Leila Suasnabar
3109-9816-56
lsuasnab@usc.edu

EE 569: Homework #3

Problem 1 : Texture Analysis and Segmentation

1.1 Motivation

By analysing the texture of an image, we can obtain its features with a filter bank and then classified them. This allows for the program to recognize similar patches of textures (based on its features extracted) and thus obtain an sketch of the objects found within the image. This is called, texture segmentation, which allows to segment the image into regions (aka. Shapes, objects, similar textures). For this problem, we cluster the pixels according to its energy feature vector.

1.2 Approach

1.2.1 Texture Classification

The training images are classified through two algorithms: supervised (visual inspection) and unsupervised (k-means) algorithm. The goal of both algorithms is to find a centroid for each cluster class, and use these to classify the testing data into a class. For the supervised method, the images are given a label and then a centroid is computed for each cluster (4 clusters total).

To find the centroids in the unsupervised method, the data was first clustered into 2 groups (mean+std, mean-std) and the new centroids were found, this was performed until a stable state was reached, this was repeated again with each group and which produced 4 centroids. In the supervised method, the mean of the features belonging to each class was computed and stored as the centroid. These centroids were compared to each testing feature, and their euclidean distance was computed in order to find the cluster each data point belongs to.

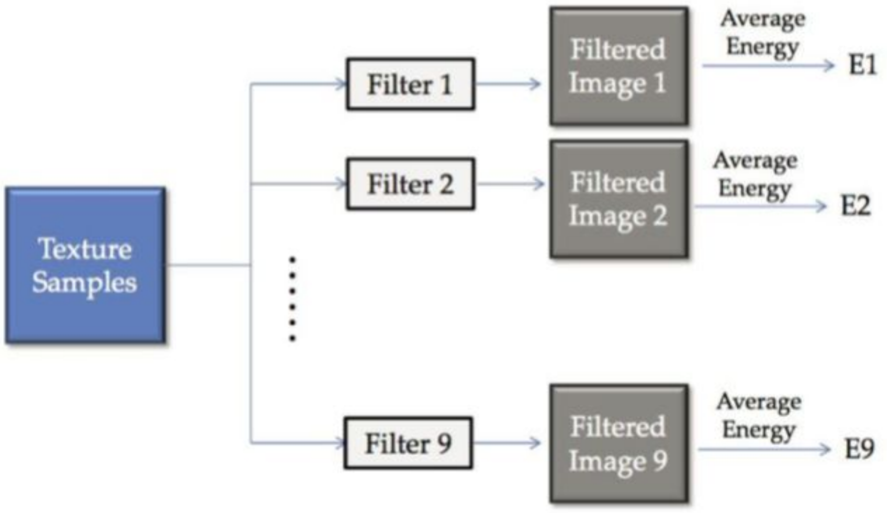


Diagram to obtain the 25D feature of every training image, the global mean was first subtracted from the original image. [1]

Table 1: 1D Kernel for 5x5 Laws Filters

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]

Outer product was performed to obtain the 25 filters.[1]

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Euclidean Distance formula

Lastly, the PCA algorithm was performed on the data to reduce the features from 25 to 3. Then the supervised/unsupervised algorithm was performed once again with these new features, and tested on the reduced features obtained from the test images in order to classify them.

1.2.2 Texture Segmentation

The image was first converted to grayscale, with:

$$\text{Grayscale Value} = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

Then the global mean was subtracted from the pixels, similarly to the previous part, the image is expanded and stored in grayscale. After obtaining 25 grayscale images, a feature vector is obtained from each pixel. This is done by selecting a window size (neighborhood around the pixel) and filtered using the laws filter to obtain the 25D energy features. This process is repeated on every pixel. Next, the features are normalized by dividing every pixel 25D energy feature vector with its first element (L5L5 energy feature value). Finally, the k-means algorithm written from the first part is used in this part as well.

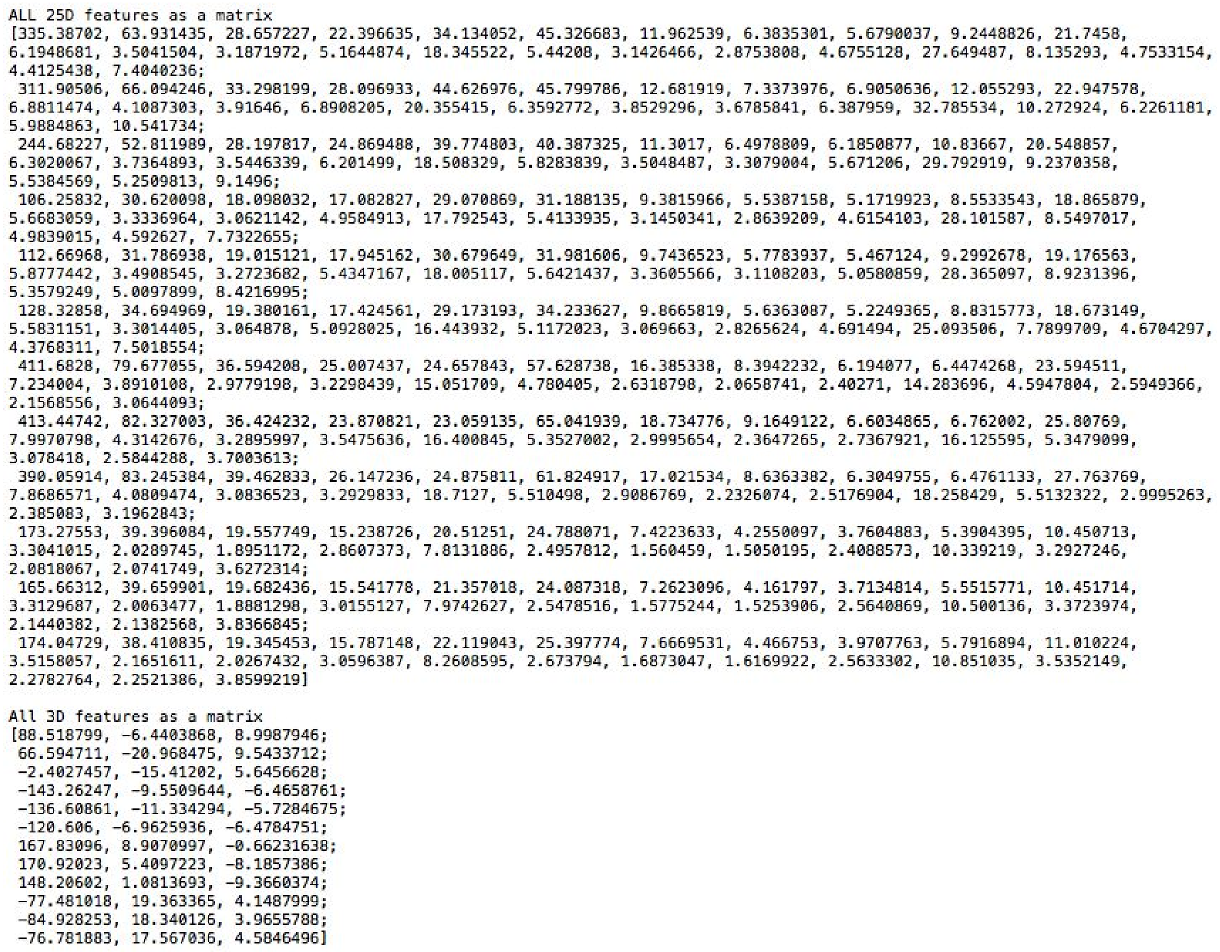
1.2.3 Further Improvement

First the texture classification is repeated as in part 1.2.2 except with unnormalized data. Second, the 25D energy features vectors are passed on PCA to reduce its dimensionality. PCA can extract the most valuable features and thus reduce the original feature dimensionality.

Lastly, the image is checked per label. This means we produce several binary images, comparing a label vs the remaining labels at a time. From these, we look for small blobs that are not connected to a bigger region. These blobs are then eliminated and the binary images are combined back into the original image showing all the clusters.

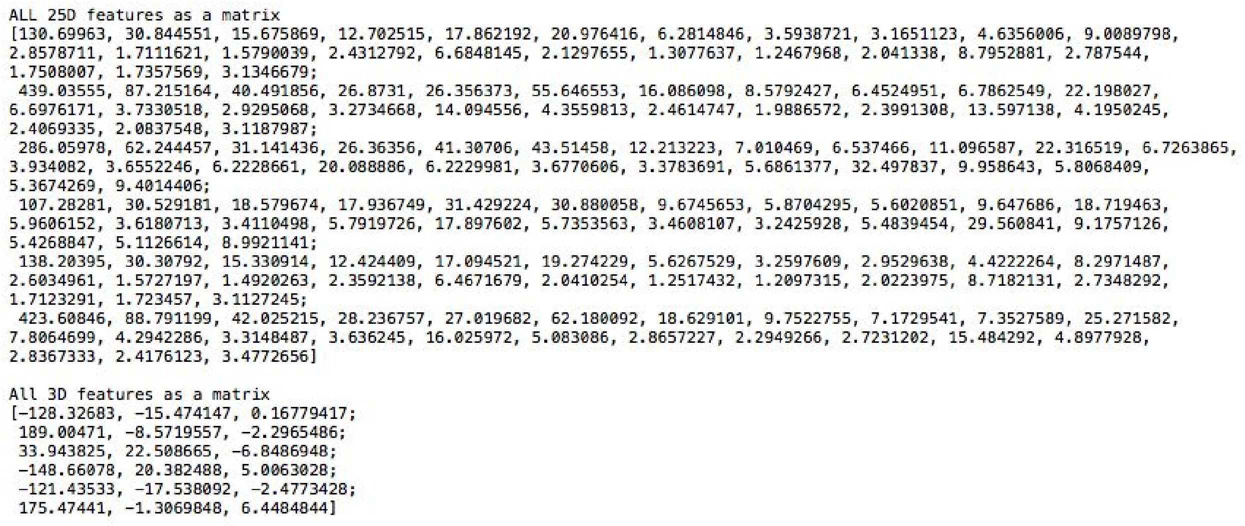
1.3 Results

1.3.1 Texture Classification



```
ALL 25D features as a matrix
[335.38702, 63.931435, 28.657227, 22.396635, 34.134052, 45.326683, 11.962539, 6.3835301, 5.6790037, 9.2448826, 21.7458,
6.1948681, 3.5041504, 3.1871972, 5.1644874, 18.345522, 5.44208, 3.1426466, 2.8753808, 4.6755128, 27.649487, 8.135293, 4.7533154,
4.4125438, 7.4040236;
 311.90506, 66.094246, 33.298199, 28.096933, 44.626976, 45.799786, 12.681919, 7.3373976, 6.9050636, 12.055293, 22.947578,
6.8811474, 4.1087303, 3.91646, 6.8908205, 20.355415, 6.3592772, 3.8529296, 3.6785841, 6.387959, 32.785534, 10.272924, 6.2261181,
5.9884863, 10.541734;
 244.68227, 52.811989, 28.197817, 24.869488, 39.774803, 40.387325, 11.3017, 6.4978809, 6.1850877, 10.83667, 20.548857,
6.3020067, 3.7364893, 3.5446339, 6.201499, 18.508329, 5.8283839, 3.5048487, 3.3079004, 5.671206, 29.792919, 9.2370358,
5.5384569, 5.2509813, 9.1496;
 106.25832, 30.620098, 18.098032, 17.082827, 29.070869, 31.188135, 9.3815966, 5.5387158, 5.1719923, 8.5533543, 18.865879,
5.6683059, 3.3336964, 3.0621142, 4.9584913, 17.792543, 5.4133935, 3.1450341, 2.8639209, 4.6154103, 28.101587, 8.5497017,
4.9839015, 4.592627, 7.7322655;
 112.66968, 31.786938, 19.015121, 17.945162, 30.679649, 31.981606, 9.7436523, 5.7783937, 5.467124, 9.2992678, 19.176563,
5.8777442, 3.4908545, 3.2723682, 5.4347167, 18.005117, 5.6421437, 3.3605566, 3.1108203, 5.0580859, 28.365097, 8.9231396,
5.3579249, 5.0097899, 8.4216995;
 128.32858, 34.694969, 19.380161, 17.424561, 29.173193, 34.233627, 9.8665819, 5.6363087, 5.2249365, 8.8315773, 18.673149,
5.5831151, 3.3014405, 3.064878, 5.0928025, 16.443932, 5.1172023, 3.069663, 2.8265624, 4.691494, 25.093506, 7.7899709, 4.6704297,
4.3768311, 7.5018554;
 411.6828, 79.677055, 36.594208, 25.007437, 24.657843, 57.628738, 16.385338, 8.3942232, 6.194077, 6.4474268, 23.594511,
7.234004, 3.8910108, 2.9779198, 3.2298439, 15.051709, 4.780405, 2.6318798, 2.0658741, 2.40271, 14.283696, 4.5947804, 2.5949366,
2.1568556, 3.0644093;
 413.44742, 82.327003, 36.424232, 23.870821, 23.059135, 65.041939, 18.734776, 9.1649122, 6.6034865, 6.762002, 25.80769,
7.9970798, 4.3142676, 3.2895997, 3.5475636, 16.400845, 5.3527002, 2.9995654, 2.3647265, 2.7367921, 16.125595, 5.3479099,
3.078418, 2.5844288, 3.7003613;
 390.05914, 83.245384, 39.462833, 26.147236, 24.875811, 61.824917, 17.021534, 8.6363382, 6.3049755, 6.4761133, 27.763769,
7.8686571, 4.0809474, 3.0836523, 3.2929833, 18.7127, 5.510498, 2.9086769, 2.2326074, 2.5176904, 18.258429, 5.5132322, 2.9995263,
2.385083, 3.1962843;
 173.27553, 39.396084, 19.557749, 15.238726, 20.51251, 24.788071, 7.4223633, 4.2550097, 3.7604883, 5.3904395, 10.450713,
3.3041015, 2.0289745, 1.8951172, 2.8607373, 7.8131886, 2.4957812, 1.560459, 1.5050195, 2.4088573, 10.339219, 3.2927246,
2.0818067, 2.0741749, 3.6272314;
 165.66312, 39.659901, 19.682436, 15.541778, 21.357018, 24.087318, 7.2623096, 4.161797, 3.7134814, 5.5515771, 10.451714,
3.3129687, 2.0063477, 1.8881298, 3.0155127, 7.9742627, 2.5478516, 1.5775244, 1.5253906, 2.5640869, 10.500136, 3.3723974,
2.1440382, 2.1382568, 3.8366845;
 174.04729, 38.410835, 19.345453, 15.787148, 22.119043, 25.397774, 7.6669531, 4.466753, 3.9707763, 5.7916894, 11.010224,
3.5158057, 2.1651611, 2.0267432, 3.0596387, 8.2608595, 2.673794, 1.6873047, 1.6169922, 2.5633302, 10.851035, 3.5352149,
2.2782764, 2.2521386, 3.8599219]

All 3D features as a matrix
[88.518799, -6.4403868, 8.9987946;
 66.594711, -20.968475, 9.5433712;
 -2.4027457, -15.41202, 5.6456628;
 -143.26247, -9.5509644, -6.4658761;
 -136.60861, -11.334294, -5.7284675;
 -120.606, -6.9625936, -6.4784751;
 167.83096, 8.9070997, -0.66231638;
 170.92023, 5.4097223, -8.1857386;
 148.20602, 1.0813693, -9.3660374;
 -77.481018, 19.363365, 4.1487999;
 -84.928253, 18.340126, 3.9655788;
 -76.781883, 17.567036, 4.5846496]
```

