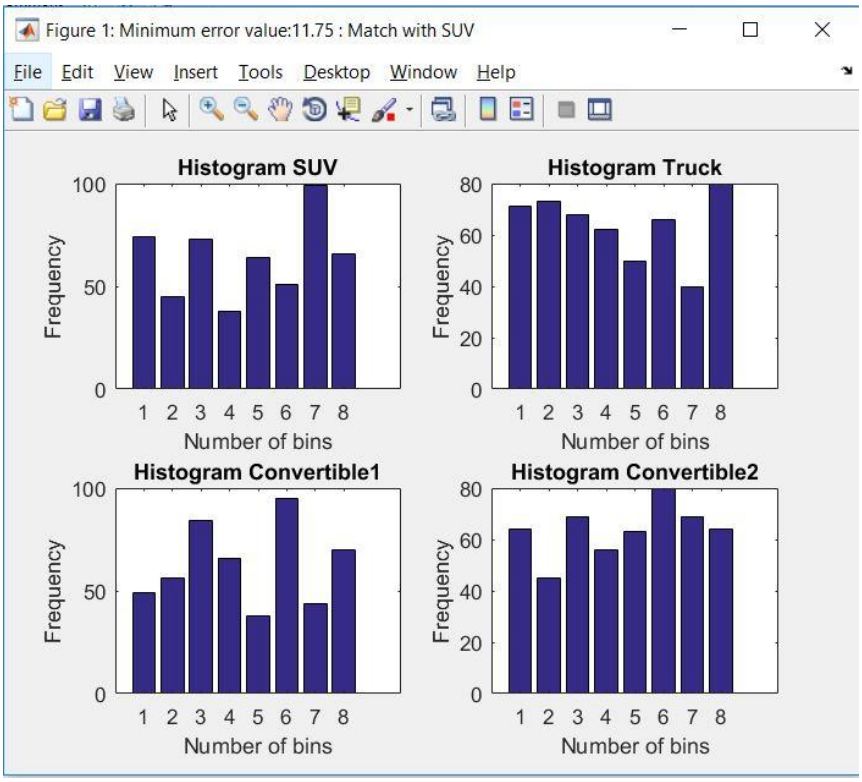

Figure-3.33: Convertible2 matching SUV based on histogram for Hessian parameter = 510 (check title bar of figure window for error value and details of match)

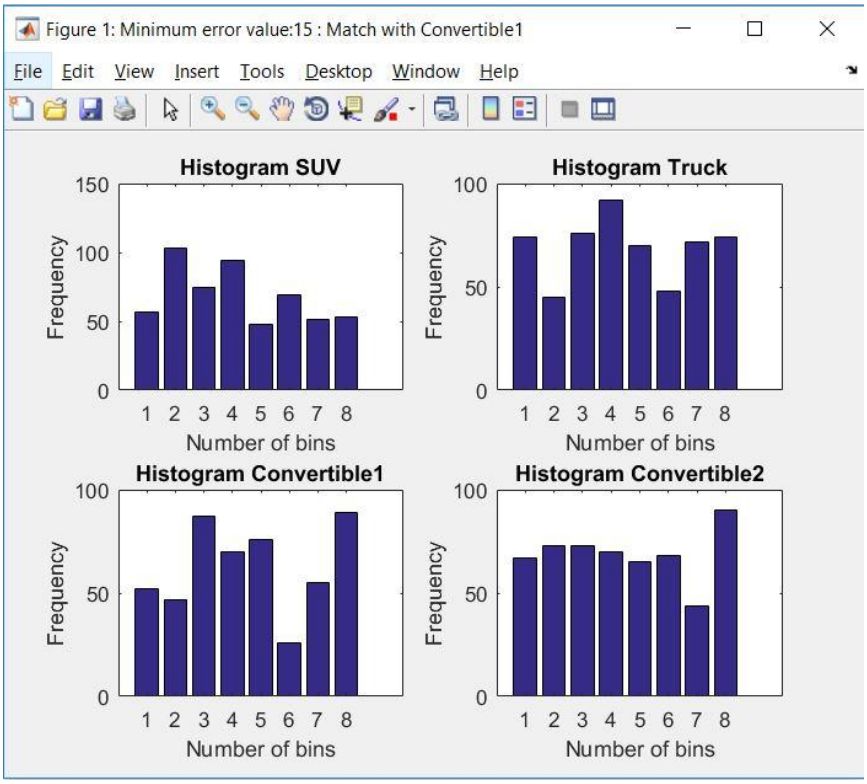

Figure-3.34: Convertible2 matching Convertible1 based on histogram for Hessian parameter = 550 (check title bar of figure window for error value and details of match)

