Midterm lab Notebook- Kaggle

The code for the midterm is in JAVA and some JAVA library available for feature extraction. At the end of all the experiments. I could get around 50-51% accuracy. I was working other effective methods but need more time. Found some difficulties in finding the libraries for most of the methods.

The Following are the final features which I could extract

Length, Width, Length/Width ratio, Area, Number of Edges, Number of Blobs, Number of cavities

Experiment: 1

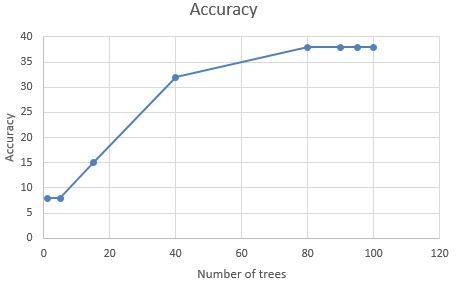
Goal: To determine the width to length ratio of all the images to classify the classes of the plankton.

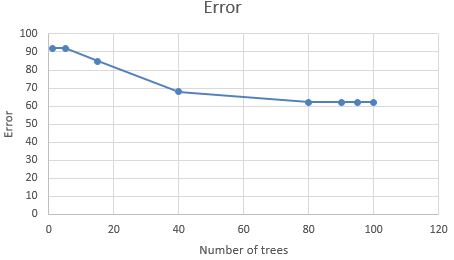
Procedure: This involves some pre-processing like Thresholding the images, segmenting the images and extracting the region properties. In order to reduce noise, the threshold on the mean value. They are then dilated to connect neighbouring pixels. Then rescale the image to be a constant size and adding fixed number of pixels to the feature list. An inbuilt library for Java provides the height and width of each of the images. Then the ratio is calculated.

Analysis: The ratio of width to length couldn’t help better to classify because the plankton from different classes might have the same range of width to length ratio. The accuracy for this only feature was about 38 percent.

Error rate:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of trees | Accuracy |  | Number of trees | Error |
| 1 | 8 |  | 1 | 92 |
| 5 | 8 |  | 5 | 92 |
| 15 | 15 |  | 15 | 85 |
| 40 | 32 |  | 40 | 68 |
| 80 | 38 |  | 80 | 62 |
| 90 | 38 |  | 90 | 62 |
| 95 | 38 |  | 95 | 62 |
| 100 | 38 |  | 100 | 62 |





Experiment: 2

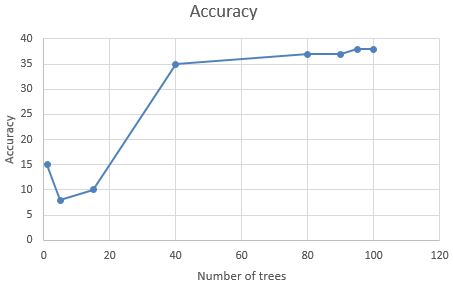
Goal: To Find the Area as feature using the Length and Width of the image.

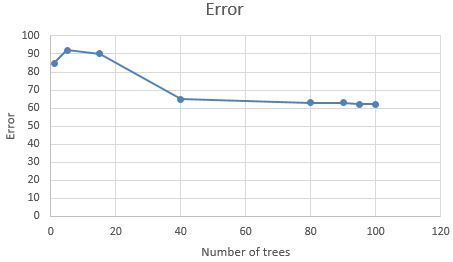
Procedure: In the previous experiment, the length and width of the image is found using the inbuilt Java function. Adding to this, the area is also calculated.

Analysis: There was not much change in the accuracy level. This might be because plankton under the same classes may vary much based on the Area.

Error rate:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of trees | Accuracy |  | Number of trees | Error |
| 1 | 15 |  | 1 | 85 |
| 5 | 8 |  | 5 | 92 |
| 15 | 10 |  | 15 | 90 |
| 40 | 35 |  | 40 | 65 |
| 80 | 37 |  | 80 | 63 |
| 90 | 37 |  | 90 | 63 |
| 95 | 38 |  | 95 | 62 |
| 100 | 38 |  | 100 | 62 |





Experiment: 3

Goal: To determine the number of edges of the plankton.