Fig 1. 25-D Feature vector and reduced 3-D Feature vector from the Training Data



```
ALL 25D features as a matrix
[130.69963, 30.844551, 15.675869, 12.702515, 17.862192, 20.976416, 6.2814846, 3.5938721, 3.1651123, 4.6356006, 9.0089798,
2.8578711, 1.7111621, 1.5790039, 2.4312792, 6.6848145, 2.1297655, 1.3077637, 1.2467968, 2.041338, 8.7952881, 2.787544,
1.7508007, 1.7357569, 3.1346679;
 439.03555, 87.215164, 40.491856, 26.8731, 26.356373, 55.646553, 16.086098, 8.5792427, 6.4524951, 6.7862549, 22.198027,
6.6976171, 3.7330518, 2.9295068, 3.2734668, 14.094556, 4.3559813, 2.4614747, 1.9886572, 2.3991308, 13.597138, 4.1950245,
2.4069335, 2.0837548, 3.1187987;
 286.05978, 62.244457, 31.141436, 26.36356, 41.30706, 43.51458, 12.213223, 7.010469, 6.537466, 11.096587, 22.316519, 6.7263865,
3.934082, 3.6552246, 6.2228661, 20.088886, 6.2229981, 3.6770606, 3.3783691, 5.6861377, 32.497837, 9.958643, 5.8068409,
5.3674269, 9.4014406;
 107.28281, 30.529181, 18.579674, 17.936749, 31.429224, 30.880058, 9.6745653, 5.8704295, 5.6020851, 9.647686, 18.719463,
5.9606152, 3.6180713, 3.4110498, 5.7919726, 17.897602, 5.7353563, 3.4608107, 3.2425928, 5.4839454, 29.560841, 9.1757126,
5.4268847, 5.1126614, 8.9921141;
 138.20395, 30.30792, 15.330914, 12.424409, 17.094521, 19.274229, 5.6267529, 3.2597609, 2.9529638, 4.4222264, 8.2971487,
2.6034961, 1.5727197, 1.4920263, 2.3592138, 6.4671679, 2.0410254, 1.2517432, 1.2097315, 2.0223975, 8.7182131, 2.7348292,
1.7123291, 1.723457, 3.1127245;
 423.60846, 88.791199, 42.025215, 28.236757, 27.019682, 62.180092, 18.629101, 9.7522755, 7.1729541, 7.3527589, 25.271582,
7.8064699, 4.2942286, 3.3148487, 3.636245, 16.025972, 5.083086, 2.8657227, 2.2949266, 2.7231202, 15.484292, 4.8977928,
2.8367333, 2.4176123, 3.4772656]

All 3D features as a matrix
[-128.32683, -15.474147, 0.16779417;
 189.00471, -8.5719557, -2.2965486;
 33.943825, 22.508665, -6.8486948;
 -148.66078, 20.382488, 5.0063028;
 -121.43533, -17.538092, -2.4773428;
 175.47441, -1.3069848, 6.4484844]
```

Fig 2. 25-D Feature vector and reduced 3-D Feature vector from the Testing Data

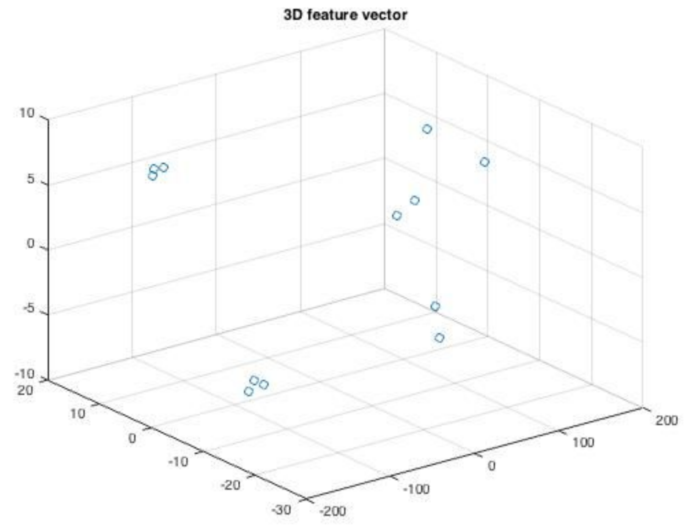


Fig 3. Plot of the 3D training features

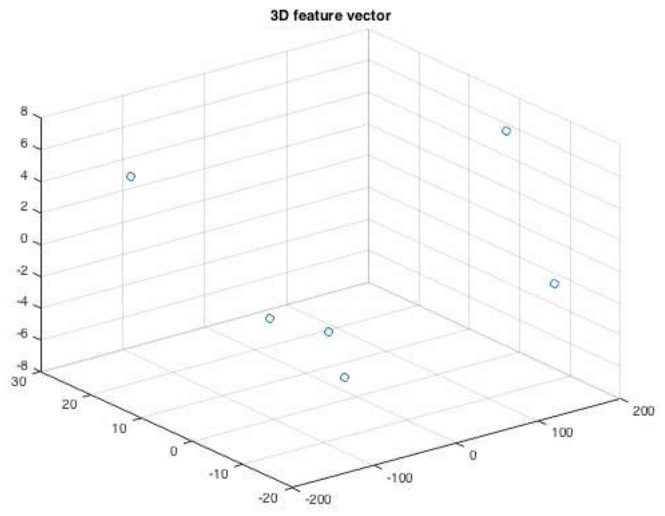
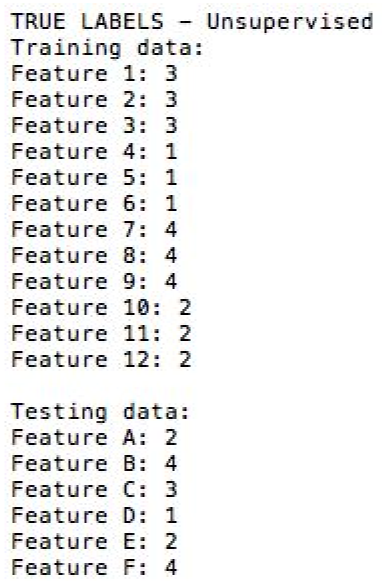
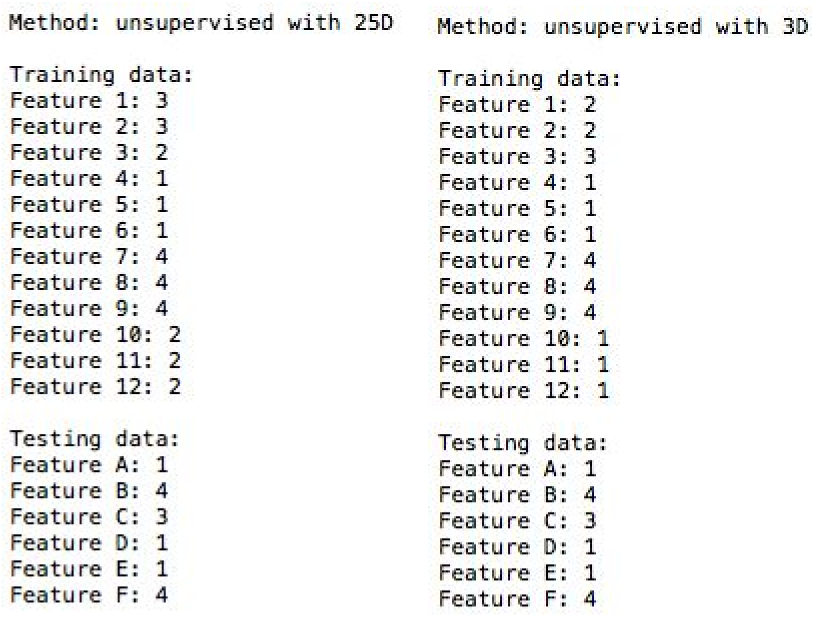


Fig 4. Plot of the 3D testing features



```
TRUE LABELS - Unsupervised
Training data:
Feature 1: 3
Feature 2: 3
Feature 3: 3
Feature 4: 1
Feature 5: 1
Feature 6: 1
Feature 7: 4
Feature 8: 4
Feature 9: 4
Feature 10: 2
Feature 11: 2
Feature 12: 2

Testing data:
Feature A: 2
Feature B: 4
Feature C: 3
Feature D: 1
Feature E: 2
Feature F: 4
```

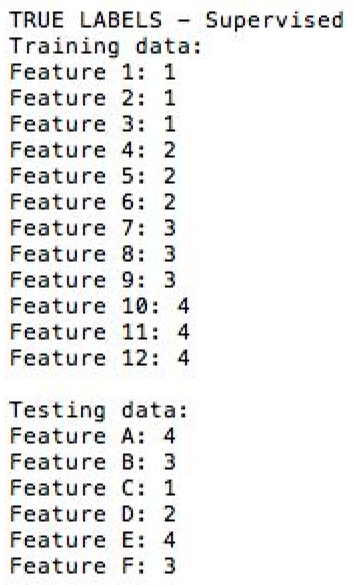
Fig 5. True Labels for Unsupervised Classification



```
Method: unsupervised with 25D

Training data:
Feature 1: 3
Feature 2: 3
Feature 3: 2
Feature 4: 1
Feature 5: 1
Feature 6: 1
Feature 7: 4
Feature 8: 4
Feature 9: 4
Feature 10: 2
Feature 11: 2
Feature 12: 2

Testing data:
Feature A: 1
Feature B: 4
Feature C: 3
Feature D: 1
Feature E: 1
Feature F: 4
```

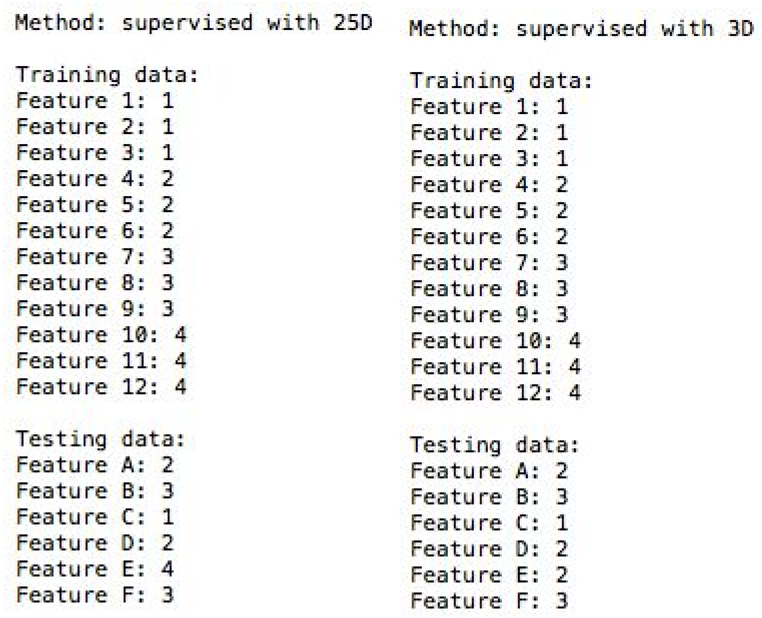
```
Method: unsupervised with 3D

Training data:
Feature 1: 2
Feature 2: 2
Feature 3: 3
Feature 4: 1
Feature 5: 1
Feature 6: 1
Feature 7: 4
Feature 8: 4
Feature 9: 4
Feature 10: 1
Feature 11: 1
Feature 12: 1

Testing data:
Feature A: 1
Feature B: 4
Feature C: 3
Feature D: 1
Feature E: 1
Feature F: 4
```

Fig 6. Label Output for Unsupervised Classification



```
TRUE LABELS - Supervised
Training data:
Feature 1: 1
Feature 2: 1
Feature 3: 1
Feature 4: 2
Feature 5: 2
Feature 6: 2
Feature 7: 3
Feature 8: 3
Feature 9: 3
Feature 10: 4
Feature 11: 4
Feature 12: 4

Testing data:
Feature A: 4
Feature B: 3
Feature C: 1
Feature D: 2
Feature E: 4
Feature F: 3
```