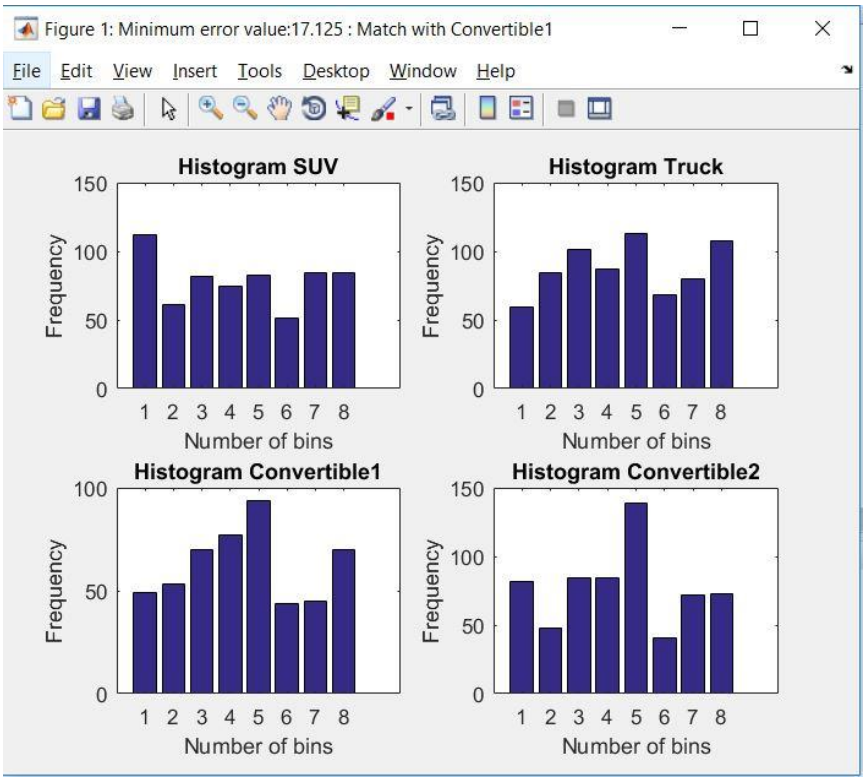

Figure-3.35: Convertible2 matching Convertible1 based on histogram for Hessian parameter = 700 (check title bar of figure window for error value and details of match)