Procedure: The method that was used initially was the **Canny Edge Detection method**.

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Edge\_Gradient(G)=(G2x+G2y)^1/2

Angle(θ)=tan−1(GyGx)

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighbourhood in the direction of gradient. The result you get is a binary image with thin edges.

 setSourceImage (image) : which sets the input image.  
    - process ( ) : which is the main method and computes the edge map of the given image.  
    - getEdgesImage ( ) : which returns the computed edge map.  
  
    Please note that I have used a method called "rgb2bin" which gets the output of previous step, and maps the black and white pixels to 0 and 1, respectively. This is done to make further extractions easier.

Analysis: I was trying to count the number of edges from the measured gradients. I tried to have a histogram based on edge directions and thereby I can count the number of directional and non-directional edges. I couldn’t move further because

Canny provides an edge map based on the gradient magnitude because it only returns the threshold gradient magnitude information it cannot provide you with the orientation information.

Error rate: Nil

Experiment: 4

Goal: To count number of sharp corners of the grayscale image. This would help to have some clarity to find the structure of the image to be predicted. I started with the **Hough Transform**.

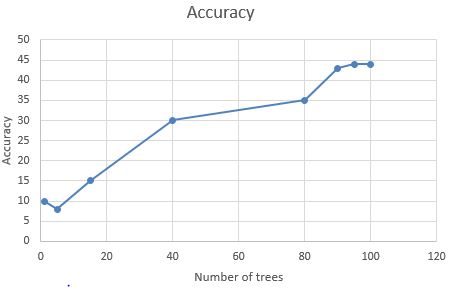
Procedure: The transform maps each point in the target image, (ρ,θ), to the average color of the pixels on the corresponding line of the source image (in (*x*,*y*)-space, where the line corresponds to points of the form *x*cosθ + *y*sinθ = ρ). The idea is that where there is a straight line in the original image, it corresponds to a bright (or dark, depending on the color of the background field) spot; by applying a suitable filter to the results of the transform, it is possible to extract the locations of the lines in the original image.

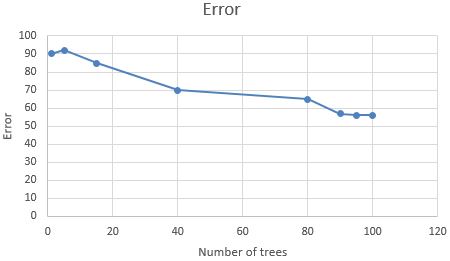
Analysis: I could extract the number of sharp corners from the image. But they were not effective for classification purpose. The idea of this transform is that, where there is a straight line in the original image, it corresponds to a bright (or dark, depending on the color of the background field) spot. In the images we have, most don’t have straight lines. They are mostly curves. The accuracy rate was very low.

Accuracy: The accuracy rate was around 43-44%

Error rate:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of trees | Accuracy |  | Number of trees | Error |
| 1 | 10 |  | 1 | 90 |
| 5 | 8 |  | 5 | 92 |
| 15 | 15 |  | 15 | 85 |
| 40 | 30 |  | 40 | 70 |
| 80 | 35 |  | 80 | 65 |
| 90 | 43 |  | 90 | 57 |
| 95 | 44 |  | 95 | 56 |
| 100 | 44 |  | 100 | 56 |
|  |  |  |  |  |





Experiment: 5

Goal: To count number of edges of the grayscale image. As the accuracy level of Hough transform was very less, I proceeded with **Harris Detector Method**.