Fig 7. True Labels for Supervised Classification



```
Method: supervised with 25D

Training data:
Feature 1: 1
Feature 2: 1
Feature 3: 1
Feature 4: 2
Feature 5: 2
Feature 6: 2
Feature 7: 3
Feature 8: 3
Feature 9: 3
Feature 10: 4
Feature 11: 4
Feature 12: 4

Testing data:
Feature A: 2
Feature B: 3
Feature C: 1
Feature D: 2
Feature E: 4
Feature F: 3
```

```
Method: supervised with 3D

Training data:
Feature 1: 1
Feature 2: 1
Feature 3: 1
Feature 4: 2
Feature 5: 2
Feature 6: 2
Feature 7: 3
Feature 8: 3
Feature 9: 3
Feature 10: 4
Feature 11: 4
Feature 12: 4

Testing data:
Feature A: 2
Feature B: 3
Feature C: 1
Feature D: 2
Feature E: 2
Feature F: 3
```

Fig 8. Label Output for Supervised Classification:

1.3.2 Texture Segmentation

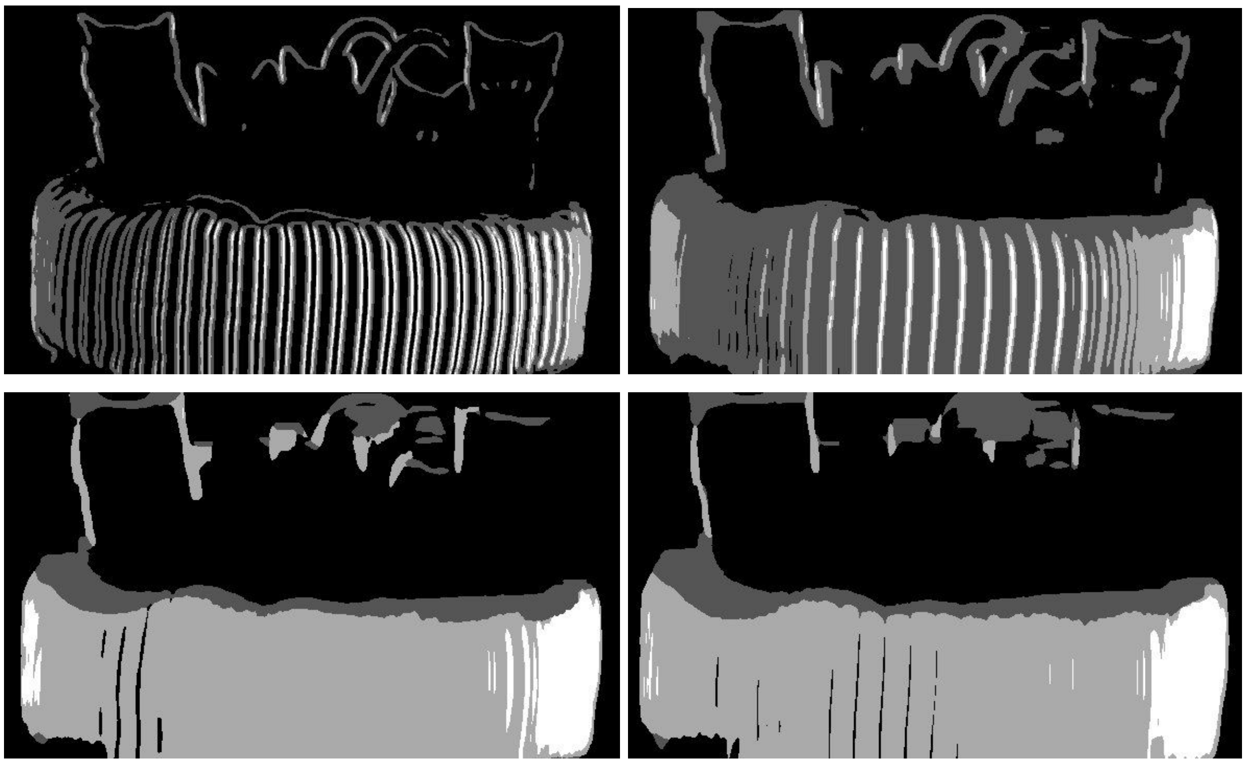


Fig 9. # of Clusters: 4. Top Left: window=5. Top Right: window=15. Bottom Left: window=25. Bottom Right: window=35.

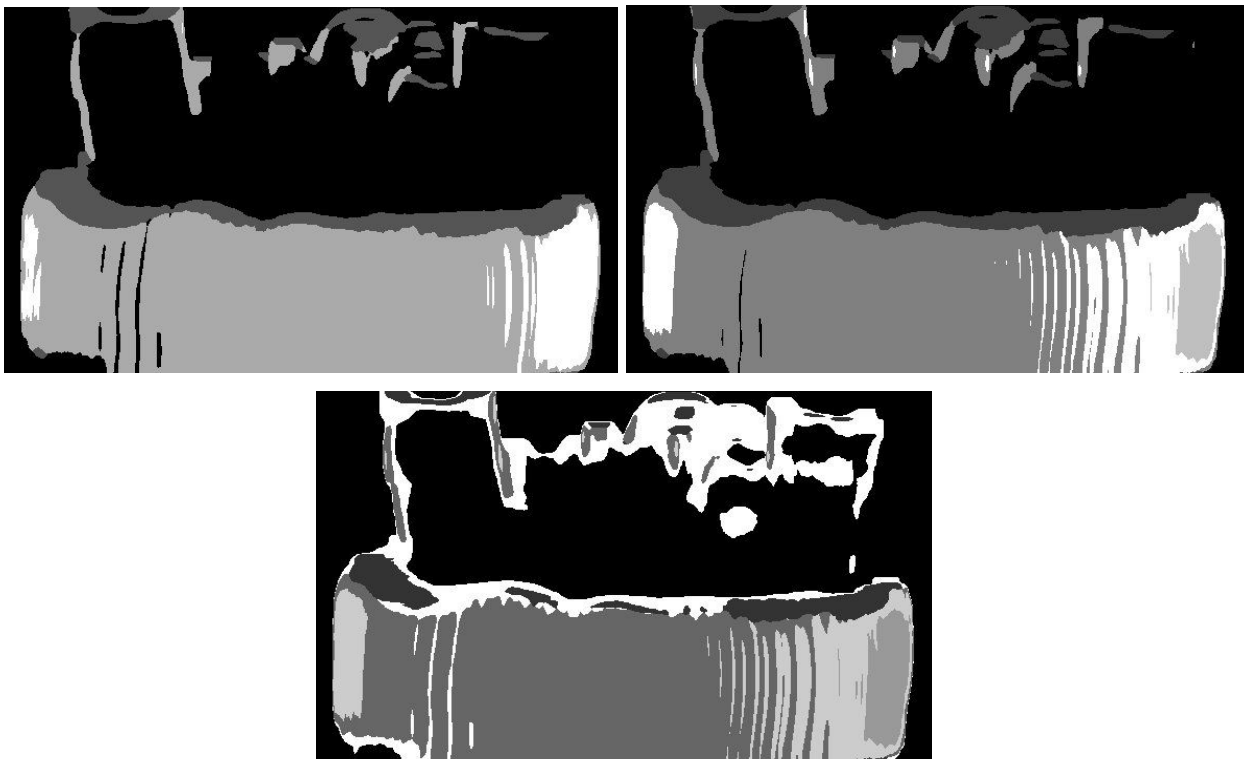


Fig 10. Window =25. Top Left: clusters=4. Top Right: clusters=5. Bottom: clusters=6

1.3.3 Further Improvement

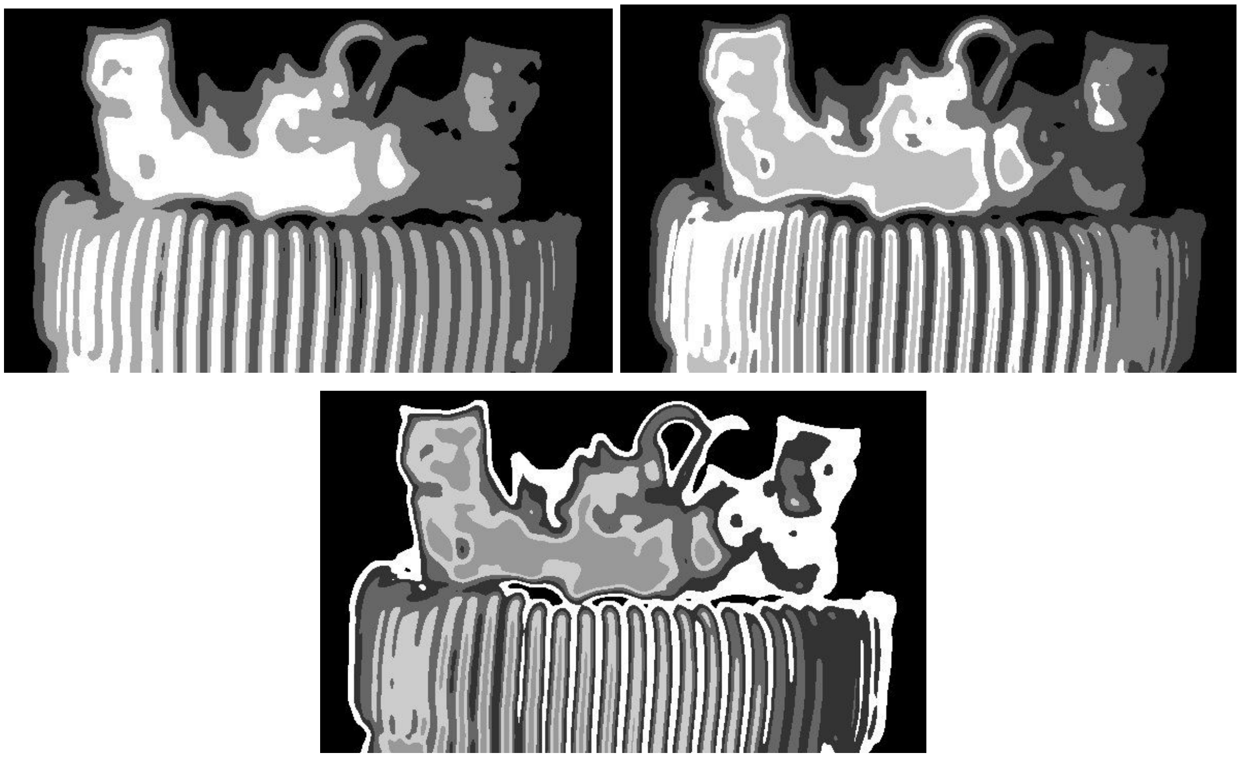


Fig 11. Unnormalized,window = 15. Left: clusters=4.Right: clusters=5. Bottom: clusters=6

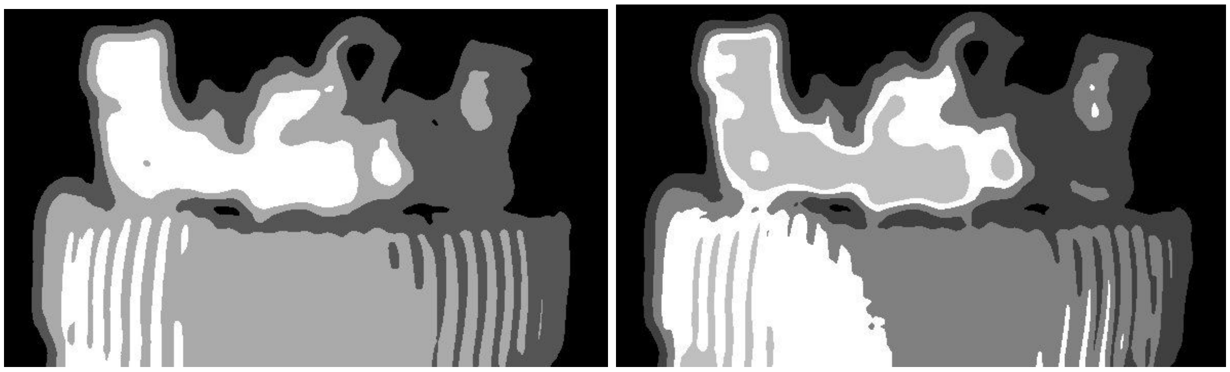


Fig 12. Unnormalized, window=25. Left: clusters=4. Right: clusters=5

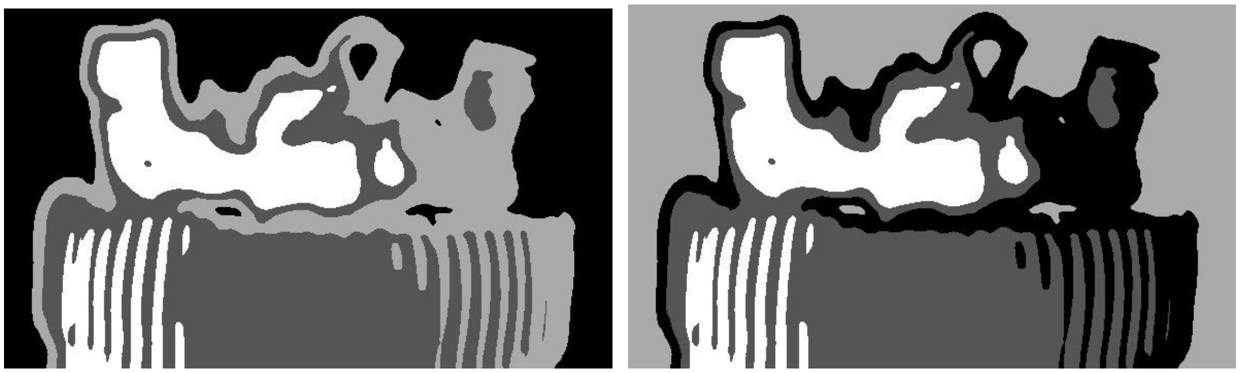


Fig 13. PCA. Left: 25 features, Right: 5 features.