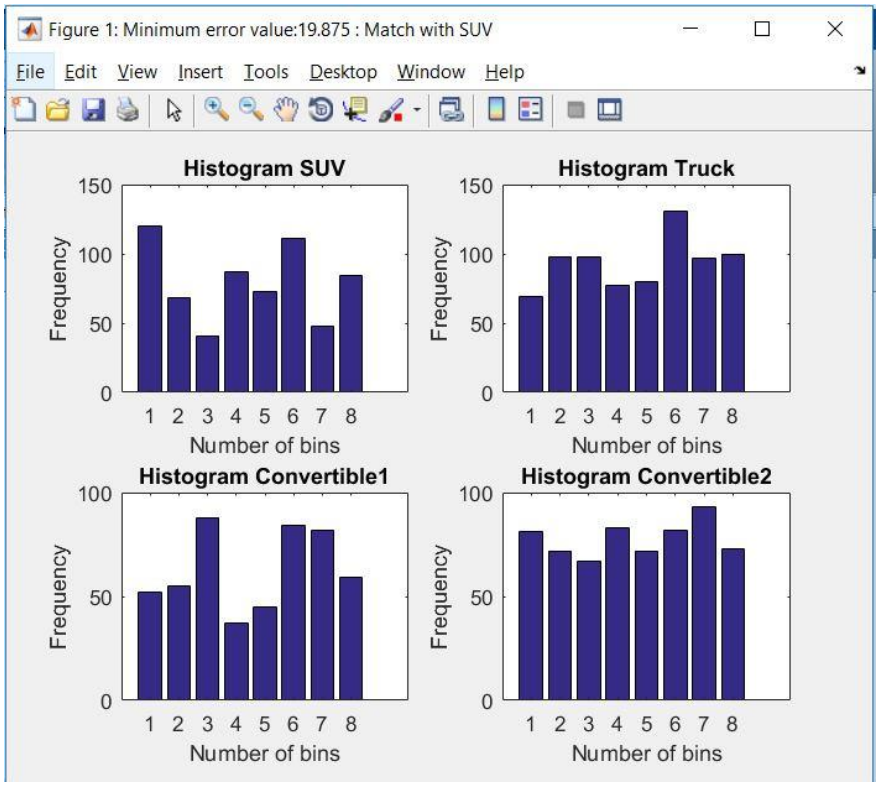

Figure-3.36: Convertible2 matching SUV based on histogram for Hessian parameter = 750 (check title bar of figure window for error value and details of match)

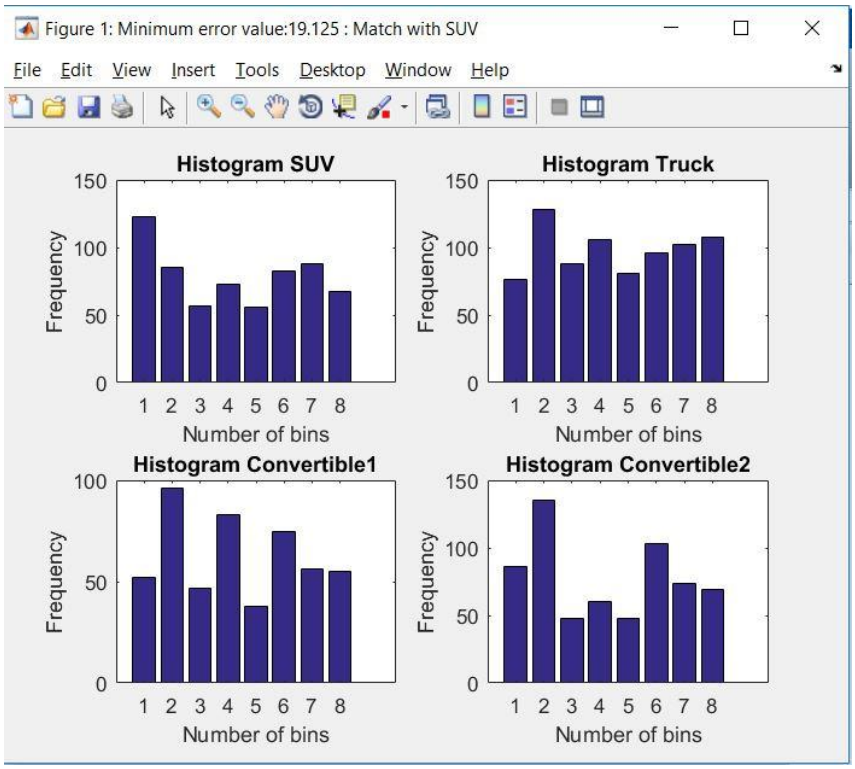

Figure-3.37: Convertible2 matching SUV based on histogram for Hessian parameter = 800 (check title bar of figure window for error value and details of match)

IV. Discussion

As it can be seen from the above figures, with the Hessian parameter range between 550 to 700, the Convertible2 matches with that of Convertible1 which is even true doing a visual inspection, the rest of the times it matches either the Truck or the SUV. These were concluded based on the error values obtained from histogram stating the degree of similarity or dissimilarity in the images. This can happen due to multiple reasons as discussed under:

- The dimensions of the images may vary.
- The orientation of the images may vary.
- The quality of the image given as input is also a major driving factor. This is something that I observed while converting the .raw file to .jpg. When I saved it at a lower quality it was always getting classified as SUV but when I saved it at the highest quality then as stated above within a defined range it matches Convertible1 which is what is expected even on doing a visual inspection. This explanation might seem a little vague but it is

References

[1] Digital Image Processing, Second Edition, Rafael Gonzalez, Richard E Woods, Pearson Education
[2] Digital Image Processing, Fourth Edition, William K. Pratt, Wiley-Interscience Publication
[3] Discussion notes
[4] Class notes
[5] Image matching using SIFT and SURF: https://www.researchgate.net/publication/292157133_Image_Matching_Using_SIFT_SURF_BRIEF_and_ORB_Performance_Comparison_for_Distorted_Images
[6] OpenCV documentation from www.opencv.org for OpenCV installation and reference source codes
[7] https://github.com/pdollar/edges for edge detection question
[8] Color code for implementation in segmentation problem from: http://www.rapidtables.com/web/color/RGB_Color.htm
[9] Complete setup of OpenCV steps followed from: http://audhootchavancv.blogspot.com/2015/08/how-to-install-opencv-30-and.html