Procedure: Harris Detector method helps to find the intersection of two edges, it represents a point in which the directions of these two edges change. Hence, the gradient of the image (in both directions) have a high variation, which can be used to detect it. The function runs the Harris edge detector on the image. Similarly to [cornerMinEigenVal()](http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=cornerharris#void cornerMinEigenVal(InputArray src, OutputArray dst, int blockSize, int ksize, int borderType)) and [cornerEigenValsAndVecs()](http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=cornerharris#void cornerEigenValsAndVecs(InputArray src, OutputArray dst, int blockSize, int ksize, int borderType)) , for each pixel (x, y) it calculates a 2\times2 gradient covariance matrix M^{(x,y)} over a \texttt{blockSize} \times \texttt{blockSize} neighbourhood.

Analysis: I could extract the number of corners from each image using this method. The time taken to perform the extraction was larger than the Hough Transform. But the accuracy level was little higher than the Hough Transform. This might be even because of change in the number of training datasets.

Error rate:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of trees | Accuracy |  | Number of trees | Error |
| 1 | 10 |  | 1 | 90 |
| 5 | 8 |  | 5 | 92 |
| 15 | 15 |  | 15 | 85 |
| 40 | 30 |  | 40 | 70 |
| 80 | 38 |  | 80 | 62 |
| 90 | 50 |  | 90 | 50 |
| 95 | 51 |  | 95 | 49 |
| 100 | 51 |  | 100 | 49 |

Experiment: 6

Goal: To Find the number of Blobs in the image. Counting the number of blobs would help to match the plankton because most of the plankton has a blob shaped structure in the center part.

1. Procedure: Thresholding: Convert the source images to *several* binary images by thresholding the source image with thresholds starting at minThreshold. These thresholds are incremented by threshold Step until maxThreshold. So the first threshold is minThreshold, the second is minThreshold + threshold Step, the third is minThreshold + 2 x threshold Step, and so on.
2. Grouping: In each binary image, connected white pixels are grouped together.  Let’s call these binary blobs.
3. Merging: The center of the binary blobs in the binary images are computed, and blobs located closer than minDistBetweenBlobs are merged.
4. Center & Radius Calculation:  The center and radii of the new merged blobs are computed and returned.

Analysis:

Error rate:

Experiment: 7

Goal: To Detect Key points detectors in the image using **SURF** in JAVA.

Procedure: It is an algorithm which extracts some unique key points and descriptors from an image. A SURF key points that are extracted from an image is used to detect a same type of image. It uses an intermediate image representation called Integral Image which is computed from the input image and is used to speed up the calculation in any rectangular area. It is based on the determinant of Hessian matrix.

Analysis: Open CV provides a non-free module on SURF. But implementing it was very difficult in JAVA. Spent an entire day on reading this but couldn’t.

The SURF descriptor describes how pixel intensities are distributed within a scale dependent neighbourhood of each interest point detected by Fast Hessian

Object detection using SURF is scale and rotation invariant which makes it very powerful. Also it doesn’t require long and tedious training as in case of using cascaded haar classifier based detection. But the detection time of SURF is a little longer than Haar, but it doesn’t make much problem in most situations if the robot takes some tens of millisecond more for detection. Since this method is rotation invariant, it is possible to successfully detect objects in any orientation. This will be particularly useful in mobile robots where it may encounter situations in which it has to recognize objects which may be at different orientations than the trained image, say for example, the robot was trained with the upright image of an object and it has to detect a fallen object. Detection using haar features fails miserably in this case.

Error rate: Failure

Experiment: 8

Goal: To find the **contours** of the image by which the number of cavities present in the image can be found. This feature will helps us to differentiate most of the classes of the plankton.

Procedure:

* Find the contours of the image.
* Copy the image countour to a new variable.
* Paint the inside of the countour.
* Now everything inside the countour turns black and remaining outside turnd white.
* Compare this image to the original image.
* Divide the total number of white pixels from the original image to the black pixels of the painted image.

Analysis: Have found the contours for all the images. But the further process in progress. (Processing takes more time).

Error rate: Couldn’t complete- lack of time (no libraries)