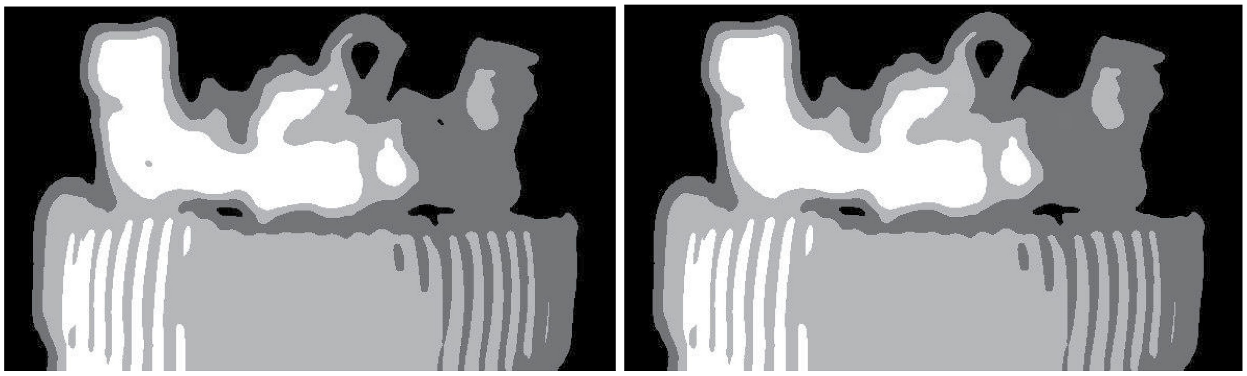


Fig 14. Fixing small holes, before and after. Window=25, cluster=4.

1.4 Discussion

1.4.1 Texture Classification

As can be seen from fig 1-2, the data was not normalized, additionally it shows that the first element has the strongest discriminant power (L5L5), while W5W5 had the weakest discriminant power. Notice Fig 3-4 show the 3D feature points and from this we can see that although two clusters can be correctly separated, the other two are close to one another and hence might produce errors when classifying data into these two clusters. Also, note that with the results we can see that these two clusters are the sand and grass classes, which are similar visually.

For the unsupervised method, we get from fig 5 the following labeling of classes: 3-bark , 1-sand, 4-straw, and 2-grass. On fig 6, we see that when using 25 or 3 dimensions the data is classified correctly except for one image and five images respectively. However, when looking at the classification of the testing data compares to its expected classification (fig 5), we see that when using 25 or 3 dimension feature vector there are exactly 2 misclassified images (feature A and E). Both, classified class 2-grass as class 1-sand.

For the supervised method, we get from fig 7 the following labeling of classes 1-bark , 2-sand, 3-straw, and 4-grass. Fig 8 shows that when using a 25 dimension feature vector, feature A was classified as class 2-sand instead of class 4-grass. Similarly, when using a 3 dimension feature vector, feature A and E were classified as class 2-sand instead of class 4-grass.

In both cases we see that the testing data was misclassified on class grass and sand, which is not surprising since the feature vectors of these two classes are far more similar than with the other classes. In this case, feature reduction wasn't that efficient for the training data, but performed equally with the testing data.

1.4.2 <u>Texture Segmentation</u>

Fig 9, shows the result of using different window sizes, from this we can see that as we increase the window size too much the segmented object's shape becomes less sharp. For instance the contour of the kitties' tails and ears lose their shape. Additionally, when using a small window size, the output segments even small areas from objects, which is not ideal. Later on we can see more proof of this in fig 11 and 12. Among the images from fig 9 we see that a window of size 25 seems to be the best choice in segmenting the shapes. Also, note that after normalizing the image seems to be unable to notice the difference between the texture of the kittens (fur) and the background. Also, in fig 10 we see the result of using different k values. For this part k=6 seems to be able to hold the outer shape of the kittens far more than the other values, but when k=5 the segmenting of the bed is more accurate.

1.4.3 <u>Further Improvement</u>

Fig 11 and 12, shows the result of not normalizing the data. This results in a clear segmentation of the kittens, further more the kittens are segmented into 2 groups, which are based on their fur color. For this part and from these two figures we see that the best result comes from using a window of size 25 and 4 clusters.

Upon using PCA for dimension reduction on unnormalized data we see that since the first feature of the feature vector has a larger value then the output of using fewer features produces no significant change (see fig 13).

Lastly, one fig 14, we see that the fixing holes algorithm was able to slightly enhance the segmentation. The small holes that did not belong within the regions were eliminated.

Problem 2 : Edge and Contour Detection

2.1 Motivation

Edge and contour detection is among one of the fundamental operations in image processing, and computer vision techniques. Edge detection is a mathematical method that attempts to find pixels where the image brightness changes sharply, find discontinuities in depth or surface orientation [3]. There are various techniques to detect edges within an image, and in this problem we'll see the Canny Edge Detector and the Structured Edge Detector.

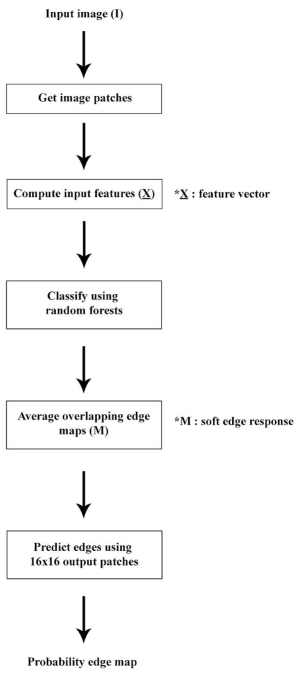
2.2 Approach

2.2.1 Canny Edge Detector

Canny edge detection allows to extract the structure information of an image; additionally, it is a an algorithm that provides good results and reliable edge detection. This algorithm applies double thresholds to determine possible edge pixels. The main idea of this detector is to pass a gaussian filter to smooth the image and reduce noise, and then apply a non-maximum suppression algorithm to obtain the binary edge map.[2]

For this part, the input image was first converted into grayscale and then applied with the Canny OpenCV function. This function takes as inputs the grayscale image and 2 threshold values (max and min). Multiple threshold pair values were applied to the images and stored back into raw files which are needed for part 2.2.3. Note, the output of the Canny function gives a 0 for the edges and 255 for the background, this binary data needs to be switched in order to be used for part 2.2.3.

2.2.2 Contour Detection with Structured Edge


Input image (I)
Get image patches
Compute input features (X)
*X : feature vector
Classify using random forests
Average overlapping edge maps (M)
*M : soft edge response
Predict edges using 16x16 output patches
Probability edge map

From the flowchart, the structured edge algorithm can be broken down in steps:

1. Find image patches from the input image and augment these new patches
2. Use these new patched to obtain its feature vector (X)
3. Classify using random forest
4. While merging, we average the overlapping edge maps to obtain a soft edge response
5. Run sliding 16x16 output patches to predict information for an entire image neighborhood
6. Obtain the final probability edge map

For this edge detector we classified the data using the random forest classifier. To understand this topic, we need to first understand that a decision tree describes information. Additionally, a decision tree classification can be the input for a further decision making algorithm like the random forest classifier. Nevertheless, a disadvantage of a decision tree is that it may overfit its training data.
The random forest classifier makes use of multiple decision trees to improve the classification rate of the data. This allows to correctly classify data and thus correct the disadvantage from the decision tree construction.

2.2.3 Performance Evaluation

The computed edge maps (from the previous parts) and ground truth edge map comparison is performed with the edgesEvalImg function which is found in the structured edge detection toolbox written by Piotr Dollar. For this problem the thresholds quantities is set to 5.
Classes that an edge map pixel can belong to:

FP	Detected as edge but it's not
TP	Detected as edge and it is
FN	Detected as background but it’s not
TN	Detected as background and it is

Formula to compute the recall, precision and F-measure of an edge compare compared to a ground truth image:

$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$

$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

$$F = 2 \times \frac{P \times R}{P + R}$$

2.3 Results

2.3.1 <u>Canny Edge Detector</u>

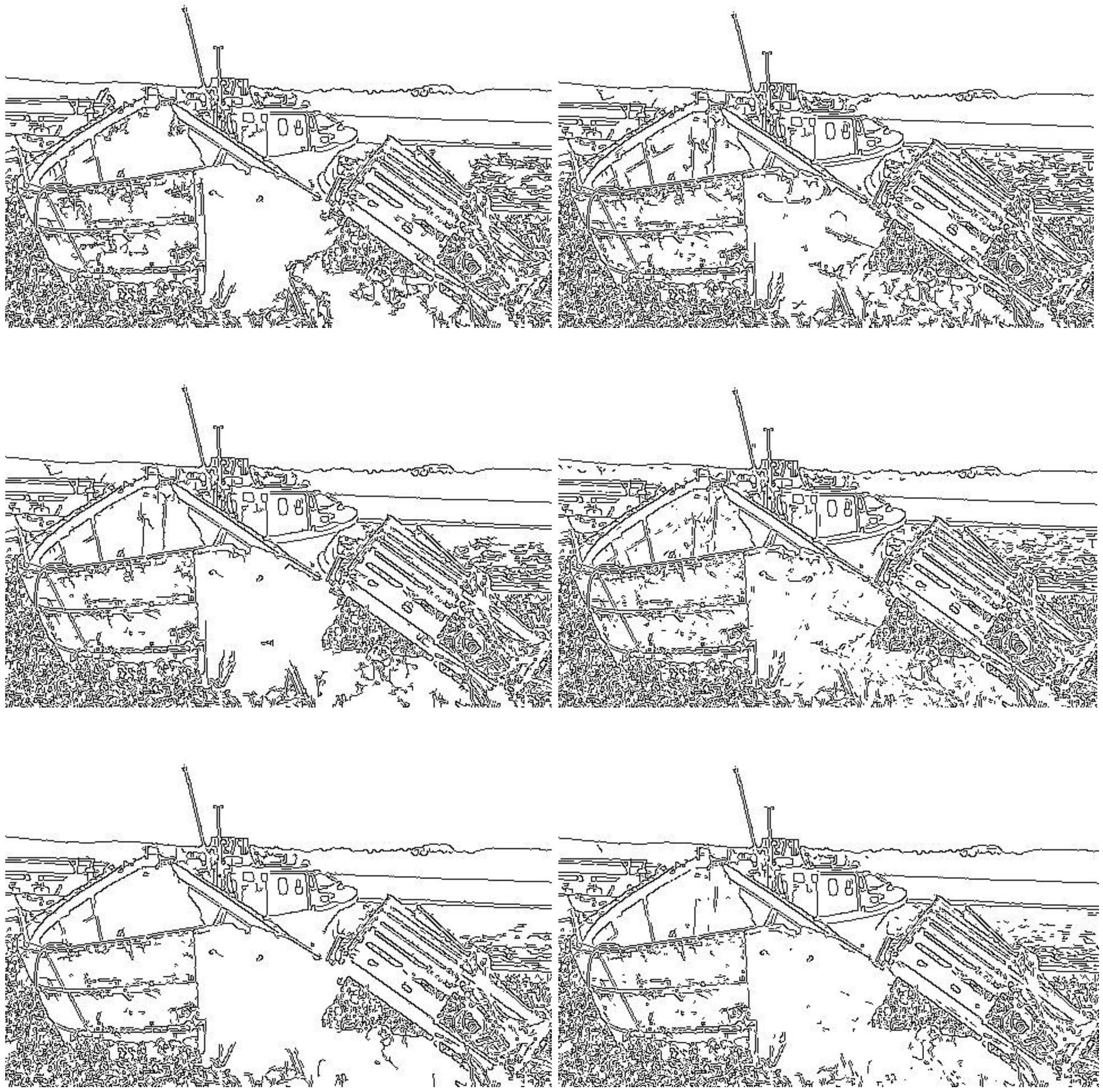


Fig 15. Edge maps of image with different low and high thresholds. Top Left: 0.1,0.9. Top Right: 0.2,0.5. Center Left: 0.2,0.7. Center Right: 0.3,0.4. Bottom Left: 0.3,0.8. Bottom Right: 0.4,0.6.

