Table of Contents

LIST OF FIGURES

LIST OF TABLES

Amruta Kulkarni
USC ID: 6914970827
Email: arkulkar@usc.edu
Date: 11 Oct 2015

EE 569 HOMEWORK #2

1. Problem #1: Texture Analysis and Classification

1.1 Texture Classification: Two Classes + Minimum Mean Distance Classifier

1.1.1 Motivation

Texture Analysis is an important concept in various applications such as object recognition and image retrieval. It is mainly used in image processing and computer vision. For a machine to be able to classify textures, we first need to train the machine by providing labeled input images of different textures. After the training stage, we can check the performance of the machine by testing it on data that it has never seen before. The idea of first extracting the texture features and then classifying them into different categories has been implemented in this problem.

1.1.2 Approach and Implementation

1.1.2.1 Feature Extraction

- We have two sets of image data with us, labeled and unlabeled.
- The labeled dataset has two types of images namely grass image and straw image
- The main aim is to be able to determine the type of images in the unlabeled data set.
- Every image has a certain texture and this texture comprises of a set of specific features.
- Thus, to determine the texture of an image, we first need to extract the features.

- Using the combination of Laws' filters, we can do this feature extraction.
- But first, in order to remove the redundant dc component that might be present, we need to perform pre-processing.
- In this, a local window is considered around each pixel and then the mean of that window is subtracted from the central pixel.
- We have a set of 25 Laws' Filters and applying every filter to an image will result in a new image with a specific feature.
- Thus, when we apply these Laws' filters to all the labeled and unlabeled images, each image gets converted to 25 new images corresponding to 25 features that they represent.
- The images that we are using just have one texture type and because of this we can do global averaging on each image to get a single value of that corresponding feature.
- Thus, we now have 25 different feature values instead of 25 different images for every image.

1.1.2.2 Minimum Distance to Class Means Classification

- After performing feature extraction, we can start the process of classification by using training and test data
- We have a set of labeled images of grass and straw and we also have the set of 25 features for each of these images.
- After separating the labeled data into two classes, the mean and variance of the 25 dimensional features can be calculated for both the classes (straw and grass).
- We now have 2 values of mean and variance, each belonging to grass class and straw class respectively.
- By using these mean and variance values we can classify all the images based on their distance from the two class means.
- While calculating the class variance, it has been assumed that the two classes are independent which leads to null values of covariance between classes.
- Thus we only have the class variance and class covariance is not considered.
- Now, distance of every 25 dimensional feature from either of the class mean is calculated using Mahalanobis Distance by using the formula below:

$$d_M(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)}$$

$$= \sqrt{\begin{bmatrix} x_1 - y_1 & x_2 - y_2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \end{bmatrix}}$$

$$= \sqrt{\begin{bmatrix} \frac{x_1 - y_1}{\sigma_1^2} & \frac{x_2 - y_2}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \end{bmatrix}}$$

$$= \sqrt{\frac{(x_1 - y_1)^2}{\sigma_1^2} + \frac{(x_2 - y_2)^2}{\sigma_2^2}}$$

- In our case, ‘X’ will be the 25 dimensional feature corresponding to the image being classified and ‘Y’ will be the 25 dimensional mean feature of either grass class or the straw class.
- Also, σ^2 is the 25 dimensional variance value of either grass or straw class.
- So, the Mahalanobis distance of all the labeled and unlabeled images will be calculated from both the class mean values of straw and grass.
- If the distance from straw class mean is lesser than that from the grass class mean, then the image will be classified as straw, and otherwise it will be classified as grass.
- Thus, after classification, we assign a separate label to all the training and test images.
- For the training i.e. labeled data, we can count the number of misclassified images by comparing the new labels and the labels that were already there, and this will give us the error rate on the training data.
- We can initially give true labels to the test data by observing every image, and then by comparing these labels with the ones assigned by the minimum mean distance classifier, we can get the error rate for test data.

1.1.2.3 Feature Reduction

- Until now, we were dealing with 25 features of every image, but in reality all these features do not necessarily contribute anything to the image.
- Thus we can reduce the number of features to a value in such a way that every feature is different and uniquely important.
- In our case, we want to reduce 25 features to a single feature.
- Feature reduction can be obtained by various methods but the ones that are being used in our case are Principal Component Analysis and Linear Discriminant analysis.
- Principal Component Analysis is an orthogonal linear transform that converts the data into a set of linearly uncorrelated variables that are nothing but the principal components.
- PCA has the following steps:

  Step 1: Obtain the data and normalize it to have zero mean
  Step 2: Calculate the covariance matrix and get the eigenvectors and eigenvalues
  Step 3: Choose the eigenvector with the highest eigenvalue resulting into the principal component of the dataset. The eigenvectors that we want to retain are the feature vectors.
  Step 4: Multiply the dataset with the feature vectors.
  (In our case we will have only one feature vector)

- First, we perform PCA on the training data to get the feature vector.
- Using the same feature vector obtained from the training data, we perform PCA on the test data.
- In this manner, the entire data has now been reduced to features with 1 dimension instead of 25 dimensions.
- Another way of doing feature reduction is by using Linear Discriminant analysis that is different from PCA, since unlike PCA, LDA considers class labels along with the data.
- Linear Discriminant Analysis is used to find the linear combination of features which best explain the data.
- LDA has the following steps:

  Step 1: Compute the mean vectors for different classes

Step 2: Calculate the scatter matrices for inter-class scatter and intra-class scatter and get the eigenvectors and eigenvalues

Step 3: Sort the eigenvectors in decreasing eigenvalues order and choose the eigenvectors that we want to retain. These are the feature vectors.

Step 4: Multiply the dataset with the feature vectors
(In our case we will have only one feature vector)

- Similar to PCA, we first perform LDA on the training data to get the feature vectors and then perform LDA on test data using the same feature vectors.
- Thus, we now have two reduced datasets obtained from applying PCA and LDA respectively.

1.1.2.4 Classification with two reduced Feature sets

- We perform the same steps on the reduced datasets for classification as we did earlier using all 25 features.
- So, we again need to calculate the two class mean and variance values and then take the Mahalanobis distance of every image feature value from both the class means.
- Depending upon which distance is smaller, we assign the class labels.
- By comparing the true class labels with the ones given by the classifier, we can get the error rates for training as well as test data sets.

1.1.3 Results

The above approach has been implemented using C++ and OpenCV.
The error rates obtained for training as well as test datasets using different techniques are as follows:

Methods used	Error on training data	Error on test data
No feature Reduction	0 %	0 %
Reduction using PCA	0 %	0 %
Reduction using LDA	0 %	0 %

Table No. 1.1 Error rates for training and test data

1.1.4 Discussion

We performed feature reduction to represent every image using just 1 feature instead of using 25 features. Feature reduction is mainly done to reduce data to low dimensionality so that it is easy to visualize and also to store. In our case, all 25 features are not equally important because all the features do not contribute anything new to the image. Moreover, these features can be represented by their linear combinations in lesser number of dimensions. Feature reduction is mainly done to beat the 'Curse of Dimensionality'. Thus, from the results, we can conclude that it is not necessary to use all 25 features as feature reduction gives better results. Now, in our problem, the error rates of all the three techniques are 0% for both the datasets. Thus, all the three techniques have given the best results. But, in reality, LDA has the best performance as it considers class labels during feature reduction. The worst performance will be when we are using all the 25 features because the computational efficiency of this technique is very low. No feature reduction method unnecessarily considers all the features for classification even though some of the features are obsolete.

1.2 Advanced Texture Classification : Multi-Class + SVM

1.2.1 Motivation

Many times, we come across problems where we need to classify instances into more than two classes. Machine Learning treats this as multi-class or multinomial classification. There are various methods of dealing with multi-class classification. One of them is to reduce it to a binary classification by using One-vs.-Rest. Thus, the multi-class classification can now be treated as binary classification. We are going to use the same strategy to classify images into four different classes of grass, straw, sand and leather by using binary classification for every class. This classification has been performed by using minimum distance to class means classifier as well as Support Vector Machines.

1.2.2 Approach and Implementation

1.2.2.1 Feature Extraction and Feature Reduction

- In this problem we have 48 images of each class namely grass, straw, sand and leather.
- We are going to use the one vs. rest type of binary classification for this multi-class problem.
- Thus, every classifier will classify images that belong to a certain class against images that do not belong to that class, e.g. Grass vs. Non-grass classifier
- For the training set of each class, we will consider 36 images that belong to that particular class and 12 images from each of the remaining 3 classes.
- For the testing set, we will consider the remaining 12 images of that same class that were not used for training and 4 images each from the other 3 classes.
- By applying Laws' filters, we will extract 25 features from each image and then we will perform feature reduction on this high dimensional feature space.
- For feature reduction, Principal Component Analysis has been used.

- We need to reduce the feature dimension from 25 to 3 because we have 4 different classes and it is best to reduce the feature dimension to No. of classes – 1.
- The steps for Principal Component Analysis have been mentioned in section 1.1.2.3 of the previous problem.
- Thus, we now have 72 images in the training set of every class and every image has 3 features.

1.2.2.2 Minimum Distance to Class Means Classification

- Now, let us consider that we want to classify grass images. In the training set we have 36 grass images and 36 non-grass images and these form the 2 separate classes whose mean and variance values need to be calculated.
- By applying this to all the four classifiers, we can compute the mean and variance of the two classes corresponding to 4 different classifiers respectively (grass vs. non-grass, straw vs. non-straw, sand vs. non-straw and leather vs. non-leather).
- Now, we can calculate the distance of all the images (training and test), of a particular classifier, from the two class means.
- We have again considered Mahalanobis distance that uses the class variance for computing the distance.
- Depending upon the distance values we assign labels to every image of the training and test data set just like we did in problem 1.
- By comparing the true labels with the labels assigned by the classifier, the error rate for training as well as test datasets of all the 4 classifiers can be computed.
- This approach has been implemented using C++

1.2.2.3 Support Vector Machines

- A Support Vector Machine maps the data points in space in such a way that the categories that they belong to can be separated with a distinct gap that is as wide as possible.
- SVM can efficiently work in multi-dimensions with the help of hyperplane, which it uses to classify the data that might or might not be linearly separable.

- The following figure will make the idea more clear:

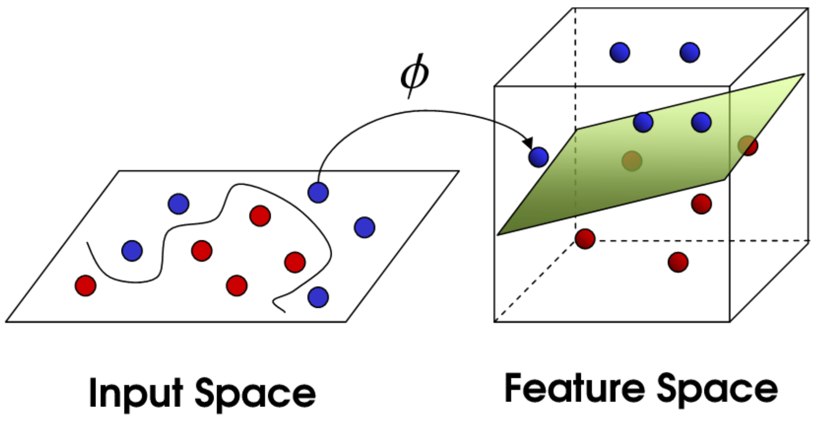

Figure 1.1 SVM Concept [1]

- We can see how SVM mapped the data into two classes that are well divided by using a hyperplane.
- The nearest data points to the hyper plane on both sides are the support vectors that define the margin of SVM.
- In our problem, we are using Support Vector Machines to classify the training and test data more accurately.
- We first apply SVM to the training data to determine the support vectors and the coefficients.
- By using the parameters obtained during the training stage, we apply the same SVM model to the test data.
- Thus, we have performed binary classification for all the images in all the 4 classifiers by using Support Vector Machines.
- This approach has been implemented using libSVM in MATLAB.

1.2.3 Results

The following table shows the error rates for training and test datasets of all 4 classifiers:

1) Minimum Distance to Class Means Classification:

Type of Classifier	Error on training data	Error on test data
Grass vs. Non-grass	1.389 %	0 %
Straw vs. Non-straw	5.556 %	25 %
Sand vs. Non-sand	2.778 %	4.167 %
Leather vs. Non-Leather	11.112 %	0 %

Table No. 1.2 Error rates for training and test data using minimum distance to class means classification

2) Classification using Support Vector Machines:

Type of Classifier	Error on training data	Error on test data
Grass vs. Non-grass	0 %	0 %
Straw vs. Non-straw	0 %	0 %
Sand vs. Non-sand	0 %	0 %
Leather vs. Non-Leather	0 %	0 %

Table No. 1.3 Error rates for training and test data using Support Vector Machines

1.2.4 Discussion

As we can see from the results, SVM gives more accurate results than the minimum distance to class means classifier. In minimum distance to class means classifier, the decision boundary is defined by the locus of hyperplane that is midway between the two class means and is orthogonal to the line connecting them. In SVM, a hyperplane is chosen in such a way that the distance from it to the nearest data point on each side be maximized. Moreover, in SVM, data points are considered as N dimensional vectors and it tries to separate the data using n-1 dimensional hyperplane. Now, for leather vs. non-leather classifier, the training error for minimum distance classifier is 11.112 % whereas for SVM is 0%. The decision boundaries for the two classifiers are as shown in the figures (generated using MATLAB) below:

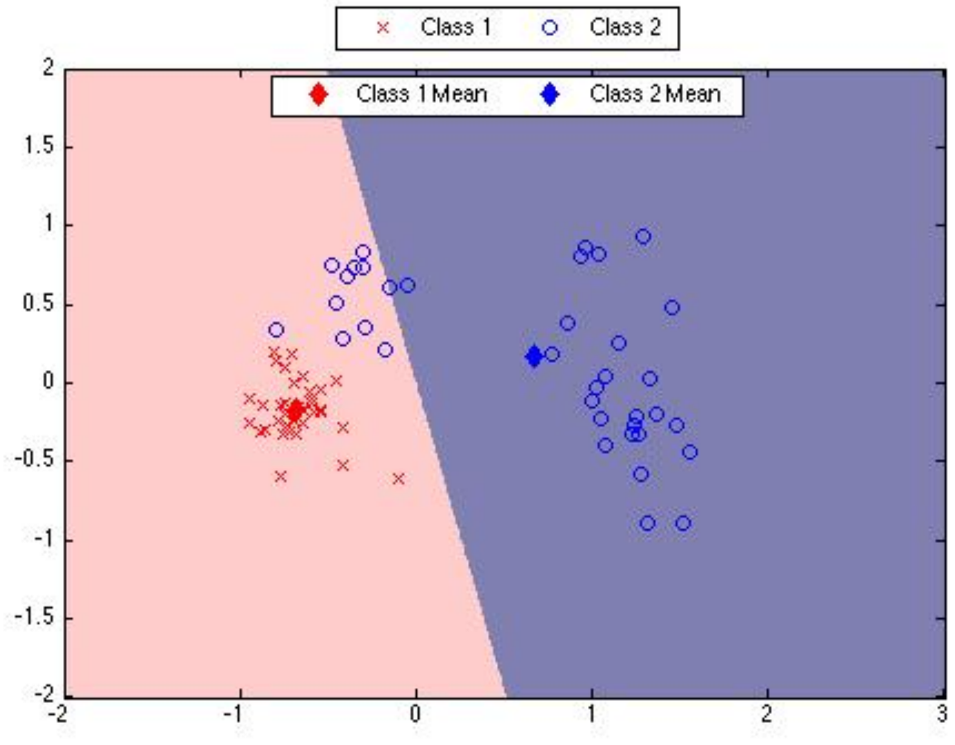

Figure 1.2 Minimum Distance classifier decision boundary of the first two features of training data for Leather vs. Non-leather classifier (blue and red diamond are the class mean)

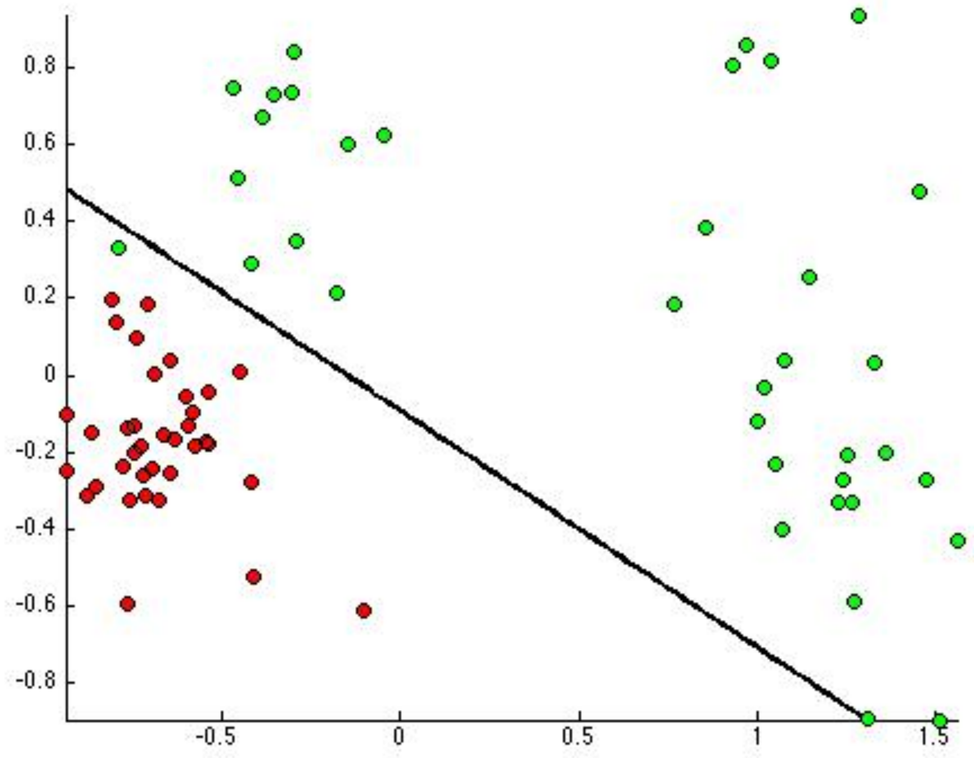

Figure 1.3 SVM decision boundary of the first two features of training data for Leather vs. Non-leather classifier

Thus, we can see how SVM adjusts its decision boundaries to achieve maximum accuracy. In this problem, we are using linear kernels. SVM can perform even better by using other non-linear kernels like gaussian that help in classifying data that is not linearly separable. This briefly explains how SVM is better than the minimum distance to class means classifier.

2. Problem #2: Edge Detection

2.1 Sobel Edge Detector and Non-Maximal Suppression

2.1.1 Motivation

Every image contains certain edges that are a result of sharp discontinuities of intensity values. Edge Detection is mainly done to extract the object boundaries or to preserve the structural properties of an image. There are various edge Detection techniques with different approaches like differential edge detection, phase congruency-based edge detection etc. In this problem, we are going to consider Sobel edge detector that performs gradient operation on the image along X and Y direction. To get thin edges, we are also going to perform non-maximum suppression on the edge maps that are obtained after thresholding.

2.1.2 Approach and Implementation

- Sobel Edge detectors are applied to gray scale images to extract the image boundaries and hence we first need to convert the color images into gray scale images.
- The formula for this color to gray scale conversion is
  0.2126 R + 0.7152 G + 0.0722 B

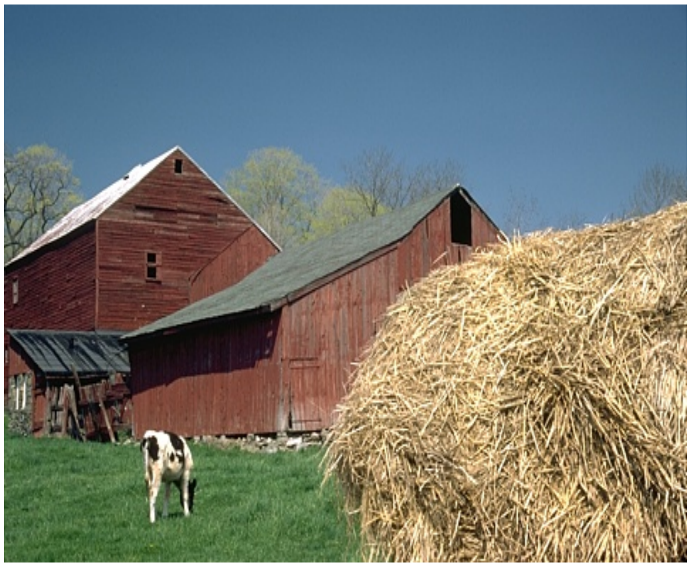
Figure 2.1a Colour

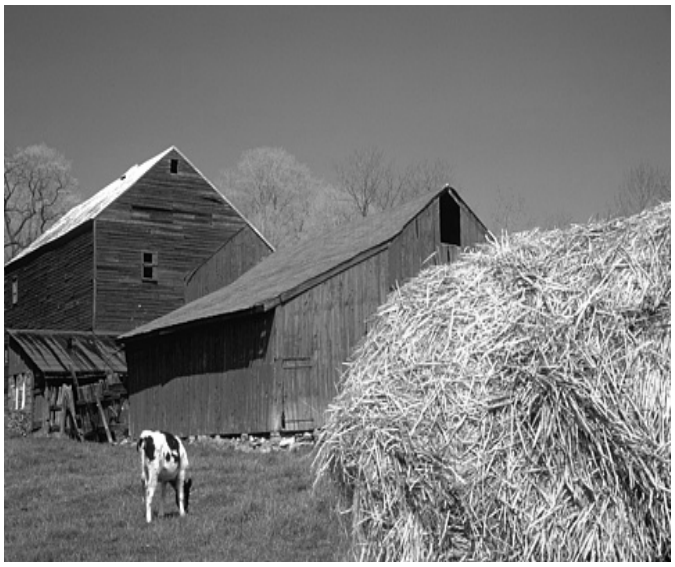
Figure 2.1b Grey scale

- We need to perform normalization on this gray scale image so as to get all the intensity values to lie in the range of 0 to 255.
- Sobel edge detection technique applies two different filters to an image to calculate Gx and Gy i.e. the gradient of that image along X and Y direction respectively.
- The following are the two filters to calculate Gx and Gy of the image A:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

- In our gray scale image, we take the local 3 × 3 window around each pixel and then convolve this window with each of the filters Gx and Gy separately.
- This convolution will result in the X gradient and Y gradient values respectively, for that single pixel.
- Now, using these Gx and Gy values, we can get the magnitude as well as orientation of the gradient at that pixel by using the following formula:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$
$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

- In this manner, we need to calculate G and Θ for each pixel of the gray scale image and store them in two different matrices.
- The matrix of all the G values of the image is the Gradient Magnitude Map and the matrix with all the Θ values is the Direction Map.
- Once we have the gradient magnitude map, we need to find the edge and non-edge pixels.
- Setting a threshold and then defining pixel values above that threshold as edge pixels can do this.
- We will consider two different thresholds, one with 10 % and the other with 15 % pixels to be edge points.
- To do this, we can simply calculate the histogram of the gradient magnitude map and then separate the 90 % pixels (non-edge) from the 10 % pixels (edge).

- For example, if the intensity values from 0 to 70 make up 90 % of the pixels in an image, then 70 will be the threshold value and all the intensity values that are more than 70 will contribute to the remaining 10 % of the image pixels.
- This means that the intensity (or gradient) values above 70 are the edge pixels and those below 70 are the non-edge pixels.
- On similar lines, we can perform thresholding for 15 % pixels to be edge pixels.
- In our problem, we have represented edge points by black color (intensity value of 0) and non-edge points by white color (intensity value of 255).
- This is how we have generated two different edge maps of the same image.
- Once we have obtained the edge maps, we can perform non-maximal suppression.
- In non-maximal suppression, the image is scanned along the gradient direction and if the pixels are not a part of the local maxima, then they are considered to be non-edge pixels.
- Now, to do this, we will need the direction map that we calculated earlier.
- To keep it simple, we have allowed the orientation values to be either 0 or 90.
- So, if the absolute value of the original direction lied somewhere in between 0 to 45 or 135 to 180, then they were changed to 0 otherwise (absolute value from 45 to 135) they were changed to 90.
- Now, we will again consider a 3 × 3 local window around each pixel (suppose K) in the gradient magnitude map.
- We consider the pixel (suppose K') in the direction map that corresponds to the pixel K in the gradient map and then we check if the orientation value of K' is 0 or 90.
- If it is 90, then we will go along Y direction of K and if it is 0 then we will go along X direction of K.
- Now, for the pixel K, depending upon the direction, we will check if K is the local maxima in its local window along that same direction.
- If it is the local maxima, then it is still an edge pixel, otherwise it becomes a non-edge pixel i.e. its value is set to 255 instead of 0.

- The following figure will make the idea more clear:

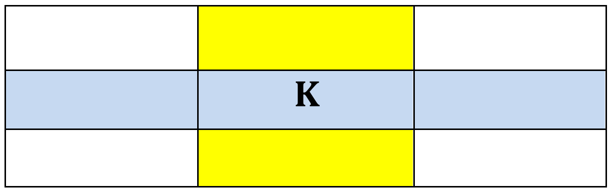

Figure No. 2.2 A local window with X (blue) and Y (yellow) direction around each pixel in the gradient magnitude map

- So if the value of K' is 90, then check if K is the local maximal amongst the yellow colored pixels and if K' is 0, then check if K is the local maximal amongst the blue colored pixels.
- Thus, we allow only those pixels that are local maximal in their local window, to be edge pixels.
- After performing non-maximum suppression, we get thin edges of the image that we define to be the enhanced image maps.

2.1.3 Results

The above approach has been implemented using C++. The gradient Magnitude Map of the Farm and Cougar image is as follows:

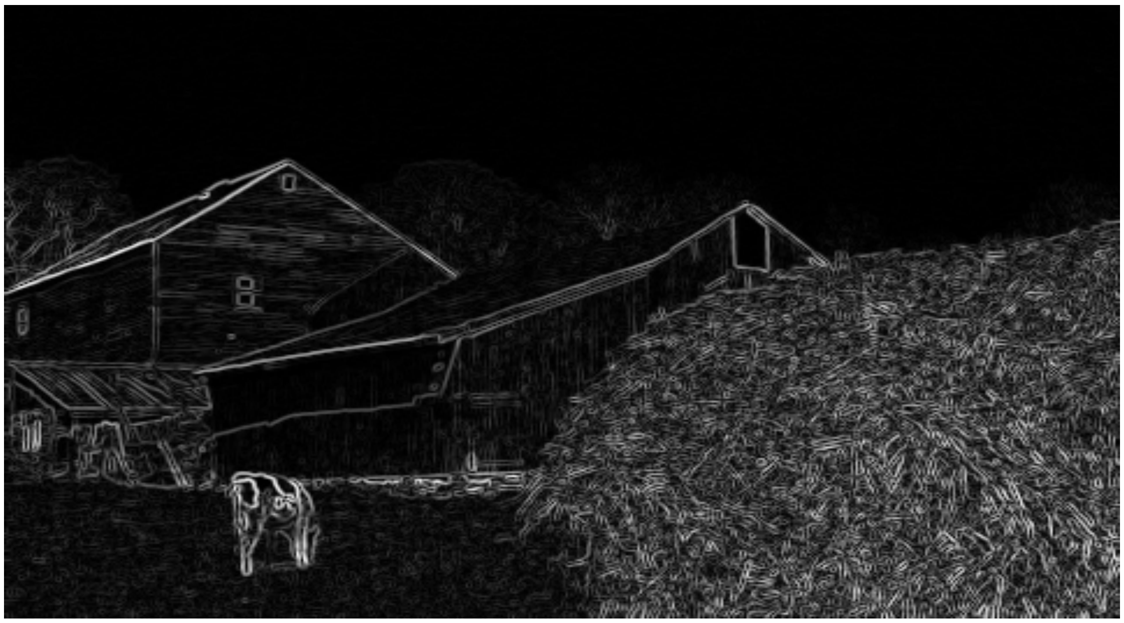

Figure 2.2a Gradient Magnitude Map of Farm

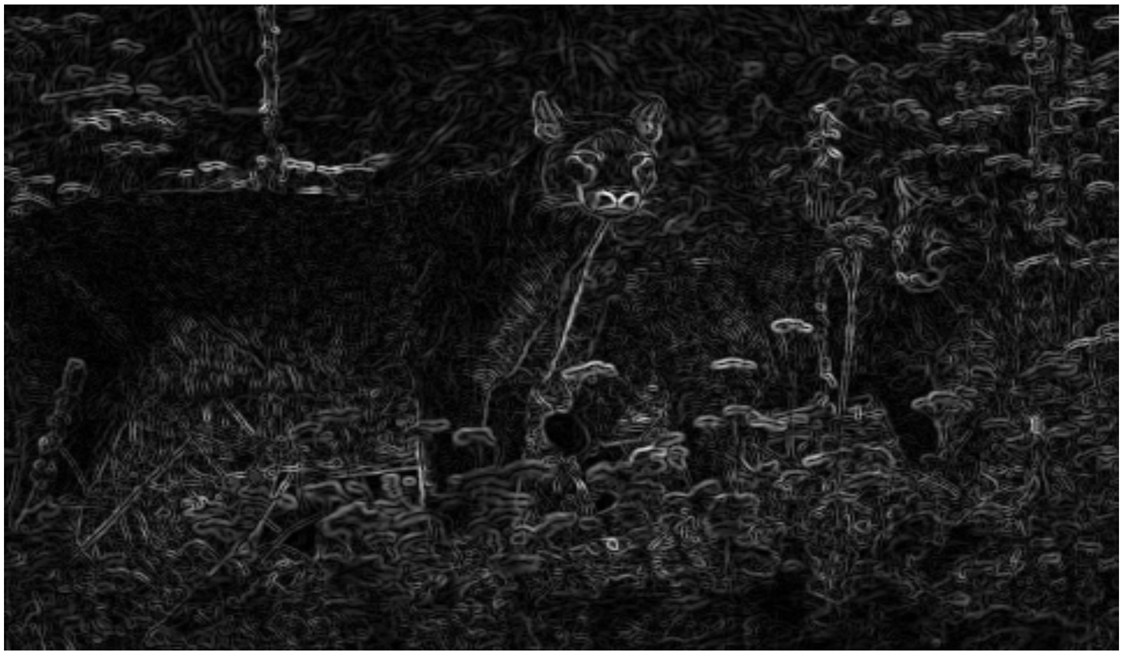

Figure 2.3b Gradient Magnitude Map of Cougar

The edge map for 10 % and 15 % of pixels as edge pixels for Farm image is as follows:

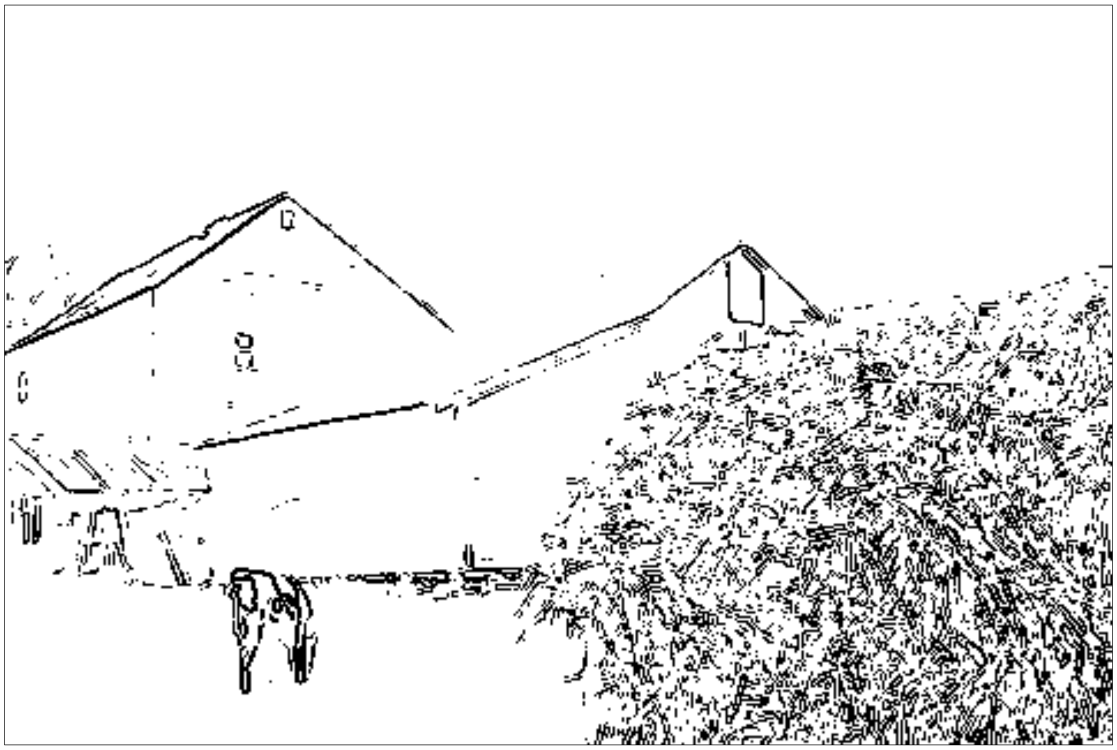
Figure 2.4a Edge Map of Farm with 10% pixels as edge pixels

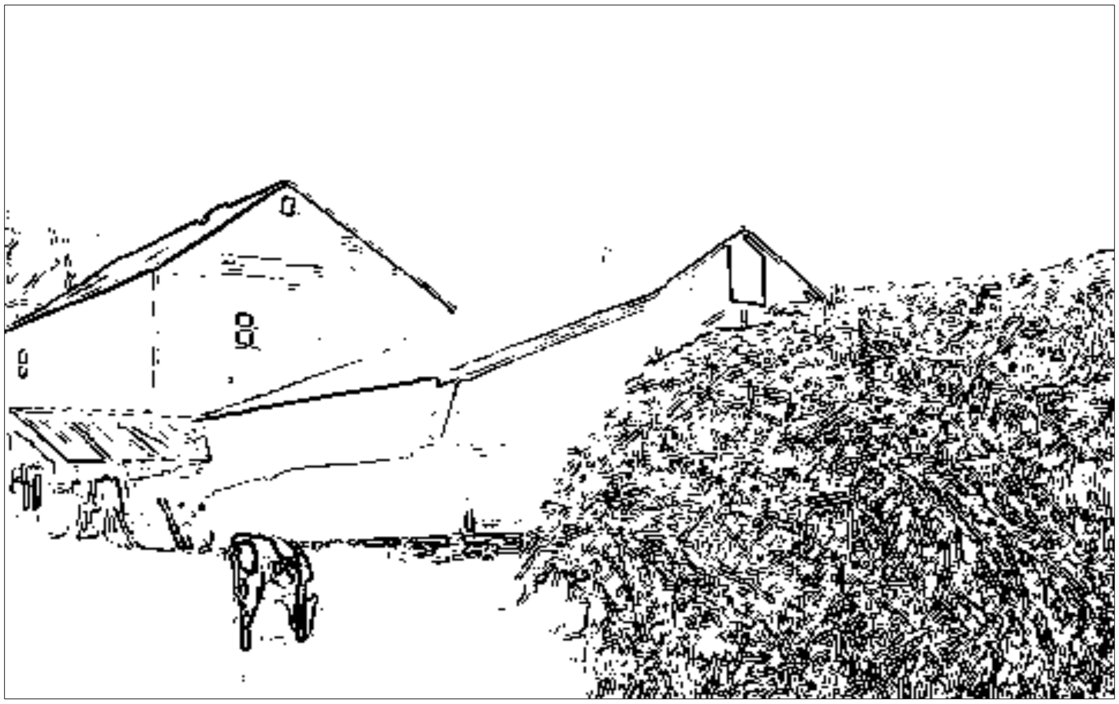
Figure 2.4b Edge Map of Farm with 15% pixels as edge pixels

The edge map for 10 % and 15 % of pixels as edge pixels for Cougar image is as follows:

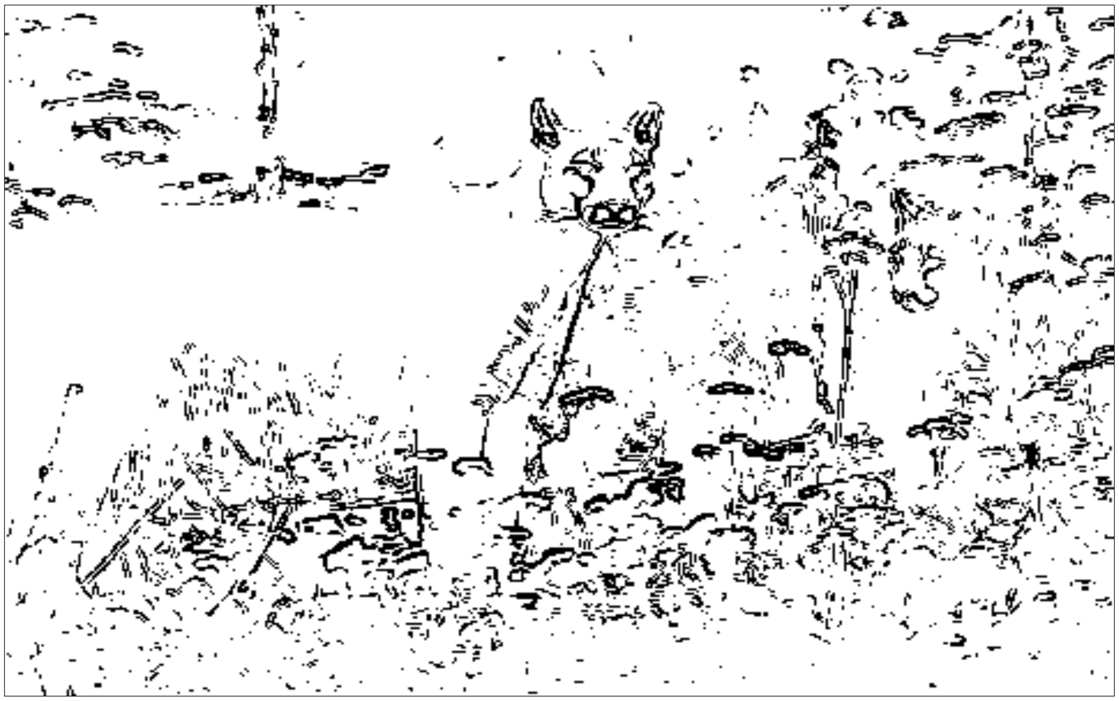
Figure 2.5a Edge Map of Cougar with 10% pixels as edge pixels

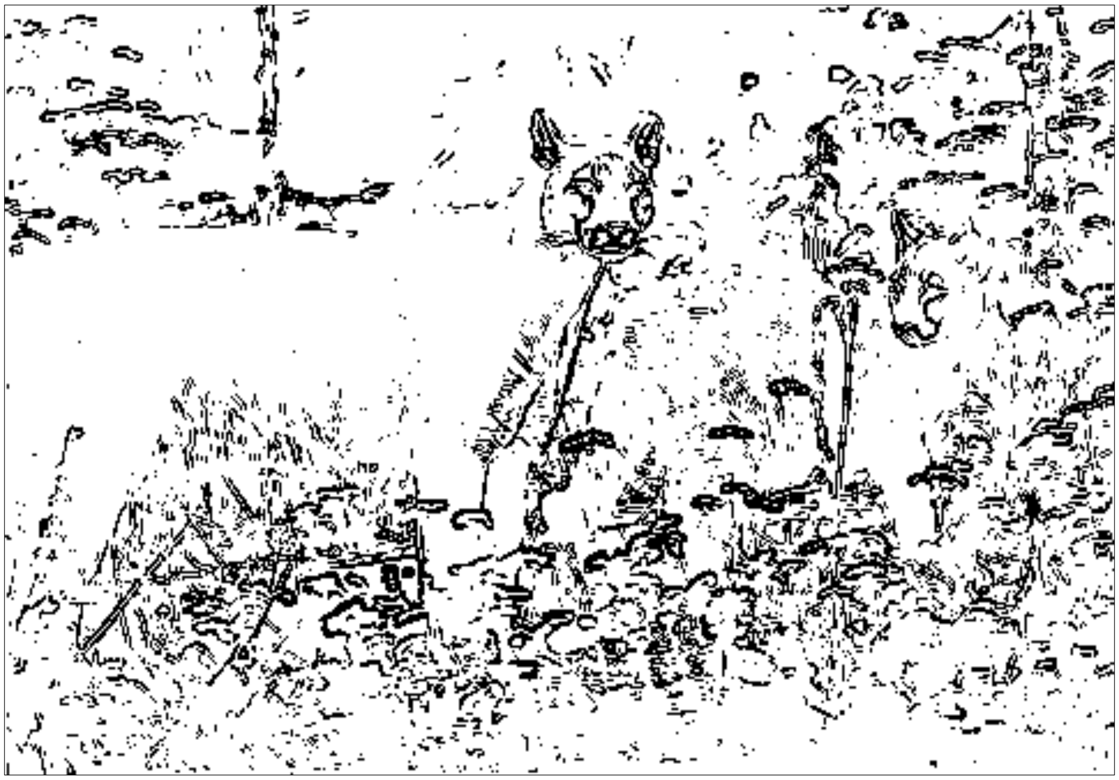
Figure 2.5b Edge Map of Cougar with 15% pixels as edge pixels

The enhanced edge maps obtained after non-maximum suppression are as follows:

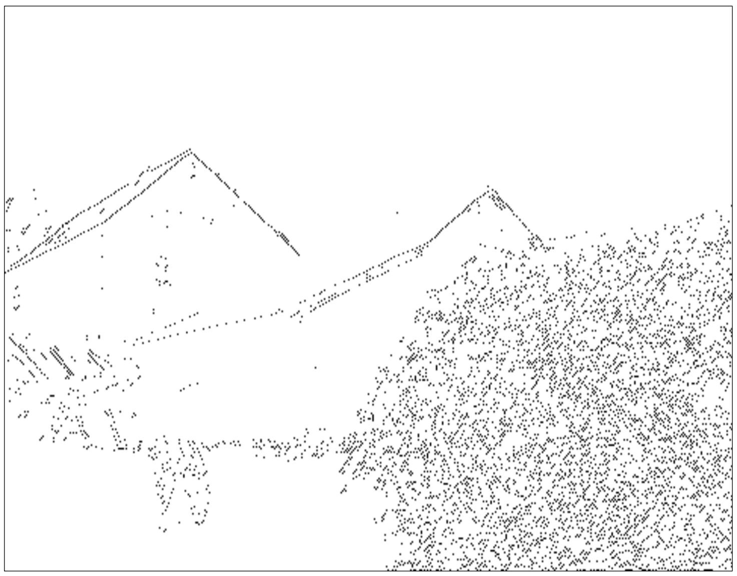
Figure 2.6a Enhanced Edge Map of Farm with 10% pixels as edge pixels

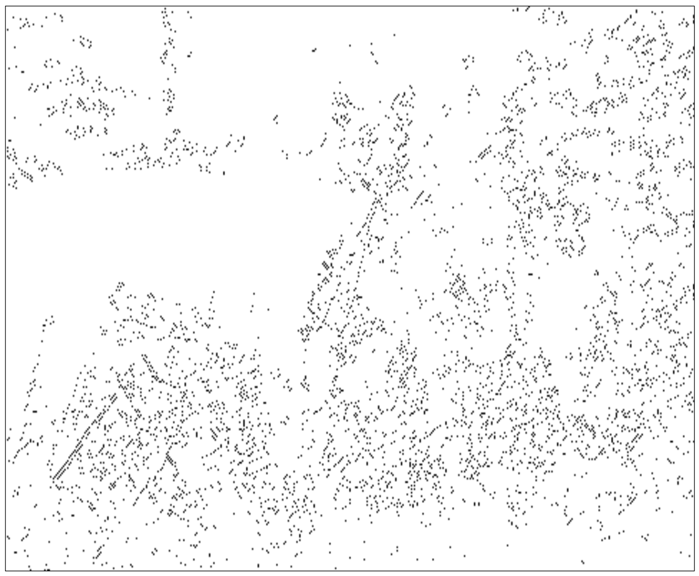
Figure 2.7a Enhanced Edge Map of Cougar with 10% pixels as edge pixels

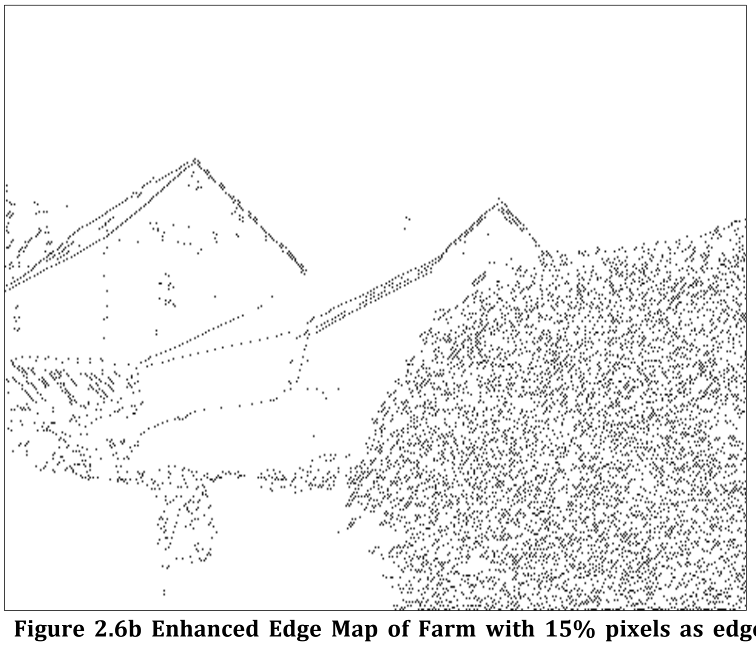
Figure 2.6b Enhanced Edge Map of Farm with 15% pixels as edge pixels

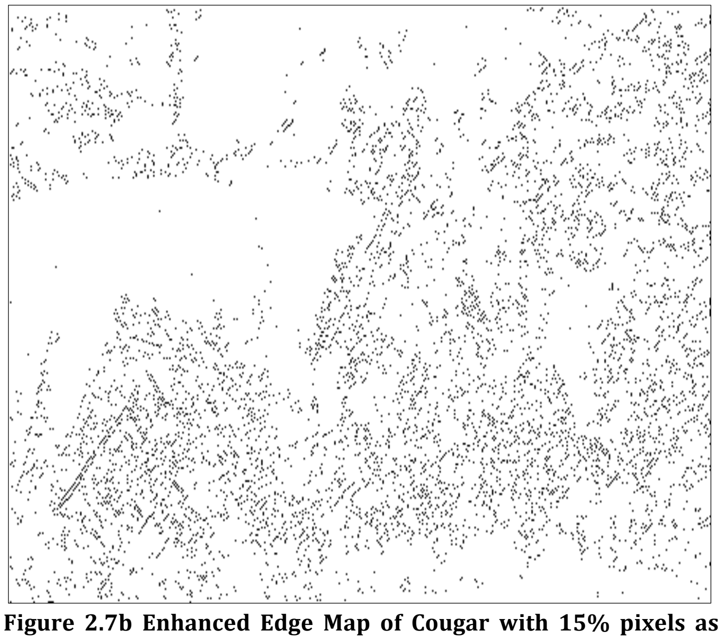
Figure 2.7b Enhanced Edge Map of Cougar with 15% pixels as edge pixels

2.1.4 Discussion

We have learnt how Sobel Edge detection operates to extract the edges from an image. Different types of images of farm and Cougar can be observed from the above results. Moreover, we can also conclude from the edge maps that the farm image has sharper edges than the cougar image. Sobel detection does not remove the speckle noise that might be present in the image. It directly applies the sobel filters to extract edges. This drawback can be improved by using a different edge detection method such as Canny edge detection that has been explored in the following problem.

2.2 Canny Edge Detector

2.2.1 Motivation

Canny Edge Detection uses an algorithm with multiple stages. Before starting the process of edge detection, it first applies a Gaussian filter to remove noise from the image. Canny edge detector provides good and reliable results and it is an optimal detection technique.

2.2.2 Approach and Implementation

- Canny edge detection technique can be implemented by using the following steps:

Step 1: Remove noise in the image by applying a Gaussian filter
Step 2: Compute the gradient magnitude map of the image
Step 3: Perform non-maximum suppression to remove spurious responses
Step 4: Use double thresholding to detect edges
Step 5: Decide the final edges by using hysteresis

- A Gaussian filter is convolved with the image to make the image smooth and to remove noise. Using an appropriate Gaussian kernel will do this.
- To calculate the intensity gradient, we perform the same procedure that has been discussed in section 2.1.2 given above.
- Non-maximum suppression is done to make the edges thin.
- Suppressing the gradient values that are not local maximal to zero does this.
- After this step, there are still some edge pixels that are caused by color variation or noise and to get rid of these, we perform thresholding.
- Canny edge detector uses two threshold values, one is the high threshold value (suppose H) and the other is the low threshold value (suppose L).
- If the pixel value is higher than H, they are considered to be strong edge pixels.
- If the pixel value is higher than L and lower than H, then they are considered to be weak edge pixels.

- If the pixel value is lower than L, then they are considered to be non-edge pixels i.e. they are suppressed.
- So now we have strong edge pixels and weak edge pixels in our edge map.
- Strong edge pixels are definitely needed but we do not yet know which weak edge pixels are to be preserved and which are to be suppressed.
- So, we now track the edges to check if the weak pixels are from true edges or from noise.
- Considering 8 neighborhood pixels that are connected to each other does this.
- We now have the true edges of the image as we have preserved only the strong edge points and suppressed the weak points that were isolated.
- For the last step of hysteresis, we can choose different values for high (H) and low (L) thresholds.
- We have considered 5 different sets of values in this problem and every set produces different edge maps that are shown in the following results.
- The above approach has been implemented using OpenCV in C++.

2.2.3 Results

The edge maps for Farm image with 5 different sets of L and H values are as shown below:

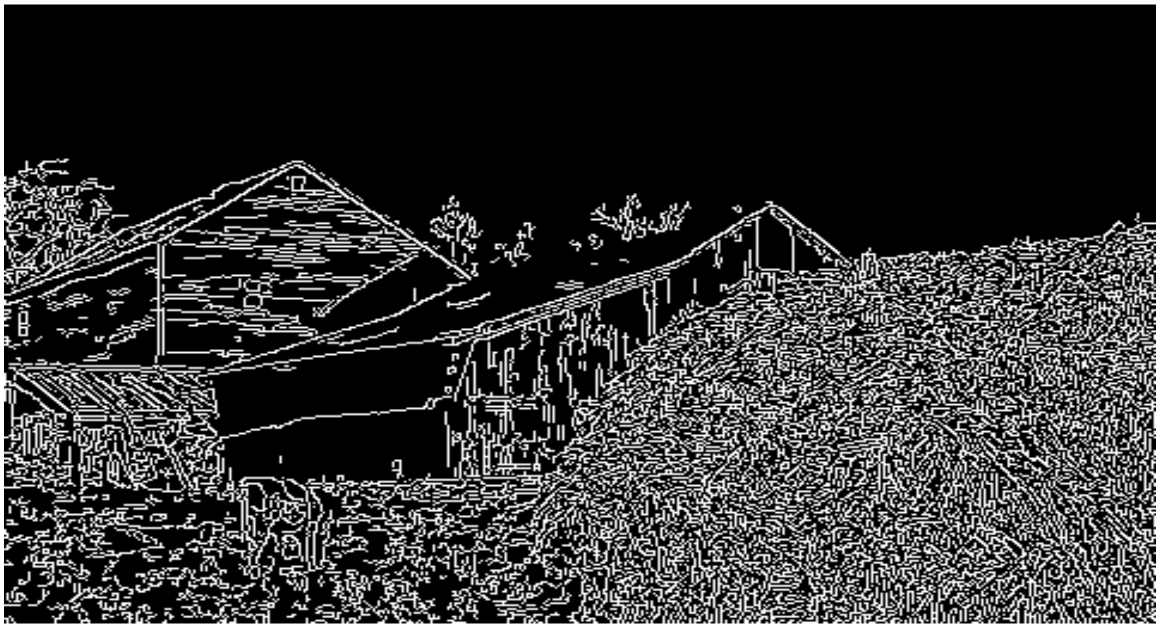
Figure 2.8a Canny edge map for Farm with L = 0.3, H = 0.6

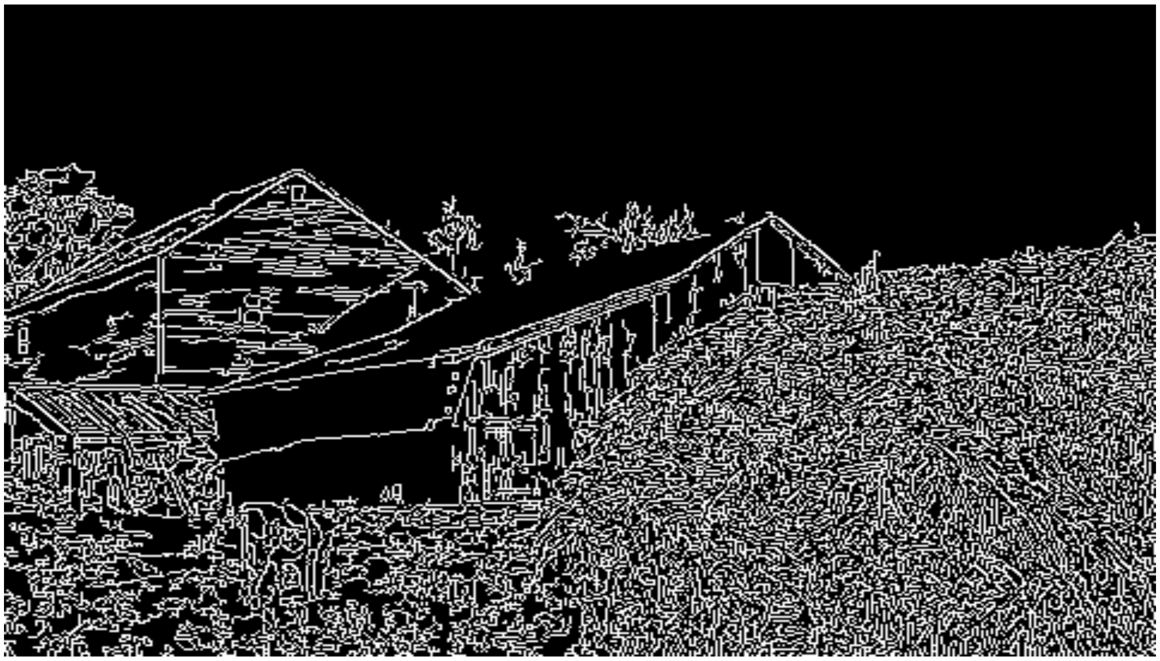
Figure 2.8b Canny edge map for Farm with L = 0.2, H = 0.7