Index

1. Algorithm for allocating memory dynamically using C++

This method is used to initialize the variables required to get the data in the images to be accessible and manipulated through. Since the image pixel values lie between 0-255 and the image types that are being used for processing at RAW format, the variables to be defined are taken to be of unsigned char datatype. The size of these variables could vary depending on what the user has given as input through command line. Hence, the concept of dynamic memory allocation has been used to use these command line fed values to dynamically be assigned as size of different variables that need them. The steps involved are as under.

Step-1: Using new operator initialize a 1D, 2D or 3D dynamic array pointer using the syntax:

unsigned char *** <variable_name> = new unsigned char **[<1st dimension size>]()

where,

*** = Each star indicates a pointer pointing to a dimension in memory. The 3 pointers therefore indicate the dynamic initialization of 3D array.

The complete expression therefore indicates a pointer holding the address of where the <1st dimension size> starts.

Step-2: Initialize the next dimensions similarly looping about in for loops of size <previous dimension>.

2. Algorithm for deallocating memory dynamically using C++

Once the data has been used so it is necessary we free up that space before exiting the program else it will unnecessarily consume space in the memory. The reason as to why the memory needs to be freed before the instance of a program execution is because as the pointer randomly assigns the next available space so once the program execution stops the pointer stops pointing to that place and loses track of it. Thus, the space just remains unused and occupied permanently. So, the following steps are followed for used memory deallocation:

Step-1: Make sure there is no further use of the variable to be deleted

Step-2: Using *N* nested for loops for *N* dimensions, use *delete[]* to free up variable spaces from higher dimension to lower dimension as under:

delete[] <variable name>[1st dimension][2nd dimension]….[Nth dimension]

3. Algorithm for FileRead() function developed using C++

Step-1: Define a FILE pointer.

Step-2: Define an error handling variable.

Step-3: Ensure that the file can be read using the error handling variable above and the use function fopen() to open the file.

Step-4: Read data from the file using fread() and store it to a variable. The data read will always be in the form of a 1D array.

Step-5: Close the file.

4. Algorithm for FileWrite() function developed using C++

Step-1: Define a FILE pointer.

Step-2: Define an error handling variable.

Step-3: Ensure that the file can be written using the error handling variable above and the use function fopen() to open the file.

Step-4: Write data into the file using fwrite(). The data to be written should always be in a 1D array format.

Step-5: Close the file.

5. Algorithm for Histogram3d() function developed using C++

Step-1: Define a HistData[256][3] array.

Step-2: Take the 3D image data as input.

Step-3: Use a counter for each channel to keep track of how many times an intensity occurs.

Step-4: Save this data to HistData[256][3] which then has the final histogram data for all the 256 grayscales across each of the 3 channels.

6. Algorithm for Histogram2d() function developed using C++

Step-1: Define a HistData[256] array.

Step-2: Take the 2D image data as input.

Step-3: Use a counter for to keep track of how many times an intensity occurs.

Step-4: Save this data to HistData[256] which then has the final histogram data for all the 256 grayscales.

7. Algorithm for FileWriteHist3d() function developed using C++

Step-1: Define a FILE pointer.

Step-2: Ensure that the file can be written using the error handling variable above and the use function fopen() to open the file.
Step-3: Write HistData[256][3] in 3 column format to a text file .
Step-4: Close the file.

8. Algorithm for FileWriteHist2d() function developed using C++

Step-1: Define a FILE pointer.
Step-2: Ensure that the file can be written using the error handling variable above and the use function fopen() to open the file.
Step-3: Write HistData[256] in 1 column format to a text file .
Step-4: Close the file.

9. Algorithm for plotting histogram using MATLAB

Step-1: Use fopen() to open and read the contents of a file and save it to a variable
Step-2: Use textscan() function to scan the text content is the file and store each column to a separate variable
Step-3: The first column gives the pixel intensity and the remaining columns gave the number of occurrences of these pixels in the R, G, B channels respectively.
Step-4: Using the above stored column data along with subplot() and plot() functions the final histogram of each channel can be plotted.