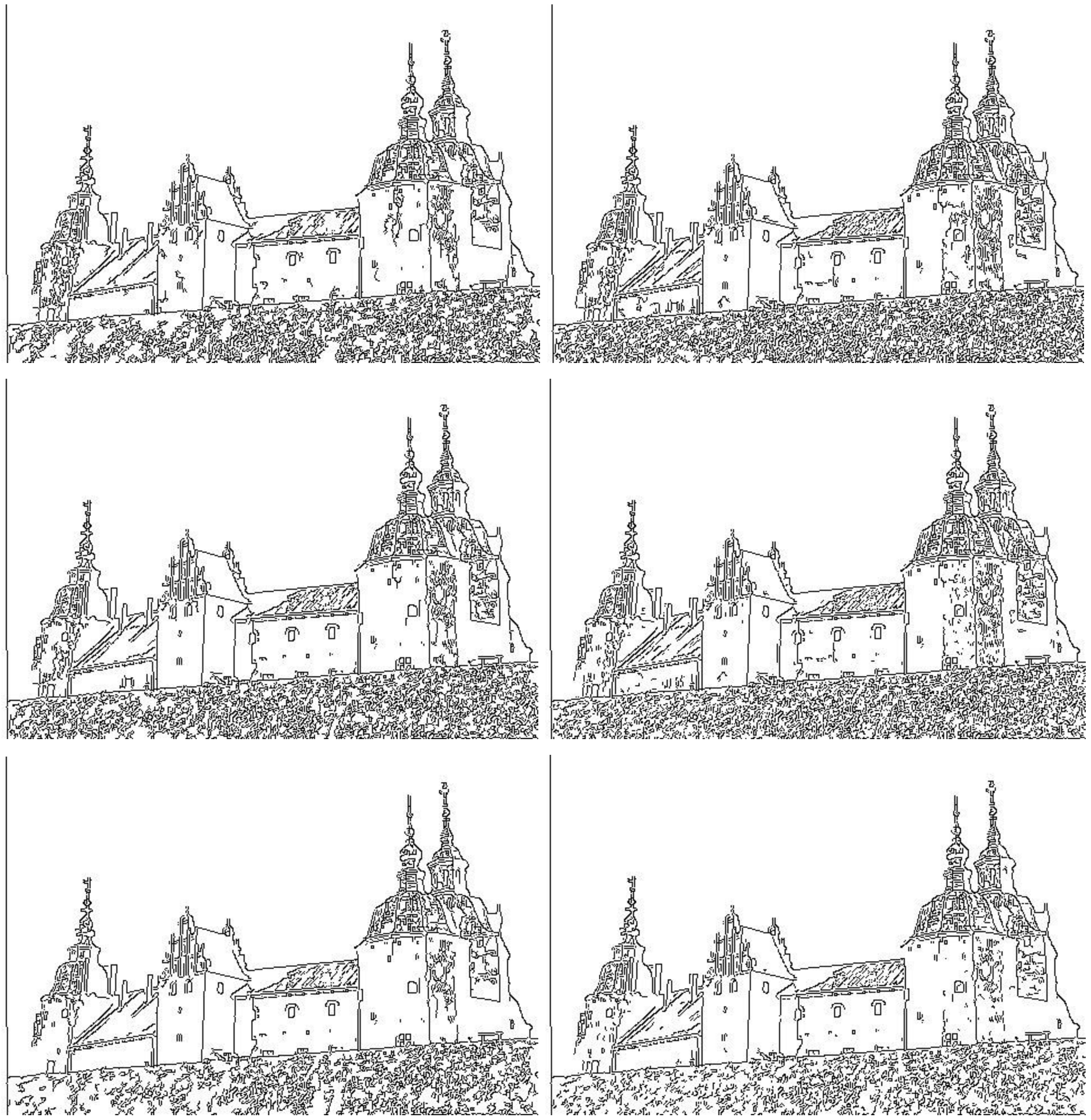


Fig 16. Image with different top and bottom thresholds. Top Left: 0.1,0.9. Top Right: 0.2,0.5. Center Left: 0.2,0.7. Center Right: 0.3,0.4. Bottom Left: 0.3,0.8. Bottom Right: 0.4,0.6.

2.3.2 Contour Detection with Structured Edge

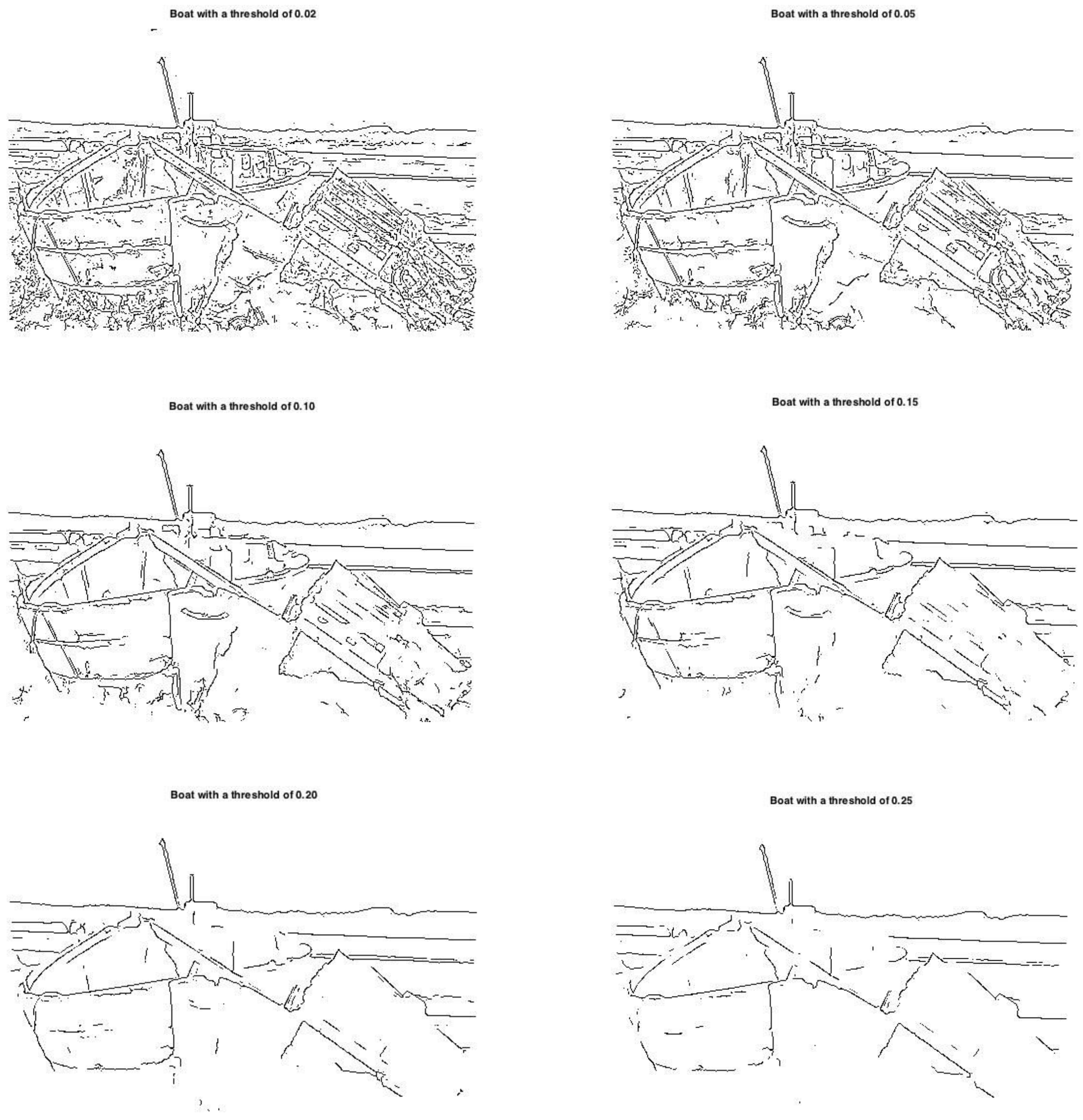

Boat with a threshold of 0.02
Boat with a threshold of 0.05
Boat with a threshold of 0.10
Boat with a threshold of 0.15
Boat with a threshold of 0.20
Boat with a threshold of 0.25

Fig 17. Boat Image with different thresholds. Top Left: 0.02. Top Right: 0.05. Center Left: 0.10. Center Right: 0.15. Bottom Left: 0.20. Bottom Right: 0.25.

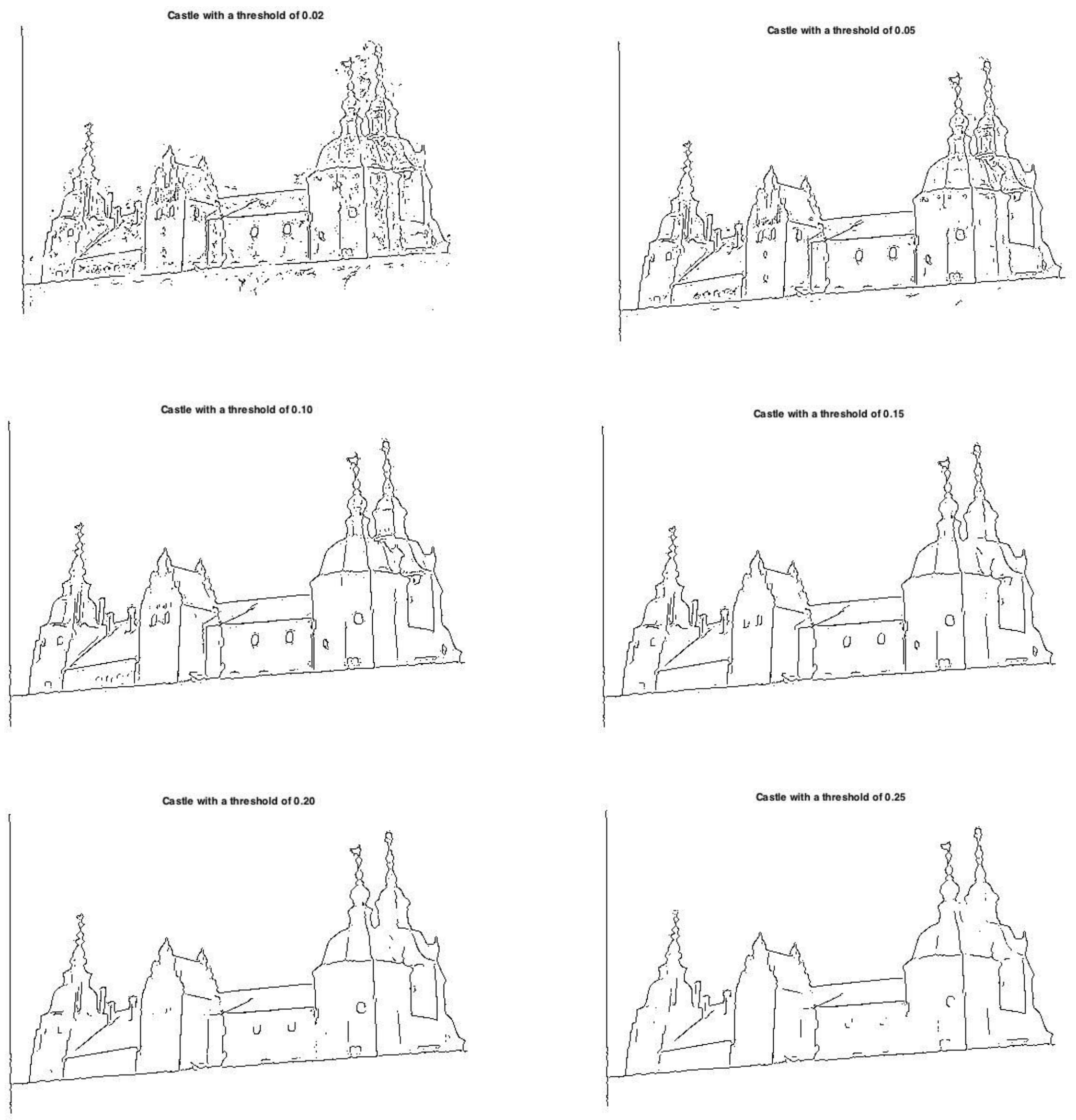
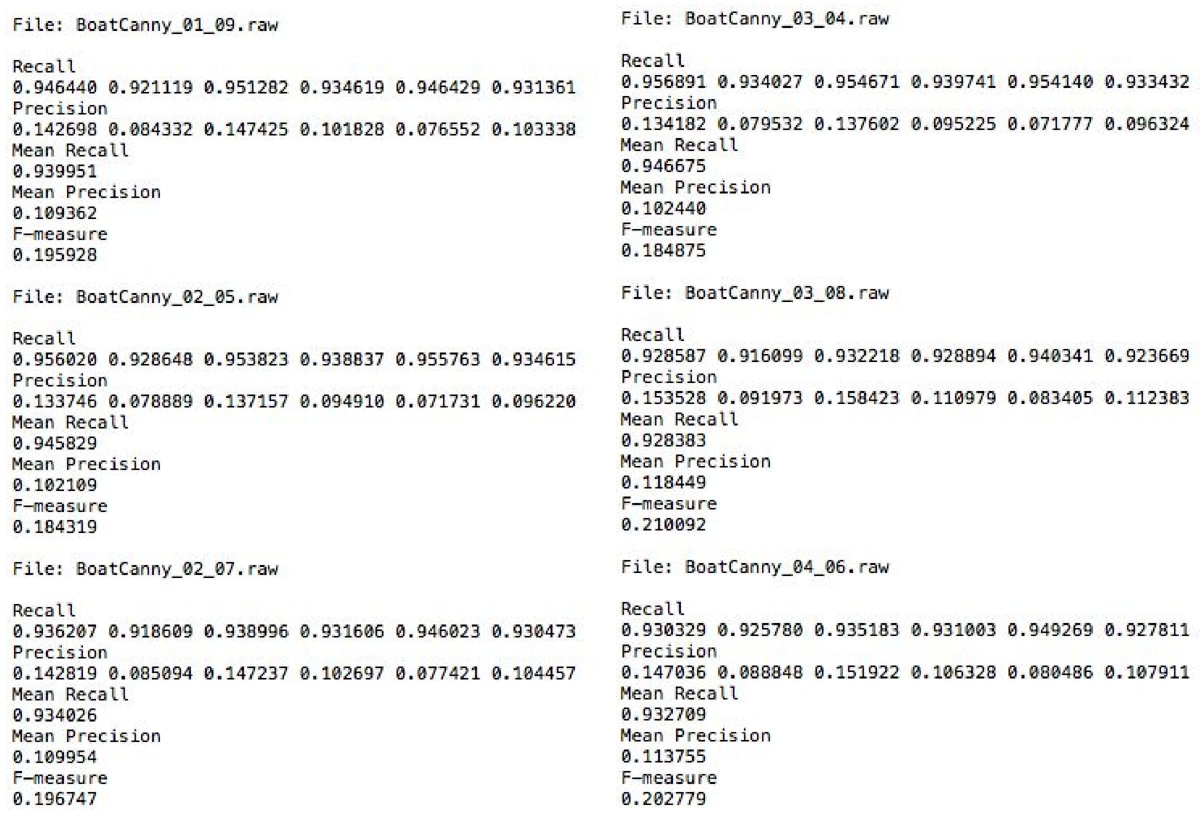

Castle with a threshold of 0.02
Castle with a threshold of 0.05
Castle with a threshold of 0.10
Castle with a threshold of 0.15
Castle with a threshold of 0.20
Castle with a threshold of 0.25

Fig 18. Castle Image with different thresholds. Top Left: 0.02. Top Right: 0.05. Center Left: 0.10. Center Right: 0.15. Bottom Left: 0.20. Bottom Right: 0.25.