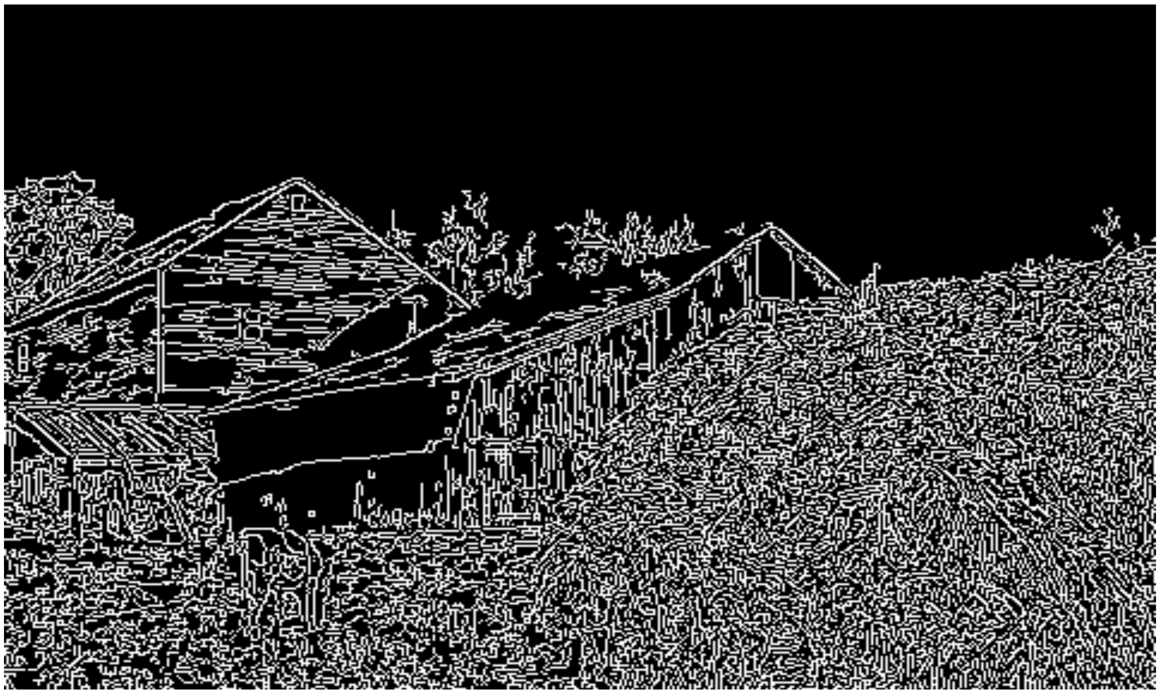
Figure 2.8c Canny edge map for Farm with L = 0.2, H = 0.5

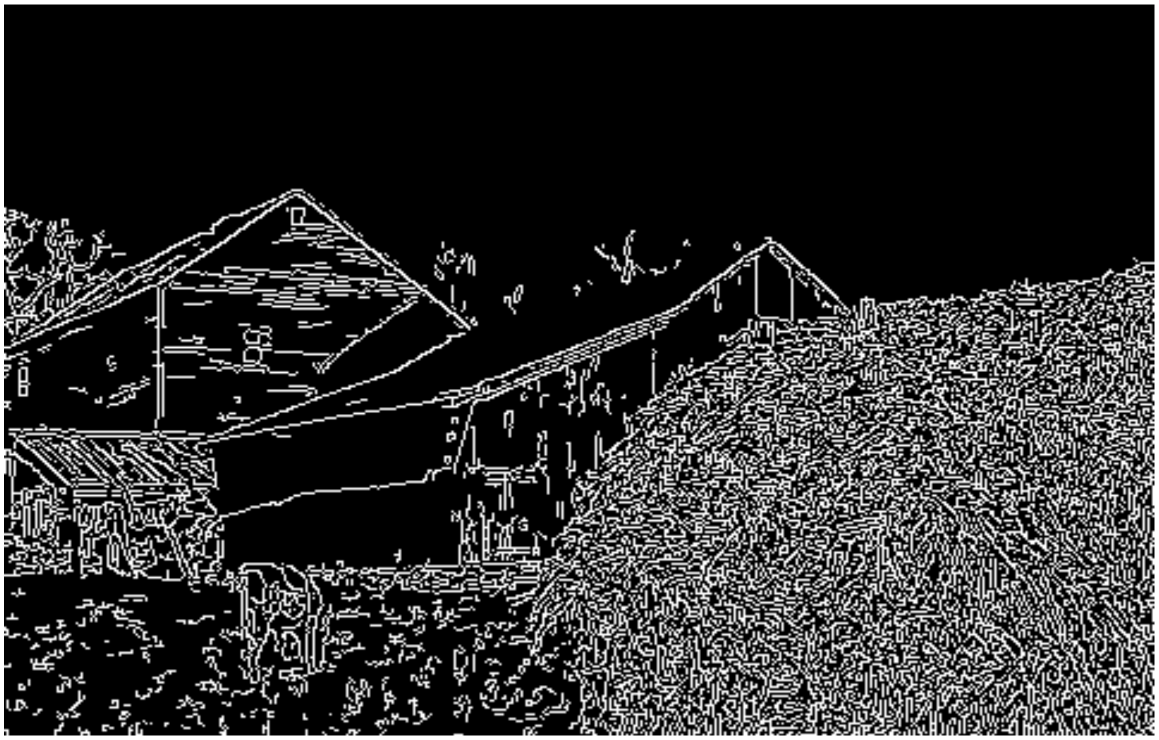
Figure 2.8d Canny edge map for Farm with L = 0.4, H = 0.7

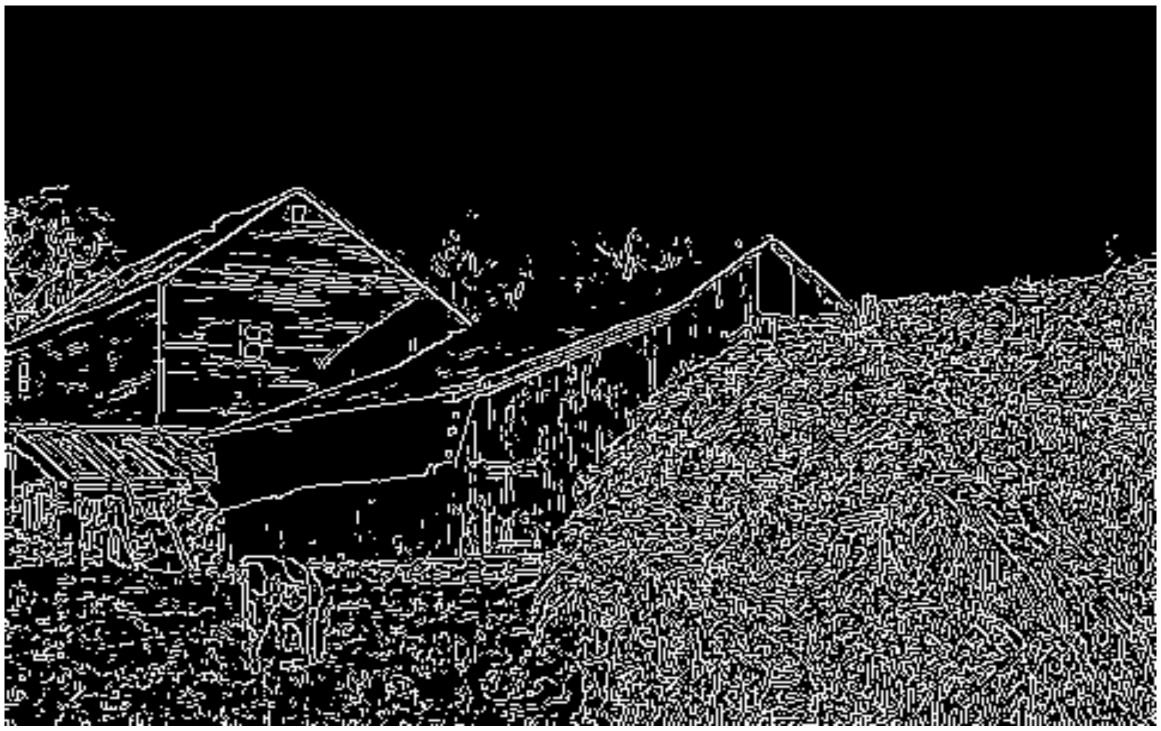

Figure No. 2.8(e) Canny edge map for Farm with L = 0.4, H = 0.5

The edge maps for Cougar image with 5 different sets of L and H values are as shown below:

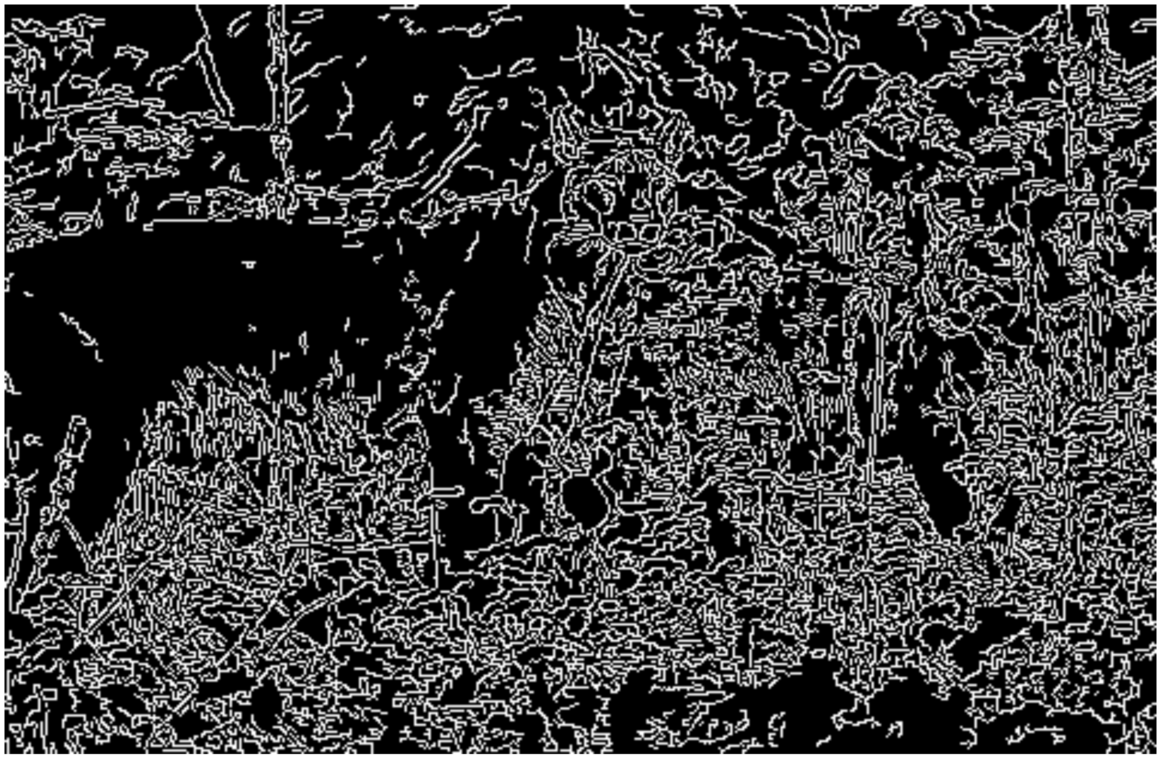

Figure No. 2.9(a) Canny edge map for Cougar with L = 0.3, H = 0.6

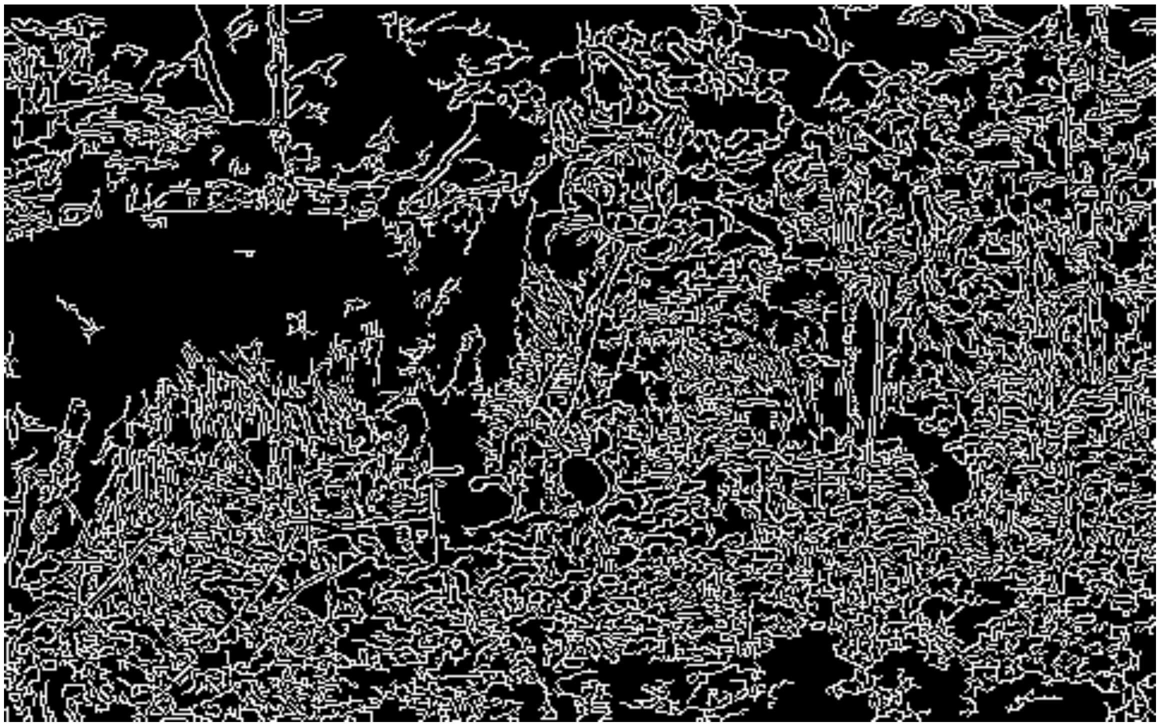

Figure No. 2.9(b) Canny edge map for Cougar with L = 0.2, H = 0.7

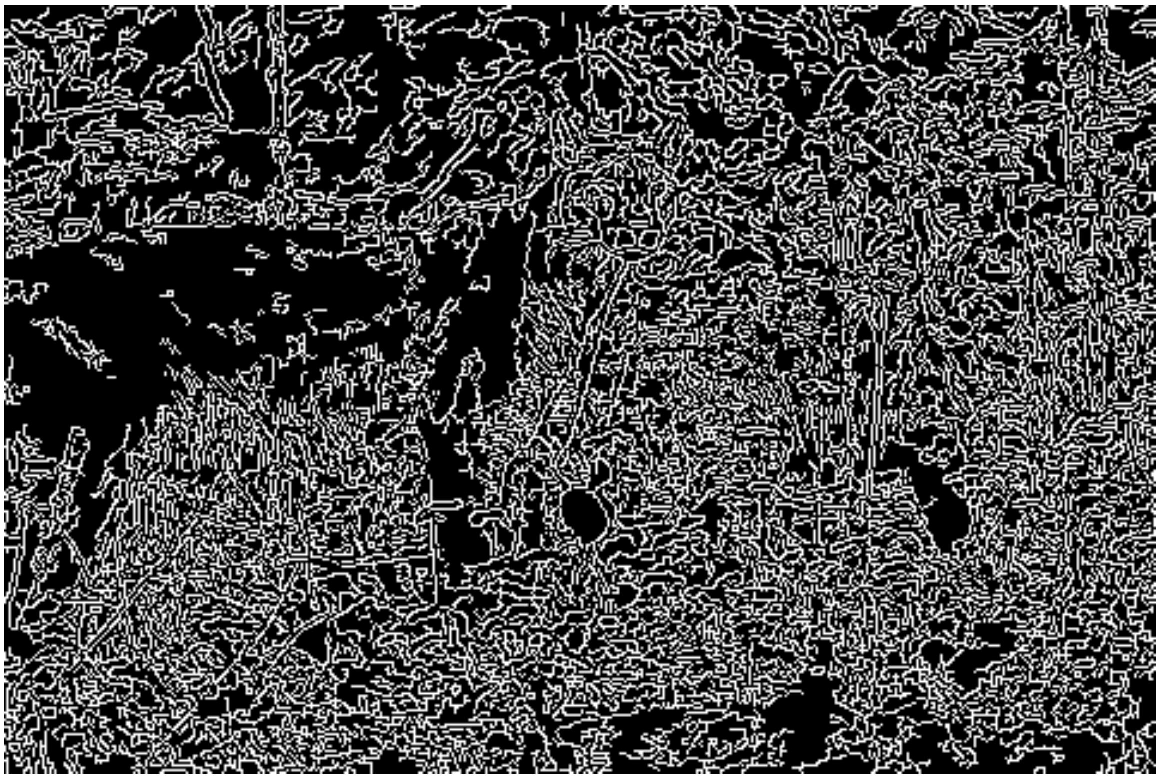

Figure No. 2.9(c) Canny edge map for Cougar with L = 0.2, H = 0.5

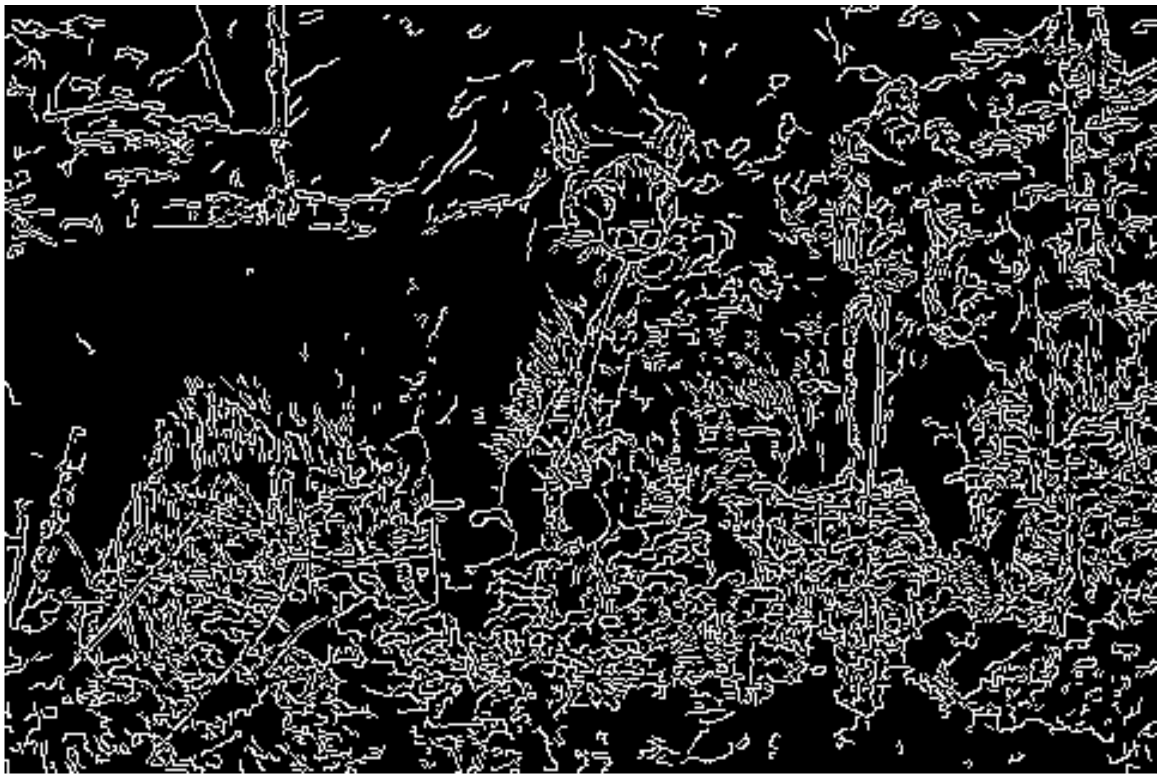

Figure No. 2.9(d) Canny edge map for Cougar with L = 0.4, H = 0.7

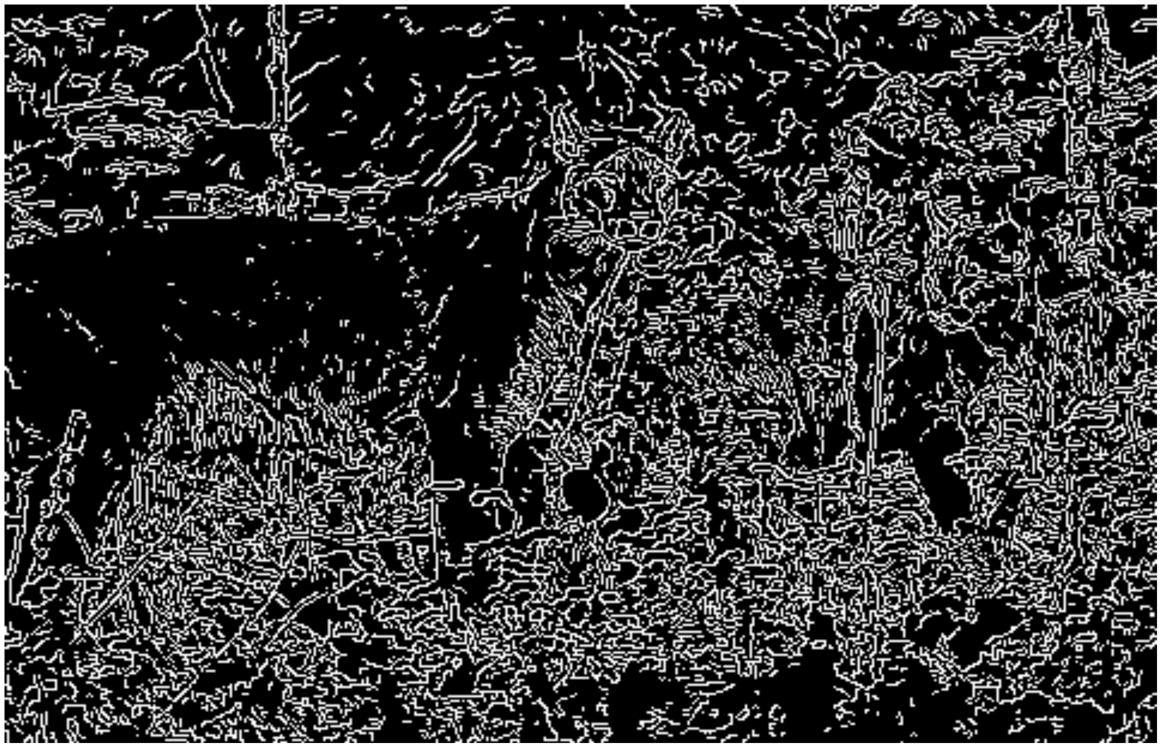

Figure No. 2.9(e) Canny edge map for Cougar with L = 0.4, H = 0.5

2.2.4 Discussion

From the above results, we can see how the edge content varies as we change the lower and upper threshold values during hysteresis. Here, when L = 0.2, it corresponds to the threshold of intensity value of 255×0.2 = 51 which means that the actual value of L is 51. We are multiplying by 255 because that is the maximum intensity value of the image. We can conclude that the edge map is the best when L = 0.4 and H = 0.7 (Figure No. 2.8(d) and 2.9(d)). The high threshold value, H, should be high enough so as to allow only those pixels that are in fact edge pixels, to be strong pixels. At the same time, the low threshold value should be such that it suppresses only the pixels that are isolated and preserves the pixels that are connected to the strong pixel's edges. When this combination is ideally met, we get a reliable edge map that does not contain any non-edge pixels. If we pick values like L = 0.2 and H = 0.5 (Figure No. 2.8(c) and 2.9(c)), we get quite bad results because the edges are not distinctly visible and the image is not smooth. Also, we can see some weak object boundary regions in the image. Thus choosing the appropriate threshold values is an important step in Canny Edge Detection.

2.3 Structured Edge

2.3.1 Motivation

As we discussed earlier, edge detection is an important step in object recognition and segmentation. We have looked at Sobel and canny edge detection that use intensity gradient approach for detecting edges. The drawback of these methods is that they fail to detect edges that are not due to gradients, but are visible in an image, e.g. texture edges. To overcome this, we can use a new state-of-the-art method of Fast Edge Detection using Structured Forests. In this approach, we consider the inherent structures, which are present in the edge patches, with the help of random forests and then compute the edge map by predicting pixel labels using gain measures on structured labels.

2.3.2 Approach and Implementation

2.3.2.1 Decision Trees and Random Forest Classifier

- Structured Edge detector uses a random forest framework to extract the structure information.
- A random forest is an ensemble of decision trees and a decision tree is used to classify a sample by creating branches using recursion.
- To create branches at its nodes, a decision tree uses its own split function defined as $h(x,\theta j) \in \{0,1\}$, where x is the sample and θj is a parameter of node j.
- Depending upon the value of 'h', the sample x will be sent to the right branch or the left branch of the node.
- Usually, the split function compares the feature value of a sample with some threshold by using an indicator function to output 0 or 1.
- At each node, a feature is selected in such a way that it classifies the samples with the best purity and this purity is measured by computing the entropy or information gain or gini index.
- Thus, features are randomly selected by comparing their entropy values.

- Now, to increase the accuracy of classification, we use the concept of ‘Bagging’, i.e. we will consider multiple decision trees that classify the samples.
- The training data samples are first divided into random subsets and then each of these independent subsets is used to train a different decision tree.
- Thus, for every decision tree we use a random subset of the training data as well as a random subset of the features.
- In this way, we construct multiple uncorrelated decision trees by using the training data.
- These trees together form the ‘Random’ Forest.
- Now, for a test sample, every tree will give a separate class label to that sample depending upon the features that it uses for classification.
- The class label that is generated by the maximum number of trees in random forest is selected and then assigned to the test sample.
- The following figure illustrates the above concept:

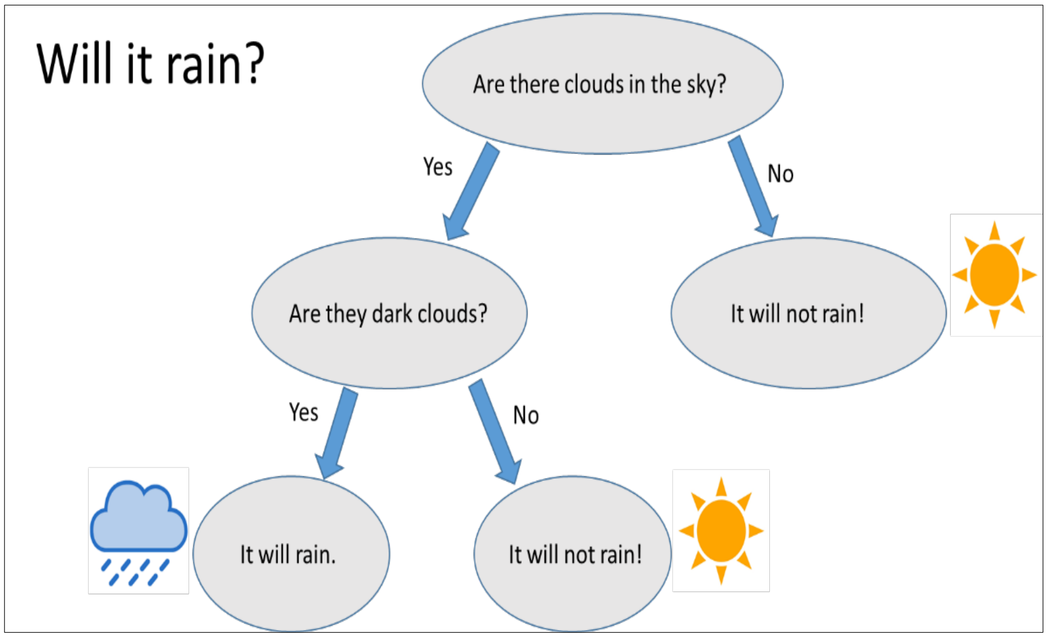

Figure No. 2.10(a) A decision tree to evaluate if it will rain or not [2]

- Now, to construct this decision tree, suppose we consider different cities as our samples and consider the two interior nodes as our features.

- This decision might not work for a different city perhaps because the probability that it will rain might not depend on the features that we have used in this decision tree, e.g. it might depend on the temperature in that city or the city's location.'
- Thus, we need to use different uncorrelated decision trees with different training samples and features to predict if it will rain in a particular city.
- This will then give rise to a random forest as shown below:

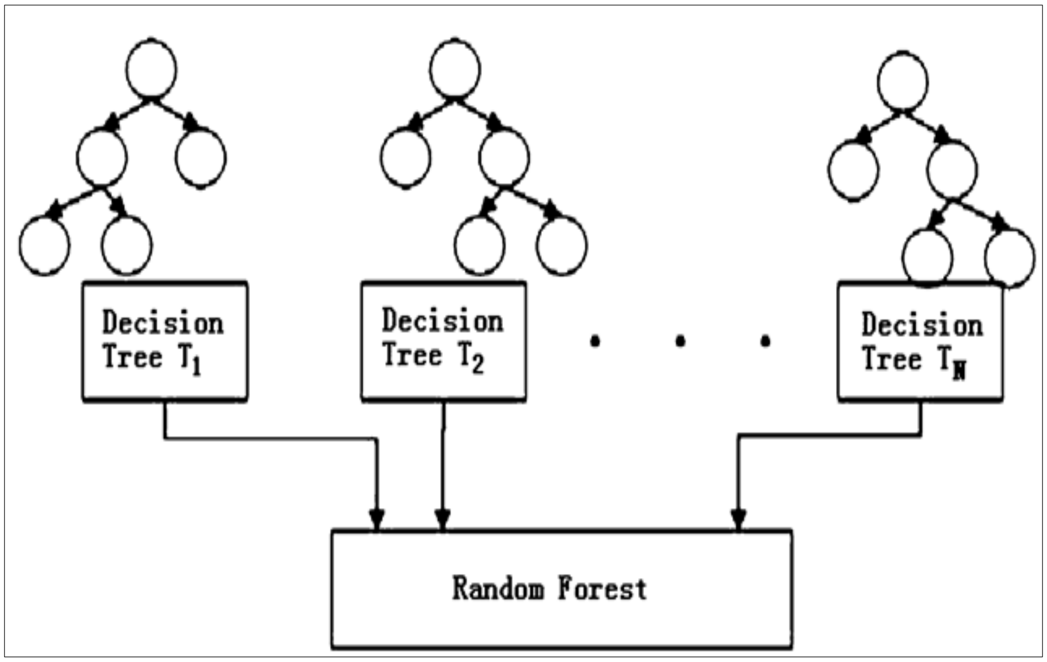

Figure No. 2.10(b) Random Forrest consisting of decision trees [3]

- Thus, for a given sample (city), every decision tree will make different predictions, but to select one true prediction, we will choose the prediction that is made by the maximum number of decision trees in the random forest.
- We can extend this concept of a random forest to a structured random forest with structured labels and a structured output space.
- We can find this difficult because the output space is now complex and the information gain over structured labels cannot be neatly defined.
- To simplify this, we perform mapping from structured labels to a discrete set of labels so that information gain can be easily calculated.

2.3.2.2 Structured Edge Detection Algorithm

The input to the SE detector is an RGB image and the output is a matrix of labels with binary values indicating an edge or a non-edge pixel, for all the pixels. The flowchart of the algorithm is as shown below:

Aim : To predict 16×16 segmentation mask from a 32×32 image patch.

- Feature processing:
  - Use 2 types of features: pixel lookups & pairwise differences
  - 3 color channels, 2 magnitude channels, 8 orientation channels = total 13 channels
  - Down-sampling by 2 in both orientations gives (32×32×13)/4 = 3328 channels of type pixel lookup
  - Additional 300 features per channel = 300 × 13 = 3900 of type pairwise difference
  - Total 7228 features for each image patch of 32×32

- Mapping Function:
  - We need 16×16 = 256 structured labels.
  - Need to convert segmentation masks by using mapping to create an edge map
  - For every pixel j (j is from 1 to 256), check if y(j1)=y(j2) to define a binary vector that encodes the indicator function on the equality
  - Might result into high dimensions, so limit the dimensions to 256

- Ensemble Model:
  - We are using structured random forest and so we will average all the different edge maps to yield the true edge response of that image patch.
  - Each pixel has (16×16)/4 predictions for one decision tree, thus total 64 T predictions for T trees.
  - We train 2T trees (due to un-correlation) and evaluate T trees at each adjacent location.

2.3.3 Results

The above approach has been implemented in MATLAB using the online source in [4].

The images of Farm after performing structured edge detection on it are as follows:

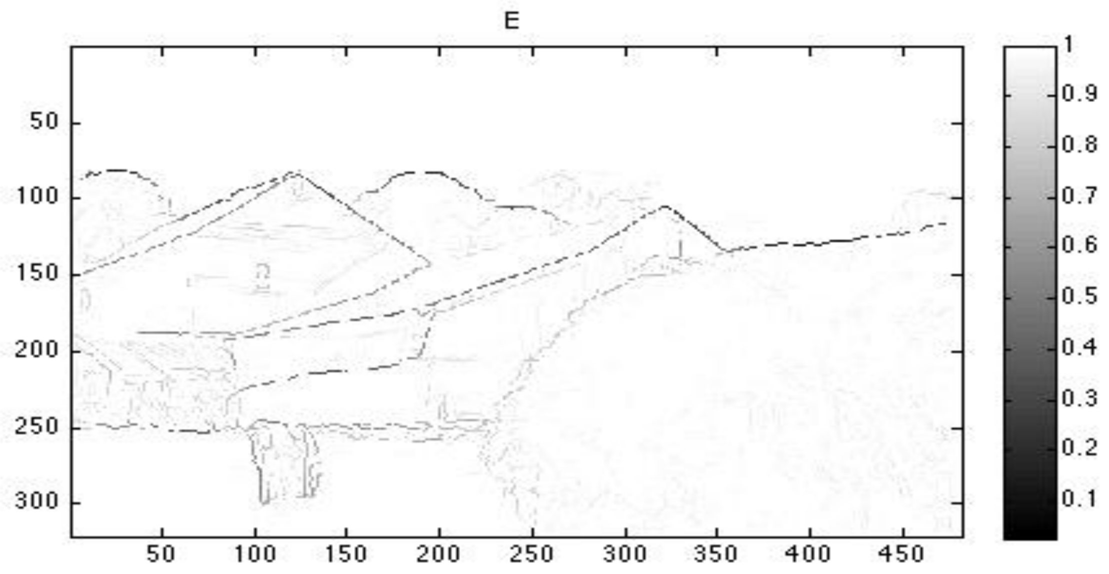

Figure No. 2.11(a) Probability edge map of Farm using SE

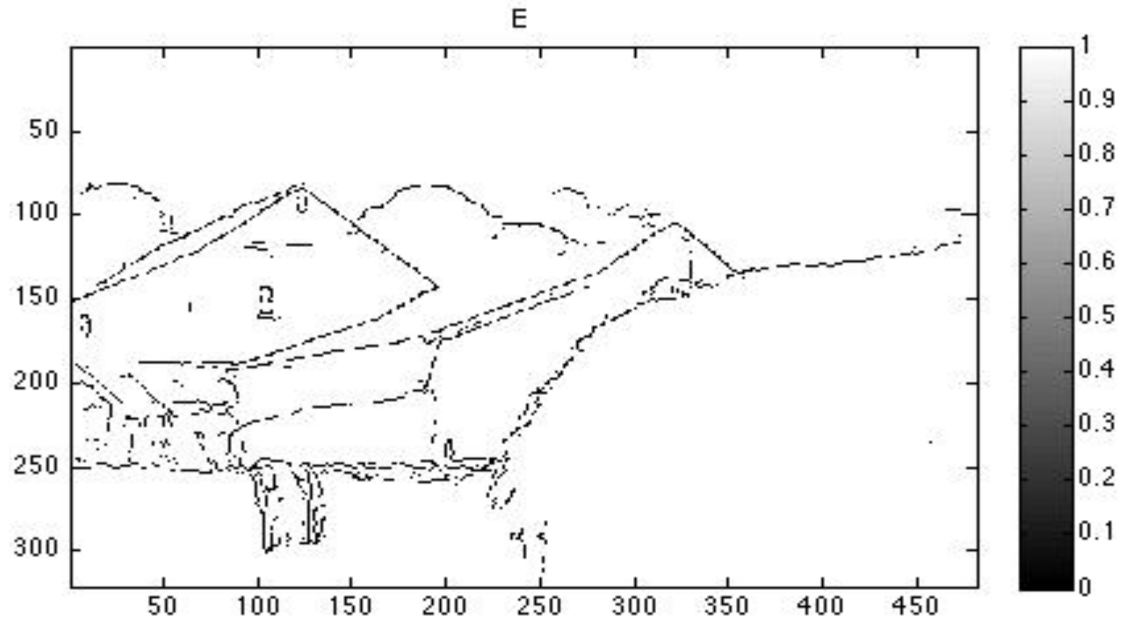

Figure No. 2.11(b) Binary edge map of Farm using SE

The images of Cougar after performing structured edge detection on it are as follows:

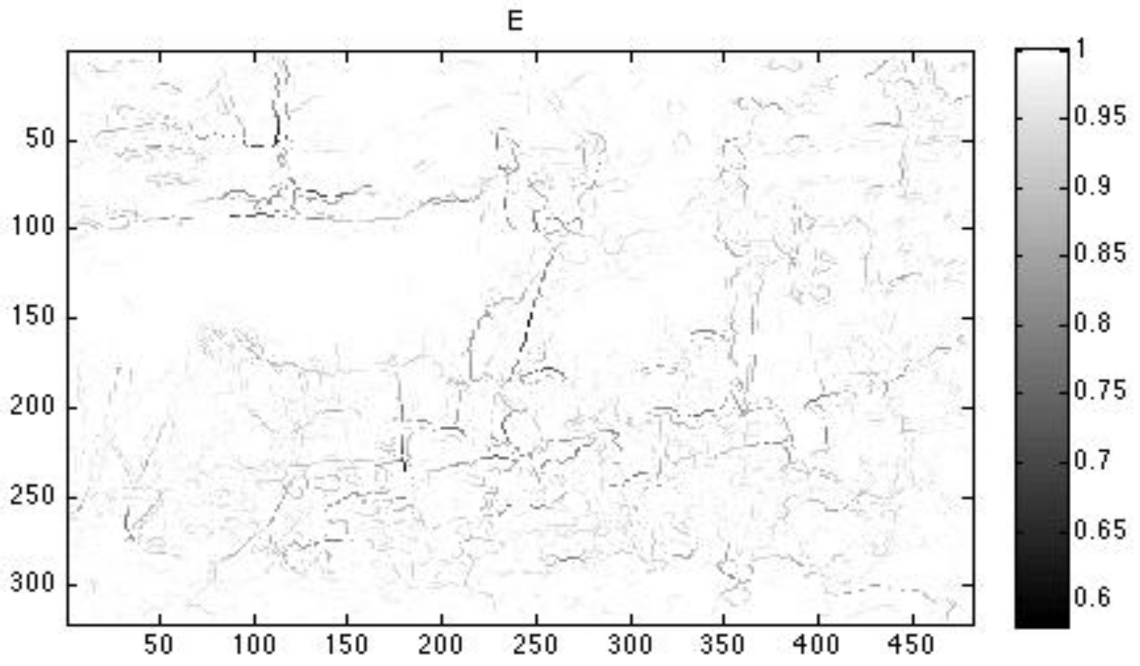

Figure No. 2.12(a) Probability edge map of Cougar using SE

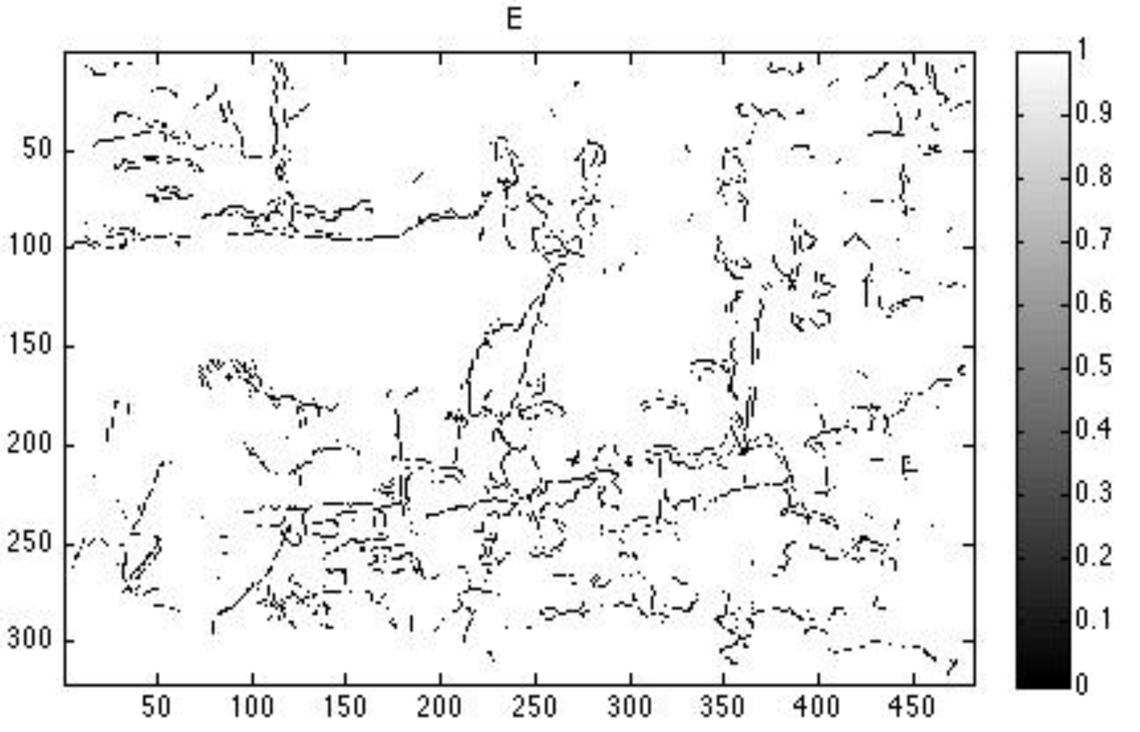

Figure No. 2.12(b) Binary edge map of Cougar using SE

2.3.4 Discussion

We can choose different parameters like sharpening value, number of trees to be evaluated, a flag to perform non-maximum suppression etc. Moreover, to calculate the binary edge map, we can set the threshold value of probability as per our requirements. The parameters that have been used for the above images are as follows:

Parameter	Farm	Cougar
Multi-scale flag	1	1
Sharpness value	10	20
No. of trees to evaluate	4	4
Number of threads	6	4
NMS flag	1	1
Probability threshold	0.125	0.06

Table No.2.1 Parameters used for SE detection on Farm and Cougar images

The value of multi-scale flag has been set to 1 for better accuracy. We can use the number of trees from 1 to 4 and to get the best performance, this number has been set to 4. The probability values have been chosen by trial and error, by observing the binary edge map that each value generates. We can see how Structured edge detector shows only the true edge pixels in the image and suppresses the other non-edge pixels.
Canny edge detector performs moderately but it contains a lot of non-edge pixels in the edge map and also it is not smooth. Sobel edge detector gives the least accurate results amongst the 3 techniques that we have considered. It does not accurately determine all the potential edge pixels. Thus, we can conclude that structured edge detector gives the most reliable results of the binary edge map for Farm and Cougar images.

2.4 Performance Evaluation

2.4.1 Motivation

In the discussion section of the above problem, we have visually compared the edge maps that were obtained using three different techniques namely Sobel, Canny and Structured Edge Detection. However, we have not considered that different people will have different opinions about the edges. Thus, to evaluate the performance on a certain technique, we need a diversity of results. For this, we can use different ground truth edge maps to evaluate 'Precision' and 'Recall' of a certain edge map and then by taking the mean of these values, we can use the 'F measure' to represent the performance of each edge detection technique. In this problem, we have calculated the mean values of precision and recall for different edge maps and their corresponding ground truths. This is followed by the computation of F measure and depending on this value, we will determine the level of performance of that specific edge detection technique.

2.4.2 Approach and Implementation

- Basically, not all the pixels, which are assigned to edge pixels, are actually edge pixels.
- Similarly, the pixels that were detected to be non-edge pixels might actually be edge pixels.
- In short, we need to explore how accurately our edge detector has classified edge and non-edge pixels (Suppose the detected edge map is E).
- So, we will compare our results with a set of ground truths that have different pre-defined edge maps (Suppose the ground truth is G).
- To do this, we make use of four terms defined as follows:
    - True Positive: The edge pixel of E coincides with the edge pixel of G
    - True Negative: Non-edge pixel of E coincides with non-edge pixel of G
    - False Positive: The edge pixel of E coincides with the non-edge pixel of G
    - False Negative: Non-edge pixel of E coincides with edge pixel of G.

- We will calculate the total number of false positive & negative and true positive & negative values in our edge map by comparing it with different ground truths.
- After we get all the four values, we define three more terms as shown below:

  - Precision = $\frac{\text{No. of True positive}}{\text{No. of true positive + No. of false positive}}$

  - Recall = $\frac{\text{No. of True positive}}{\text{No. of true positive + No. of false negative}}$

  - F = $2\times\left(\frac{P\times R}{P+R}\right)$

- We can say that precision is a measure of exactness and recall is a measure of completeness.
- Thus, a higher precision means that we get more relevant results than irrelevant whereas a higher recall means that we get most of the relevant results.
- So, we want both our precision and recall to be as high as possible.
- Finally, we can say that our edge detector performs well if it has a high value of F.
- F measure is simply the harmonic mean of precision and recall.
- When we are measuring the performance of our system, it is often useful to have a single number that describes our performance and this number if nothing but the F measure.
- In conclusion, F measure conveys the balance between precision and recall.
- If we somehow manage to get a higher precision and a lower recall or vice versa, then it is not possible to get a high F measure, mainly because the numerator term contains the product of the two values and it will result in a lower number if either of the values is low.
- That is the main idea of why we use F measure! It is high only when both precision and recall are high.
- If the sum of precision and recall is constant then the F measure will be the maximum only when Precision and recall have the same values.
- This can be proved as follows:

Let ‘p’ and ‘r’ be any two numbers such that their sum is the same number ‘s’.
We can have 3 cases: $p > r$, $r > p$, $r = p$.
Let us use numbers for p, r and s for the above 3 cases:
$s = p + r = 6$.
Case 1: $p = 4$ and $r = 2$ ($p > r$)
Case 2: $p = 1$ and $r = 5$ ($r > p$)
Case 3: $p = 3$ and $r = 3$ ($p = r$)
Now, F measure is proportional to the ratio of the product ($p \times r$) and the sum ($p + r$)
The values of F measure for the above 3 cases are: 2.67, 1.67 and 3 respectively.
Thus, we can see that F measure has the highest value when Precision and recall are the same.

- We have calculated the Precision and recall values for all the edge maps generated until now, by using the provided ground truths.
- This approach has been implemented in MATLAB by using the ‘edgesEvalImg’ function in [5].

2.4.3 Results

The precision and recall values along with their mean values and the corresponding F measure for edge maps of Sobel Edge detection are as shown below:

Image type	***Recall & GT No.***	***Precision & GT No.***	***F Measure***
NMS Farm for 10%	0.2762 With GT1	0.0726 With GT1	
	0.2938 With GT2	0.128 With GT2	
	0.2903 With GT3	0.1345 With GT3	
	0.3522 With GT4	0.125 With GT4	
	0.3011 With GT5	0.1257 With GT5	
MEAN	0.30272	0.1171	0.1688
NMS Farm for 15 %	0.3522 With GT1	0.0748 With GT1	
	0.3855 With GT2	0.1357 With GT2	
	0.3769 With GT3	0.1410 With GT3	
	0.4550 With GT4	0.1307 With GT4	
	0.3940 With GT5	0.1328 With GT5	
MEAN	0.39272	0.123	0.1873
NMS Cougar for 10%	0.4008 With GT1	0.3671 With GT1	
	0.4346 With GT2	0.3646 With GT2	
	0.4244 With GT3	0.1426 With GT3	
	0.4342 With GT4	0.3975 With GT4	
	0.4437 With GT5	0.2593 With GT5	
MEAN	0.4275	0.306	0.3566
NMS Cougar for 15%	0.5036 With GT1	0.3339 With GT1	
	0.5546 With GT2	0.3369 With GT2	
	0.5379 With GT3	0.1308 With GT3	
	0.5300 With GT4	0.3513 With GT4	
	0.5394 With GT5	0.2283 With GT5	
MEAN	0.5331	0.2764	0.3640

Table No. 2.2(a) Precision, recall and F measure for Sobel detector on Farm and Cougar Images

The precision and recall values along with their mean values and the corresponding F measure for edge maps of Canny Edge detection are as shown below:

Image type	***Precision & GT No.***	***Recall & GT No.***	***F Measure***
Farm with A=0.3,B=0.6	0.7546 With GT1	0.04547 With GT1	
	0.8418 With GT2	0.0840 With GT2	
	0.8519 With GT3	0.09041 With GT3	
	0.9823 With GT4	0.08009 With GT4	
	0.8778 With GT5	0.08393 With GT5	
MEAN	0.86168	0.07678	0.1409
Farm with A=0.2,B=0.7	0.7933 With GT1	0.0455 With GT1	
	0.8643 With GT2	0.0822 With GT2	
	0.8716 With GT3	0.0881 With GT3	
	0.9904 With GT4	0.0769 With GT4	
	0.9001 With GT5	0.0820 With GT5	
MEAN	0.88394	0.0749	0.138
Farm with A=0.2,B=0.5	0.8106 With GT1	0.0431 With GT1	
	0.8729 With GT2	0.0769 With GT2	
	0.8775 With GT3	0.0822 With GT3	
	0.9933 With GT4	0.0715 With GT4	
	0.9103 With GT5	0.0768 With GT5	
MEAN	0.89292	0.0701	0.1299
Farm with A=0.4,B=0.7	0.7179 With GT1	0.04922 With GT1	
	0.8200 With GT2	0.0931 With GT2	
	0.8318 With GT3	0.1004 With GT3	
	0.9709 With GT4	0.0900 With GT4	
	0.8534 With GT5	0.0928 With GT5	
MEAN	0.8388	0.0851	0.1545
Farm with A=0.4,B=0.5	0.7495 With GT1	0.0471 With GT1	
	0.8375 With GT2	0.0873 With GT2	
	0.8471 With GT3	0.0938 With GT3	
	0.9790 With GT4	0.0833 With GT4	
	0.8745 With GT5	0.0873 With GT5	
MEAN	0.8575	0.07976	0.1459
Cougar with A=0.3,B=0.6	0.9925 With GT1	0.1544 With GT1	
	0.9952 With GT2	0.1419 With GT2	
	0.9928 With GT3	0.0566 With GT3	
	0.9927 With GT4	0.1544 With GT4	
	0.9907 With GT5	0.0984 With GT5	
MEAN	0.99278	0.1211	0.2158
Cougar with A=0.2,B=0.7	0.9934 With GT1	0.1382 With GT1	
	0.9959 With GT2	0.1269 With GT2	
	0.9928 With GT3	0.0506 With GT3	
	0.9894 With GT4	0.1376 With GT4	
	0.9924 With GT5	0.0881 With GT5	
MEAN	0.99278	0.1442	0.2518

Cougar with A=0.2,B=0.5	0.9986 With GT1	0.1240 With GT1	
	0.9990 With GT2	0.1137 With GT2	
	0.9958 With GT3	0.0453 With GT3	
	0.9973 With GT4	0.1238 With GT4	
	0.9951 With GT5	0.0789 With GT5	
MEAN	0.99716	0.09714	0.1770
Cougar with A=0.4,B=0.7	0.9638 With GT1	0.1903 With GT1	
	0.9803 With GT2	0.1773 With GT2	
	0.9605 With GT3	0.0695 With GT3	
	0.9636 With GT4	0.1902 With GT4	
	0.9636 With GT5	0.1214 With GT5	
MEAN	0.96636	0.14974	0.2593
Cougar with A=0.4,B=0.5	0.9905 With GT1	0.1684 With GT1	
	0.9973 With GT2	0.1554 With GT2	
	0.9922 With GT3	0.0619 With GT3	
	0.9892 With GT4	0.1682 With GT4	
	0.9876 With GT5	0.1072 With GT5	
MEAN	0.99136	0.1322	0.2332

Table No. 2.2(b) Precision, recall and F measure for Canny detector on Farm and Cougar Images

The precision and recall values along with their mean values and the corresponding F measure for edge maps of Structured Edge detection are as shown below:

Image type	***Precision & GT No.***	***Recall & GT No.***	***F Measure***
Farm	0.7900 With GT1	0.3710 With GT1	
	0.8200 With GT2	0.6380 With GT2	
	0.8241 With GT3	0.6815 With GT3	
	0.8876 With GT4	0.5639 With GT4	
	0.8445 With GT5	0.6292 With GT5	
MEAN	0.83324	0.5767	**0.6816**
Cougar	0.6525 With GT1	0.4834 With GT1	
	0.6464 With GT2	0.4388 With GT2	
	0.6770 With GT3	0.1840 With GT3	
	0.6660 With GT4	0.4933 With GT4	
	0.7127 With GT5	0.3370 With GT5	
MEAN	0.67092	0.3873	**0.4911**

Table No. 2.2(c) Precision, recall and F measure for SE detector on Farm and Cougar Images

2.4.4 Discussion

From the above results we can see that the F measure value for Cougar as well as Farm image is maximum when we use Structured Edge Detection (shown in red in table No. 2.2(c)). We can see that the values of F measure obtained by using Canny and Sobel are quite low. Moreover, Sobel performs better than Canny on the Cougar and Farm image. By changing the threshold values of Canny, we might get a different result. Now, F measure depends upon the image content. We can see that it is easier to get a high F measure on the Farm image because it has sharper edges. The visualization of F measure for Cougar image with SE detection and sobel detection is as shown below:

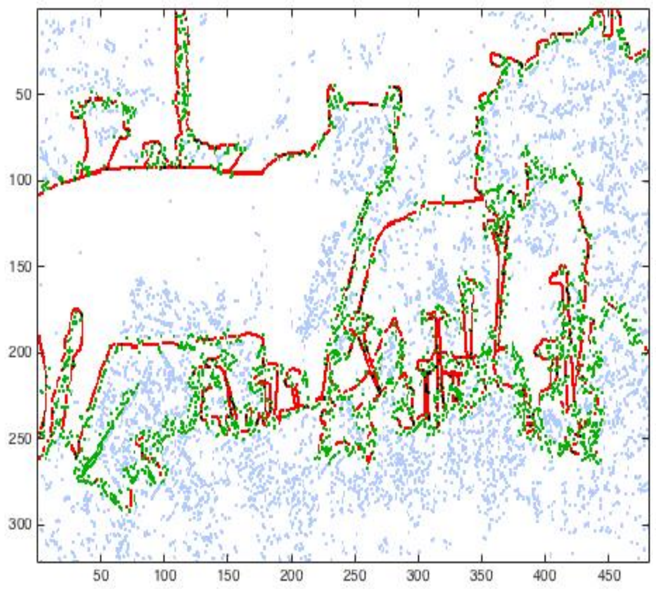

Figure No. 2.13(a) Cougar using Sobel

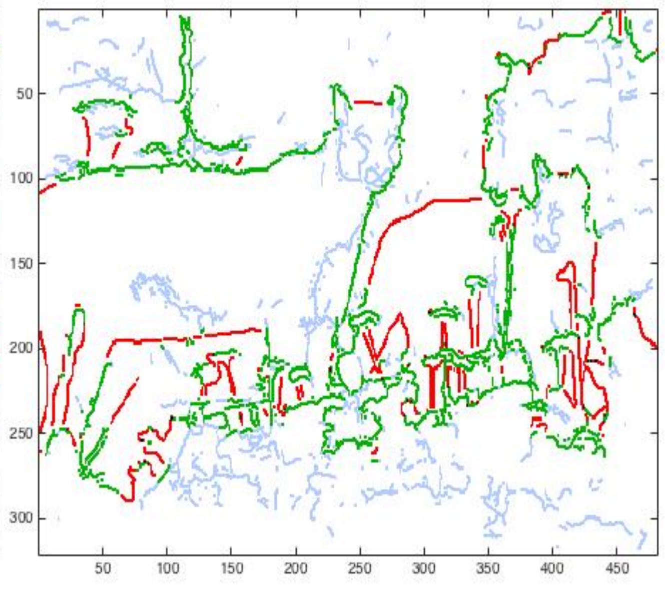

Figure No. 2.13(b) Cougar using SE

From the above images, we can see that the red colored edge is more in the sobel detected image whereas the green colored edge is more in the SE detected image. This is how we can observe the performance of an edge detector by visualizing its F measure

3. References

[1] Internet
[2] Decision tress samples - Internet
[3] Random Forest example - Internet
[4] https://github.com/pdollar/edges
[5] http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html