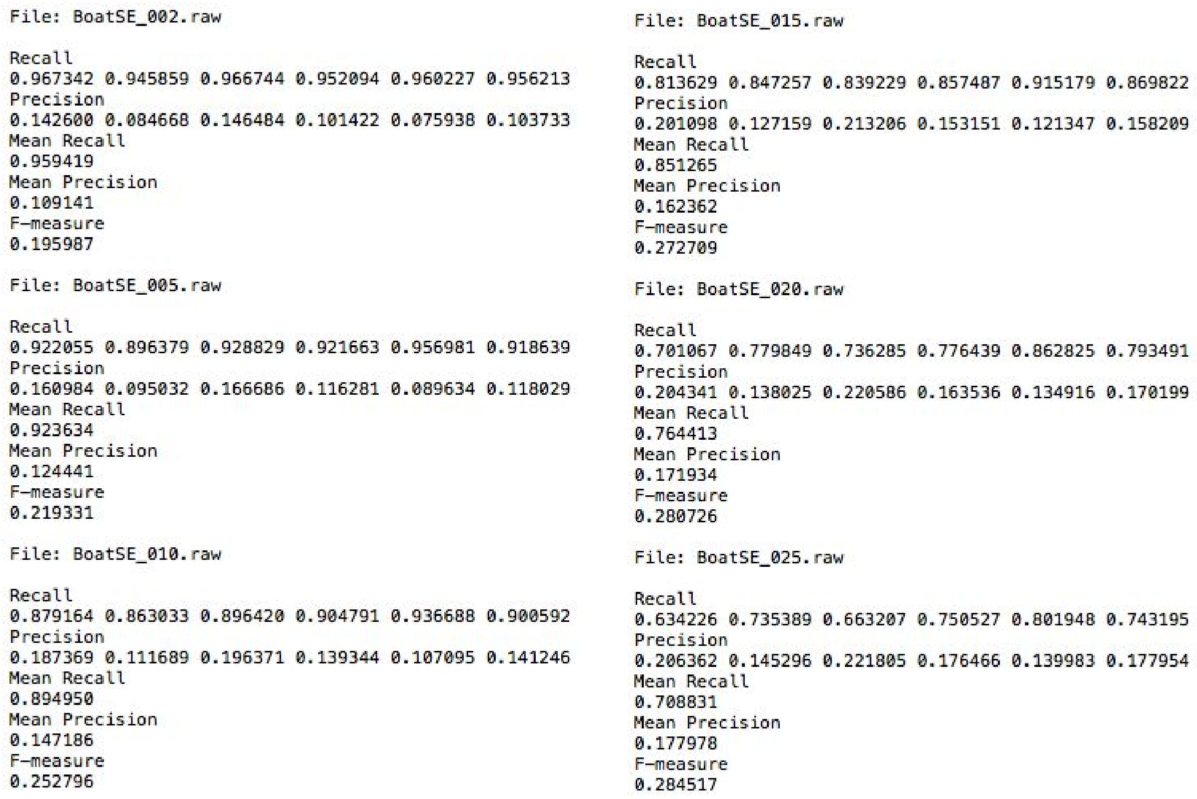
2.3.3 Performance Evaluation



```
File: BoatCanny_01_09.raw

Recall
0.946440 0.921119 0.951282 0.934619 0.946429 0.931361
Precision
0.142698 0.084332 0.147425 0.101828 0.076552 0.103338
Mean Recall
0.939951
Mean Precision
0.109362
F-measure
0.195928

File: BoatCanny_02_05.raw

Recall
0.956020 0.928648 0.953823 0.938837 0.955763 0.934615
Precision
0.133746 0.078889 0.137157 0.094910 0.071731 0.096220
Mean Recall
0.945829
Mean Precision
0.102109
F-measure
0.184319

File: BoatCanny_02_07.raw

Recall
0.936207 0.918609 0.938996 0.931606 0.946023 0.930473
Precision
0.142819 0.085094 0.147237 0.102697 0.077421 0.104457
Mean Recall
0.934026
Mean Precision
0.109954
F-measure
0.196747
```

```
File: BoatCanny_03_04.raw

Recall
0.956891 0.934027 0.954671 0.939741 0.954140 0.933432
Precision
0.134182 0.079532 0.137602 0.095225 0.071777 0.096324
Mean Recall
0.946675
Mean Precision
0.102440
F-measure
0.184875

File: BoatCanny_03_08.raw

Recall
0.928587 0.916099 0.932218 0.928894 0.940341 0.923669
Precision
0.153528 0.091973 0.158423 0.110979 0.083405 0.112383
Mean Recall
0.928383
Mean Precision
0.118449
F-measure
0.210092

File: BoatCanny_04_06.raw

Recall
0.930329 0.925780 0.935183 0.931003 0.949269 0.927811
Precision
0.147036 0.088848 0.151922 0.106328 0.080486 0.107911
Mean Recall
0.932709
Mean Precision
0.113755
F-measure
0.202779
```

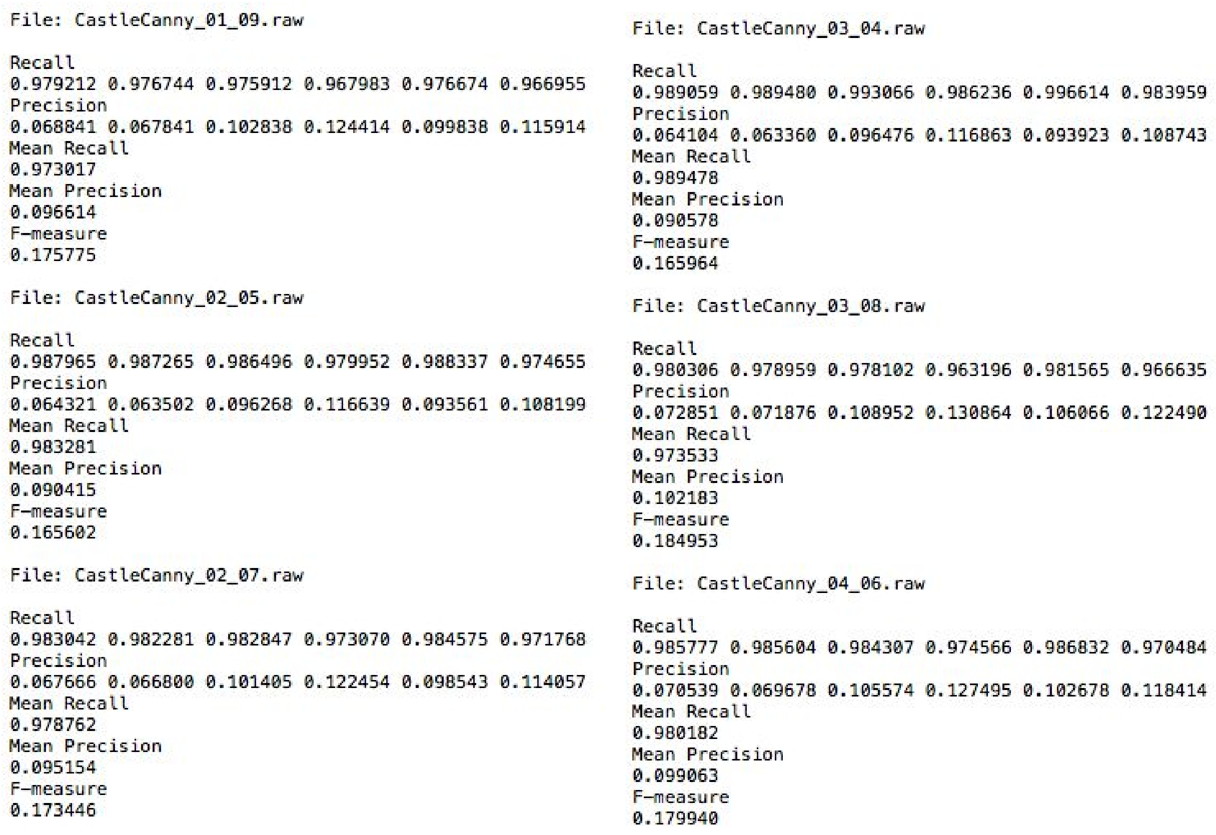
Fig 19. Evaluation data in percentages on the Canny edge maps from Boat.raw



```
File: BoatSE_002.raw

Recall
0.967342 0.945859 0.966744 0.952094 0.960227 0.956213
Precision
0.142600 0.084668 0.146484 0.101422 0.075938 0.103733
Mean Recall
0.959419
Mean Precision
0.109141
F-measure
0.195987

File: BoatSE_005.raw

Recall
0.922055 0.896379 0.928829 0.921663 0.956981 0.918639
Precision
0.160984 0.095032 0.166686 0.116281 0.089634 0.118029
Mean Recall
0.923634
Mean Precision
0.124441
F-measure
0.219331

File: BoatSE_010.raw

Recall
0.879164 0.863033 0.896420 0.904791 0.936688 0.900592
Precision
0.187369 0.111689 0.196371 0.139344 0.107095 0.141246
Mean Recall
0.894950
Mean Precision
0.147186
F-measure
0.252796
```

```
File: BoatSE_015.raw

Recall
0.813629 0.847257 0.839229 0.857487 0.915179 0.869822
Precision
0.201098 0.127159 0.213206 0.153151 0.121347 0.158209
Mean Recall
0.851265
Mean Precision
0.162362
F-measure
0.272709

File: BoatSE_020.raw

Recall
0.701067 0.779849 0.736285 0.776439 0.862825 0.793491
Precision
0.204341 0.138025 0.220586 0.163536 0.134916 0.170199
Mean Recall
0.764413
Mean Precision
0.171934
F-measure
0.280726

File: BoatSE_025.raw

Recall
0.634226 0.735389 0.663207 0.750527 0.801948 0.743195
Precision
0.206362 0.145296 0.221805 0.176466 0.139983 0.177954
Mean Recall
0.708831
Mean Precision
0.177978
F-measure
0.284517
```

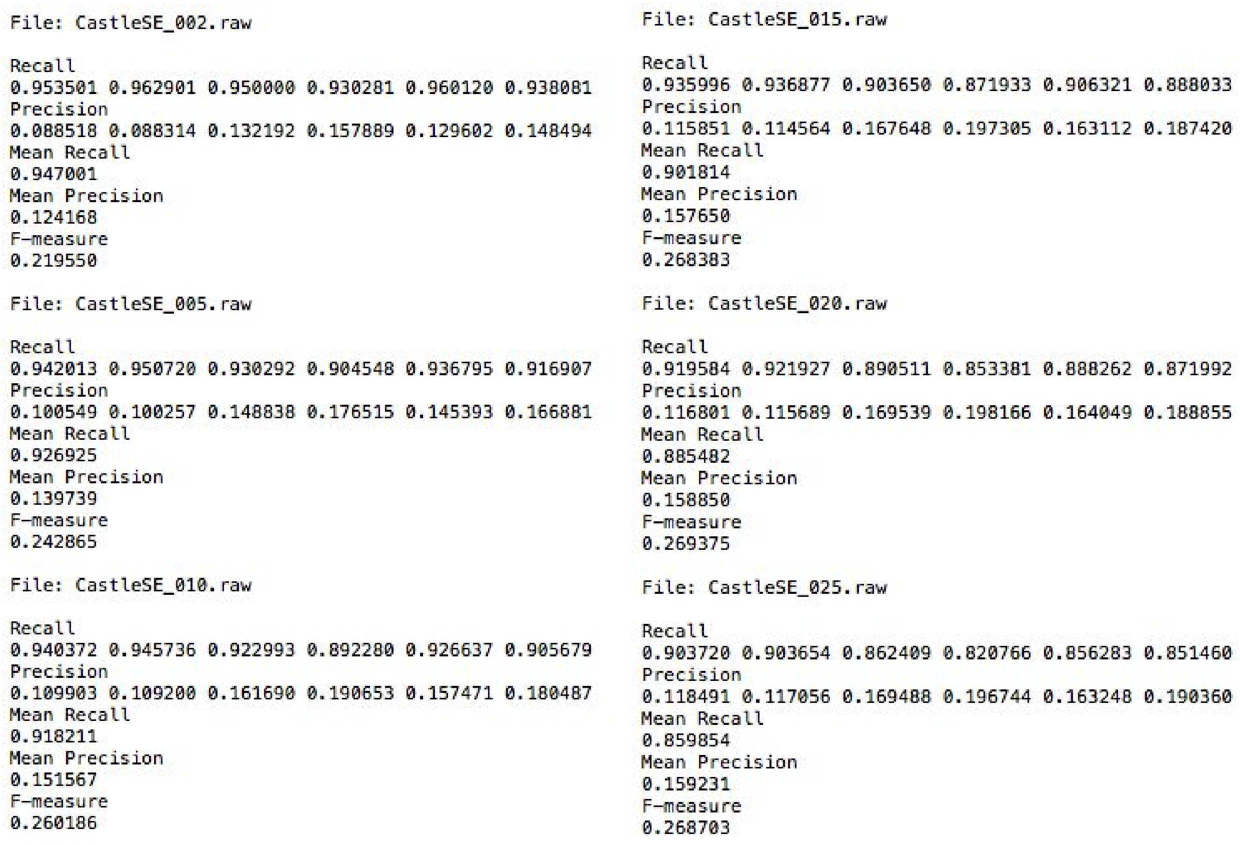
Fig 20. Evaluation data in percentages on the SE edge maps from Boat.raw



```
File: CastleCanny_01_09.raw

Recall
0.979212 0.976744 0.975912 0.967983 0.976674 0.966955
Precision
0.068841 0.067841 0.102838 0.124414 0.099838 0.115914
Mean Recall
0.973017
Mean Precision
0.096614
F-measure
0.175775

File: CastleCanny_02_05.raw

Recall
0.987965 0.987265 0.986496 0.979952 0.988337 0.974655
Precision
0.064321 0.063502 0.096268 0.116639 0.093561 0.108199
Mean Recall
0.983281
Mean Precision
0.090415
F-measure
0.165602

File: CastleCanny_02_07.raw

Recall
0.983042 0.982281 0.982847 0.973070 0.984575 0.971768
Precision
0.067666 0.066800 0.101405 0.122454 0.098543 0.114057
Mean Recall
0.978762
Mean Precision
0.095154
F-measure
0.173446
```

```
File: CastleCanny_03_04.raw

Recall
0.989059 0.989480 0.993066 0.986236 0.996614 0.983959
Precision
0.064104 0.063360 0.096476 0.116863 0.093923 0.108743
Mean Recall
0.989478
Mean Precision
0.090578
F-measure
0.165964

File: CastleCanny_03_08.raw

Recall
0.980306 0.978959 0.978102 0.963196 0.981565 0.966635
Precision
0.072851 0.071876 0.108952 0.130864 0.106066 0.122490
Mean Recall
0.973533
Mean Precision
0.102183
F-measure
0.184953

File: CastleCanny_04_06.raw

Recall
0.985777 0.985604 0.984307 0.974566 0.986832 0.970484
Precision
0.070539 0.069678 0.105574 0.127495 0.102678 0.118414
Mean Recall
0.980182
Mean Precision
0.099063
F-measure
0.179940
```

Fig 21. Evaluation data in percentages on the Canny edge maps from Castle.raw



```
File: CastleSE_002.raw

Recall
0.953501 0.962901 0.950000 0.930281 0.960120 0.938081
Precision
0.088518 0.088314 0.132192 0.157889 0.129602 0.148494
Mean Recall
0.947001
Mean Precision
0.124168
F-measure
0.219550

File: CastleSE_005.raw

Recall
0.942013 0.950720 0.930292 0.904548 0.936795 0.916907
Precision
0.100549 0.100257 0.148838 0.176515 0.145393 0.166881
Mean Recall
0.926925
Mean Precision
0.139739
F-measure
0.242865

File: CastleSE_010.raw

Recall
0.940372 0.945736 0.922993 0.892280 0.926637 0.905679
Precision
0.109903 0.109200 0.161690 0.190653 0.157471 0.180487
Mean Recall
0.918211
Mean Precision
0.151567
F-measure
0.260186
```

```
File: CastleSE_015.raw

Recall
0.935996 0.936877 0.903650 0.871933 0.906321 0.888033
Precision
0.115851 0.114564 0.167648 0.197305 0.163112 0.187420
Mean Recall
0.901814
Mean Precision
0.157650
F-measure
0.268383

File: CastleSE_020.raw

Recall
0.919584 0.921927 0.890511 0.853381 0.888262 0.871992
Precision
0.116801 0.115689 0.169539 0.198166 0.164049 0.188855
Mean Recall
0.885482
Mean Precision
0.158850
F-measure
0.269375

File: CastleSE_025.raw

Recall
0.903720 0.903654 0.862409 0.820766 0.856283 0.851460
Precision
0.118491 0.117056 0.169488 0.196744 0.163248 0.190360
Mean Recall
0.859854
Mean Precision
0.159231
F-measure
0.268703
```

Fig 22. Evaluation data in percentages on the SE edge maps from Castle.raw

2.4 Discussion

2.4.1 Canny Edge Detector

From fig 15 and 16, we see that if the thresholds are closer in value then the output is more likely to contain edges of non objects, which deteriorate the edge map accuracy since we are not getting the expected result. Additionally, if the low threshold value is too small, and the high threshold value is too large then the edge map is also more likely to detect edges in the wrong areas. Due to this, a good balance between these two threshold values should give the best results. From the edge maps, visually we can see that we obtain the best result with the threshold values of 0.3 and 0.8. This edge map, looks a little more cleaner and ignores edges on some irrelevant areas that the other images didn't. The same can be said with both input images (castle and boat).

However, when comparing the most accurate edge map from both input images, we see that the Canny edge detection overall works better for the castle image. Among these two output edge maps, it is easier to tell the location of the image in the castle image rather than the boat one.

Also, despite trying out different threshold values we see that it's not duable to find just the edges of the objects. The edge maps located the objects edges and also some edges on areas where the texture/pixel-values significantly changed. The algorithm recognizes these subtle changes as a new object and hence finds its edges. Hence why the image appears noisy. Additionally, the threshold values difference have a negative correlation with the likelihood of detecting texture changes as edges.

Lastly, note that there's a vertical edge line found on the left side of the castle edge maps. This is because in the original image, that area was blacked out.

2.4.2 Contour Detection with Structured Edge

If we see fig 17 and 18, the edge maps using structured edge are visually more pleasing than the ones from fig 15 and 16 which are edge maps found from using the Canny Edge Detection. We can see that the structured edge maps look less noisy since they don't take into account texture as much as when using the canny one.

Also, within the results from using different threshold values we see that the smaller the threshold value the more likely it is to find edges on irrelevant areas. Among these outputs, the best one seems to occur when the threshold is 0.15 or 0.20.

2.4.3 Performance Evaluation

For this part we only used one threshold which was set to 0.50, hence why there's only one output per evaluation. By comparing the F-measures of fig 19 and 20 (boat.raw), and fig 21 and 22 (castle.raw), we see that among all edge detectors we obtain the best results with the Structured Edge detector. The edge maps from the Canny Edge Detector resulted in a higher recall value but a lower precision value, while the Structured Edge Detector produces a values of recall and precision with a smaller difference. In summary, the Canny detector is able to find the edges but fails when there's noise, and also detects weak edges which may not be relevant. While the Structured Edge detector appears to ignore weak edges and can handle noise.

Intuitively we can say that castle.raw will have a higher F-measure, this is because unlike boat.raw, there are fewer areas that the detector can compute as edges. Furthermore, there are fewer objects and less details.

It is not possible to get a higher F-measure if precision is significantly higher or vice versa. This is because of how F-measure depends on Recall and Precision:

$$F = 2 \times \frac{P \times R}{P + R}$$

Due to this, if we increase one the other will decrease since they have an inverse relationship.

Now if Precision + Recall = constant → F = 2*P*R/constant
From this we can say that F depends on P*R since these are the only variables in the equation.
$(P - R)^2 \geq 0$ ––– $P^2 + 2PR + R^2 \geq 4PR$ ––– $(P + R)^2/4 \geq PR$
Therefore the maximum value of PR *is* $(P + R)^2/4$
$(P + R)^2/4 = PR$ ––– $P^2 + 2PR + R^2 = 4PR$ ––– $(P - R)^2 = 0$ ––– $P = R$ (QED)
Lastly, we can say that if the sum of Precision and Recall is constant then we obtain the maximum F-measure when Precision = Recall.

Problem 3 : Salient Point Descriptors and Image Matching

3.1 Motivation

In this problem we will deal with extracting salient point descriptors from images, and also perform image matching with these feature points. Salient points are points of interest in the image that can be used for future processing. There are many ways to obtain these points, in this problem we will use the SIFT technique (Scale Invariant Feature Transform) which can extract invariant feature points which are widely used in image recognition, image retrieval, image mosaic, etc [5], and the SURF technique (Speed up Robust Features) which is similar to the SIFT technique but is performed in a multi-scale space.[5]

3.2 **Approach**

3.2.1 Extraction and Description of Salient Points

For this part we use the SIFT function from the open source library VLFeat. We pass the images onto the function vl_sift with a peak threshold of 10 and edge threshold of 100 in order to detect only the strongest points.

On the other hand SURF is an approximate of SIFT and works equally as well [1]. We ran the function from matlab on the desired images and obtain the salient points. Similarly as in SIFT, we set it up to obtain the 100 strongest points.

3.2.2 Image Matching

Similarly to the 3.2.1, we start off by finding the individual salient points/features whether it is with SIFT or SURF (on both images). Then we proceed to match these points with the following functions: vl_ubcmatch (SIFT). And then draw lines between these two pair of points in order to see the final image matching output. The same threshold values were used as in part 3.2.1 (SIFT). For the SURF technique, only the 50 strongest points were shown. Additionally, certain parameters were changed in order to have some matching points showing.

In detectSURFFeatures the MetricThreshold was set to 10 in order to increase the number of features, and also in matchFeatures the MatchThreshold was set to 20 to decrease the tolerance of the feature matching.

3.2.3 Bag of Words

For this part, we use the k-means clustering to extract the SIFT features from three images (SUV, truck and convertible_1). This will allow us to form a codebook with k=8 bins, which will be used to create a codewords (bag of words) that we can match with other images.

Note that there is a logical error in the function computeBoV, the function always fails to start because of this if statement which is always true:

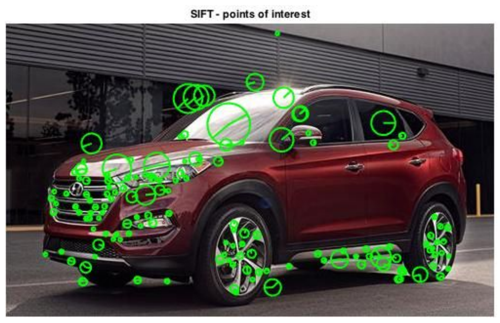
```
if(Nind < 2 || Nind >3)
```

This was changed to:

```
if(Nind<0 && Nind >2)
```

3.3 **Results**

3.3.1 Extraction and Description of Salient Points



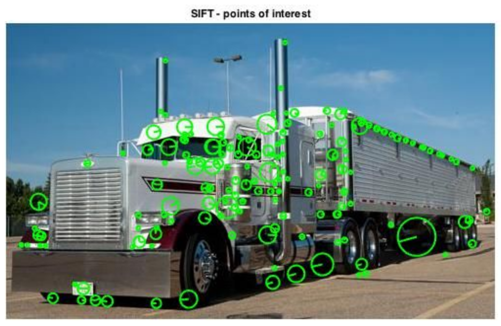
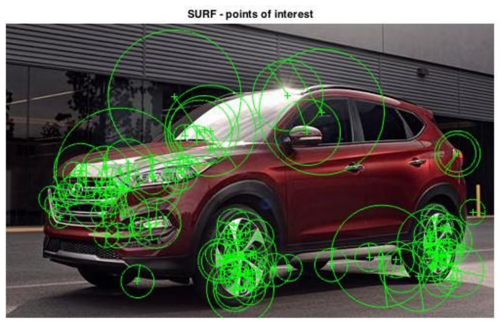


Fig 23. Salient points of SUV and Truck images with the SIFT technique



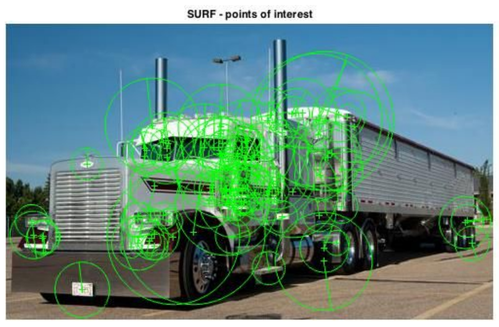
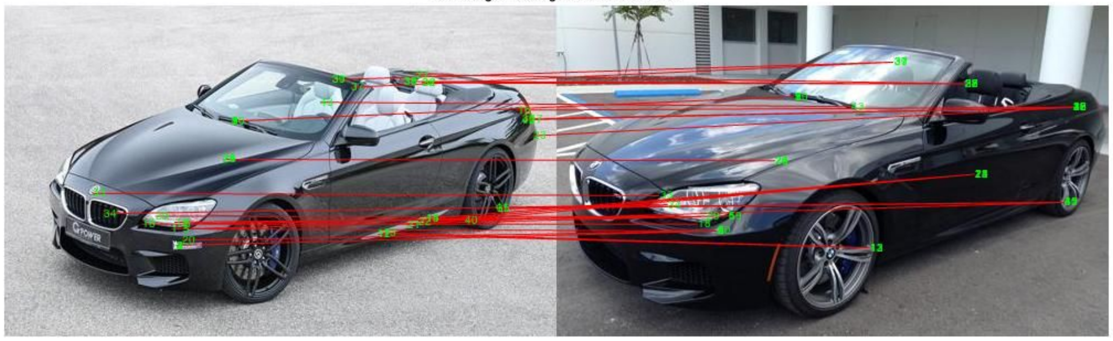


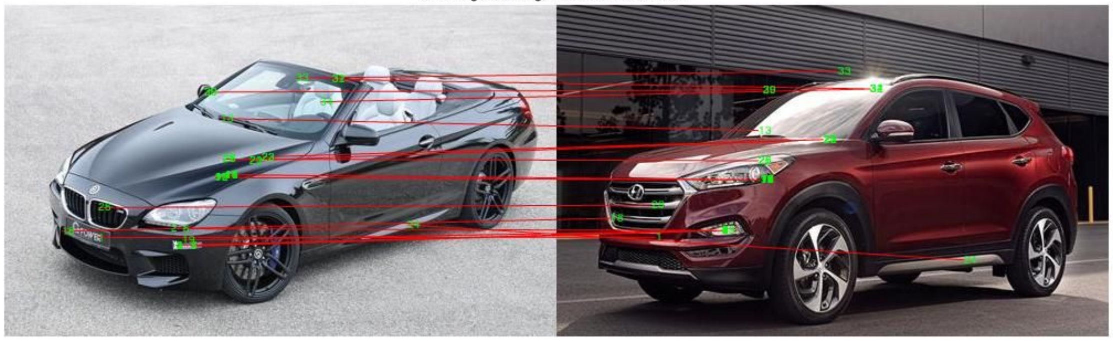
Fig 24. Salient points of SUV and Truck images with the SURF technique

3.3.2 <u>Image Matching</u>

SIFT Image Matching for convertible x.raw



SIFT Image Matching for convertible 1 and SUV



SIFT Image Matching for convertible 1 and truck

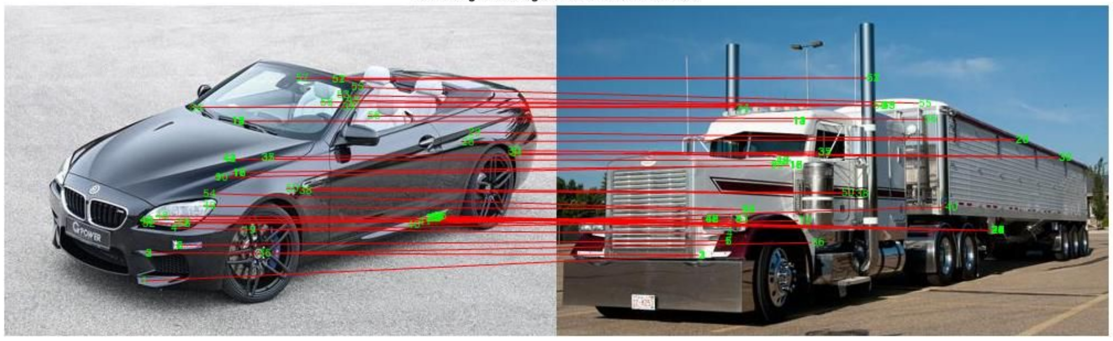


Fig 25. Image Matching using the SIFT technique

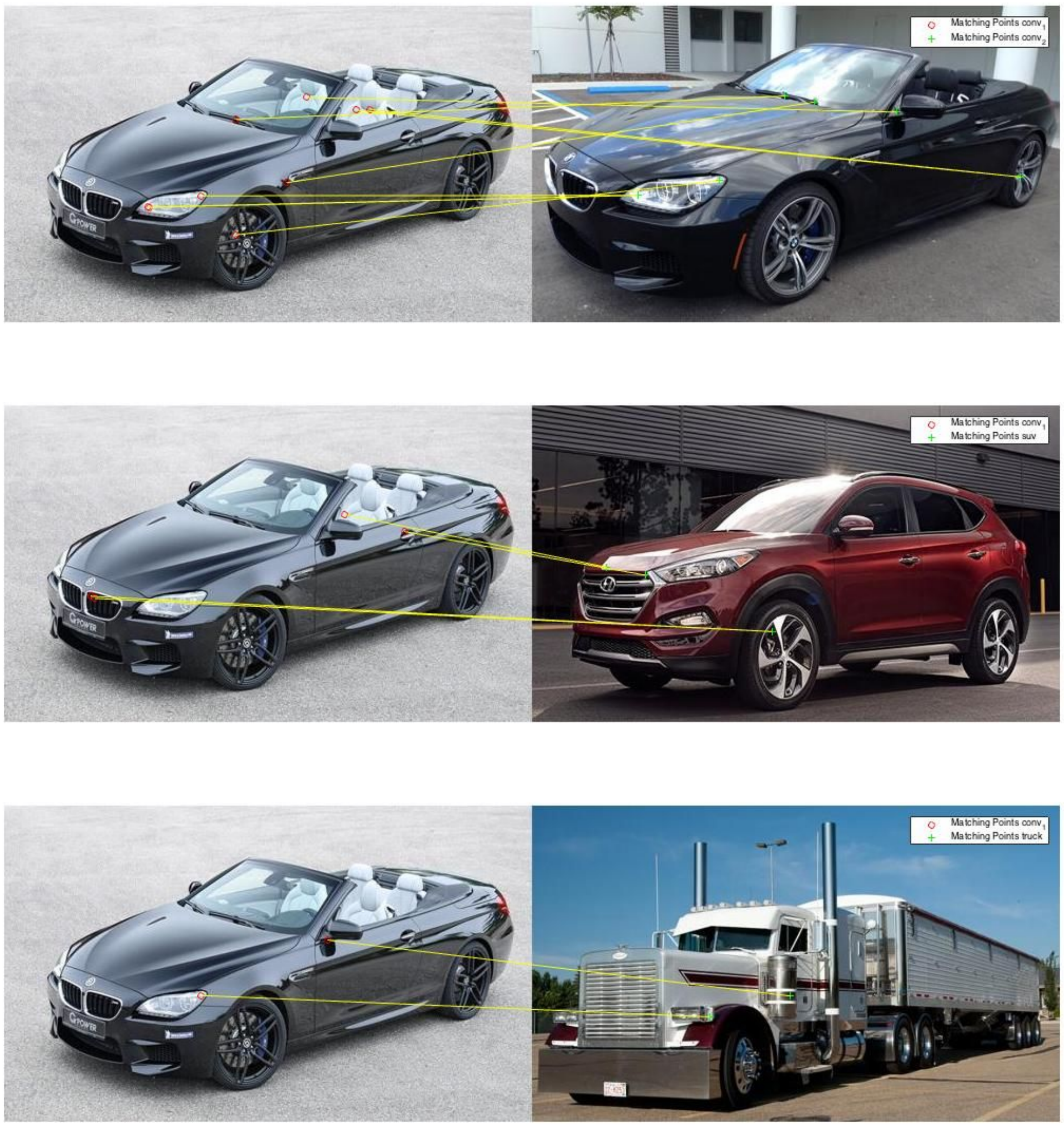


Fig 26. Image Matching using the SURF technique

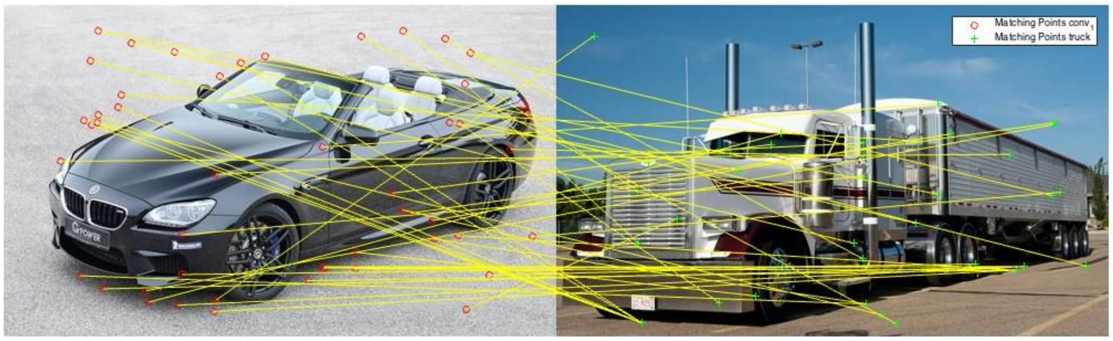


Fig 27. SURF technique feature matching with all the features

3.3.3 Bag of Words

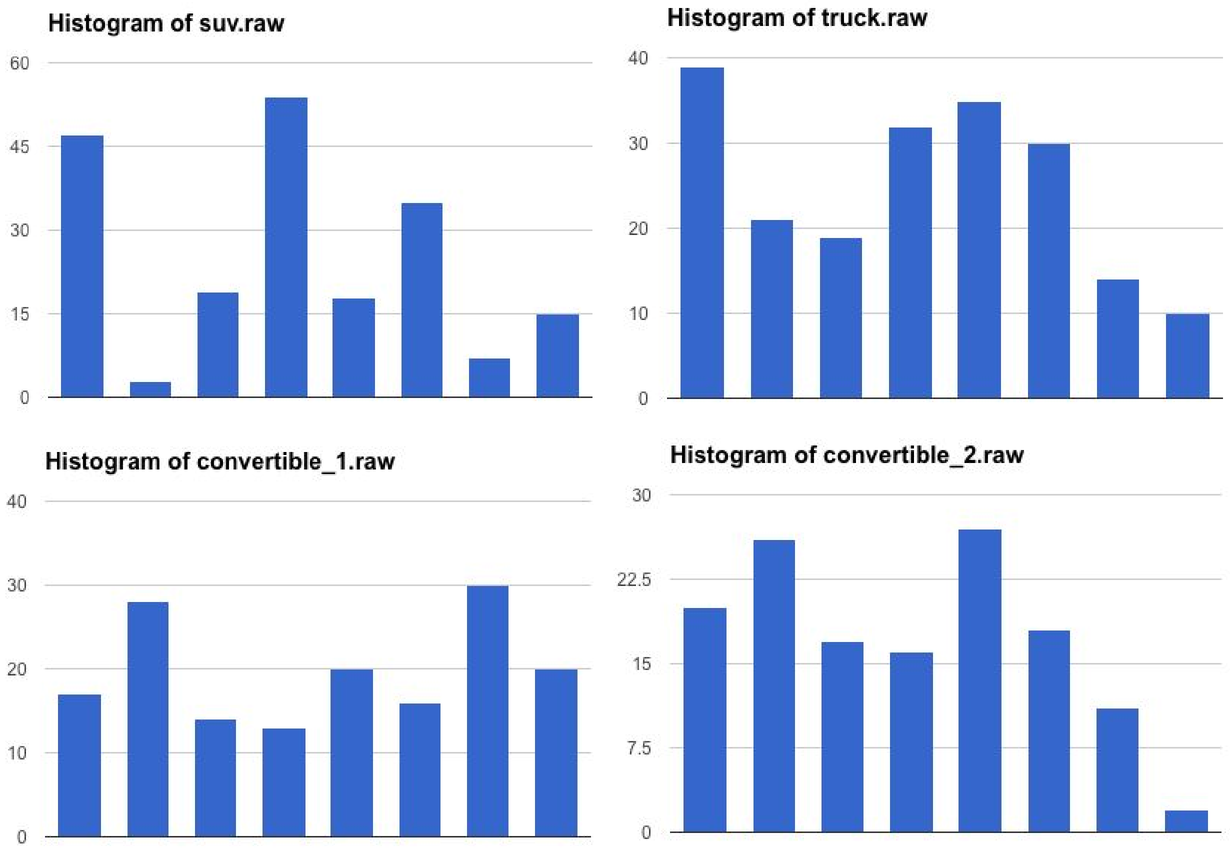


Fig 28. Bag of Words histograms of the codewords (k=8) of each image.

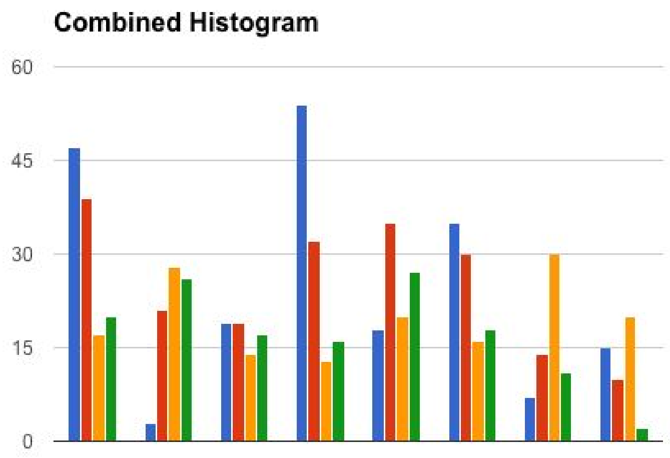


Fig 29. Combined Histograms. Blue: suv, Red: truck, Yellow: convertible_1, Green: convertible_2

CLASS	NAME
1	SUV
2	Truck
3	Convertible_1

Fig 30. Training data for the classification

CLASS	NAME
3	convertible_2

Fig 31. Classification obtained of the testing image

3.4 **Discussion**

3.4.1 Extraction and Description of Salient Points

From fig 23 and 24, we see that the salient points show a better match and more feature points when using the SIFT technique. This can be seen when comparing fig 23 (SIFT) with fig 24 (SURF). Also, we can see that SIFT is able to find more points surrounding the object edges and corners than SURF. Therefore, we can say that SIFT performs better if we're also interested in salient points on the edges of the image, while SURF performs better when we are far more interested in areas of interest that may not lie on an edge.

3.4.2 Image Matching

While performing image matching, the algorithm selects invariant features matching. And we can see that the SIFT technique has a stronger detection despite viewpoint change, and thus it is more effective than when using the SURF technique. Additionally, note that far fewer matching points have been found when using SURF, however, some of the matching points do were not correctly matched.

When looking at fig 25 (SIFT), we can see that when using the same threshold values for all three comparison we obtain different matching salient point pairs:

- For the convertible_1 and convertible_2 we obtain 43 matching points
- For the convertible_1 and suv we obtain 34 matching points
- For the convertible_1 and truck we obtain 58 matching points

From these outputs we see that we obtain the best result for convertible_1 and truck. This result can be based on the viewing angle difference, the larger the difference is the least matching points it's able to find. Nonetheless, if we see at all the matching points, we see that practically all are salient points from the vehicles which is what is wanted, since we don't care for the backgrounds.

In fig 26 (SURF), we chose th50 strongest salient point matches which were computed by using the same threshold value in all of the images. In contrast in fig 27, we can see if we don't do so then the matching points are not mostly correct. Additionally, some of the salient points are on the background. Overall, we can see that the SIFT technique performs better than the SURF technique for image matching.

3.4.3 Bag of Words

After obtaining the 8 codewords, we find a histogram of how likely each codeword has been found within the images. This provides us with the information to understand and see if there are any similar images. To do so, we use the data gathered from the histograms (fig 28) and the centroid clusters obtained which are applied onto an SVM classifier. From this, we can see that the classifier label convertible_2 as the same from convertible_1 (fig 30 and 31). Additionally, all histograms from all four images are plotted together for an easier visualization (fig 29).

References

[1] Discussion slides, assignments, lectures, and class notes.
[2] https://en.wikipedia.org/wiki/Canny_edge_detector
[3] https://en.wikipedia.org/wiki/Edge_detection
[4] Dollár, Piotr, and C. Lawrence Zitnick. "Structured forests for fast edge detection." Computer Vision (ICCV), 2013 IEEE International Conference on. IEEE, 2013.
[5]https://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf