Name: Remaldeep Singh

UID: u1143744

Email: [u1143744@utah.edu](mailto:u1143744@utah.edu)

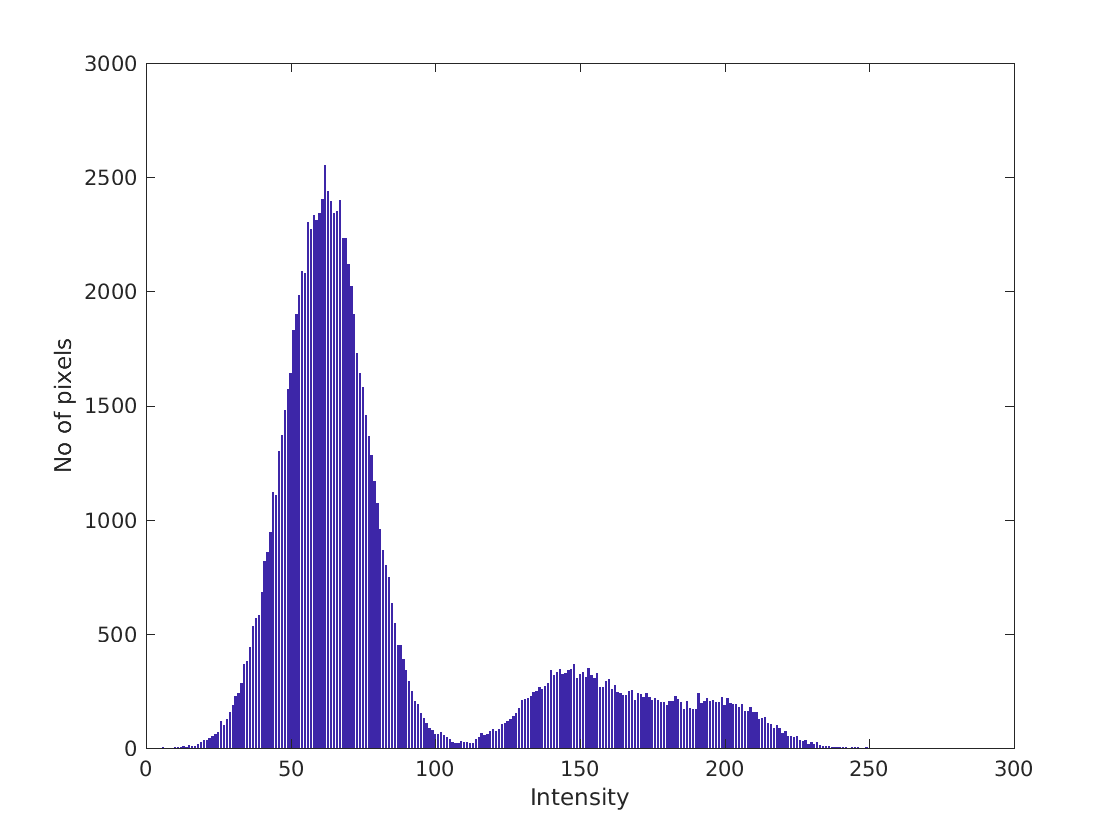
This project contains the necessary steps to process image from grayscale to denoising the image. Each step is mentioned under a different heading.

**Building a Histogram.**

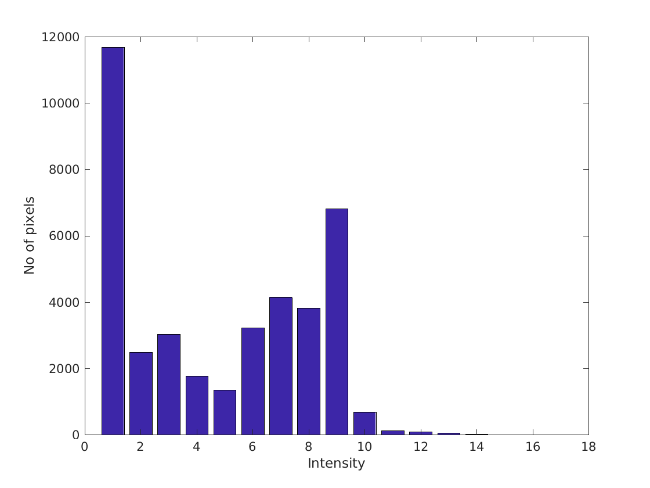
*This part is under the Step 1 of the main.m file.*

The function-histogram, in “Functions/histogram.m”, takes in the data, no of bins, min data value and max data value as the input.

It starts by finding the no of values in each bin (denoted by noBinValues). To normalize the pixal value to the respective bin, the minimum input is subtracted from the pixel value. Then the pixel value is divided by the noBinValues and ceiling it to get the bin number.

**C:\Users\Remal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\shapes_noise.tif**

In the above image, there is a lot of noise as a result of which the histogram of the image is gradual and spread out across the intensities. The shapes in the image are values of gray hence we see a small peak in the image around 150 and 200 intensity values. Whereas since most of the image is black with a lot of noise the highest peak is around 60.

C:\Users\Remal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\brain.tif

In the above image, the histogram has been made with 16 bins. Hence each bin is taking 16 intesity values. Since there is maximum of black in the image the values from 0-15, mapped to bin 1, are shown by the first bar of the histogram. Another high peak at the intensity level 9 is for the grayscale values of the brain. For the peak at intensity 3 it is because of the grey matter at the center of the brain.

In order to show the histogram I am using matlab bar(X,Y) function, that plots and array of values (Y) with respect to values in array X. I choose a figure for the bar(X,Y) to draw on, by the command “f1 = figure (1)”. xlabel diplays a label on x axis and ylabel diplays a label on y axis. Finally the saveas function saves the figure by taking the figure f1, name and fileformat as the input.

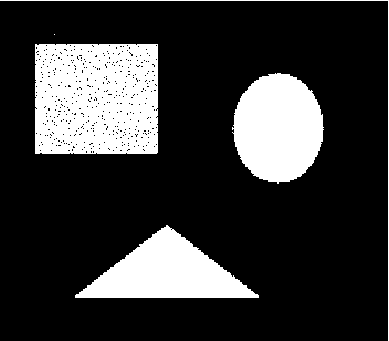
The output is in the folder “Outputs/histogram1.png”.

**Connected Components Analysis**

1. **Thresholding.**

*This part is under Step2.1 in the main.m file.*

The function-Threshold, in “Functions/Threshold.m”, takes an input image and two values. All the pixels between the two values become 255 and other pixels become 0. This function returns the final threshold image as the output. An example is shown below with thresholding of 120->256. The output is in outputs folder. “Outputs/Threshold1.png”.

C:\Users\Remal\AppData\Local\Microsoft\Windows\INetCache\Content.Word\shapes_noise.tif 

1. **Flood Fill.**

*This part is under the Step2.2 in the main.m file.*

The function-floodFill, in ”Functions/floodFill.m” takes as input a binary image, a seed values (x,y) for the starting position of the fill, the label value for the fill and labelOut for the output labels. This labelOut is returned by the function floodFill. FloodFill returns 4 variables.

1. A is the labelOut i.e the flood fill image with the label value.
2. B is the area of the labeled image.
3. C is the neighboring pixel that shares the maximum boundary with the labeled region. This is calculated by taking maximum count (i.e column 2 value) of the neighbourList.
4. D is the visitedNodes. This is used later on to check which pixels were visited during the floodFill i.e which labels were modified.

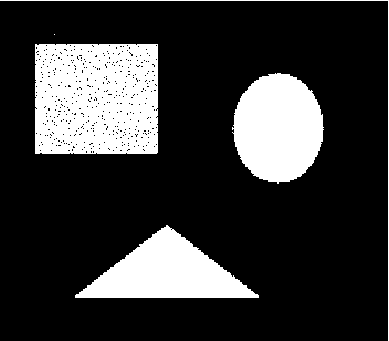
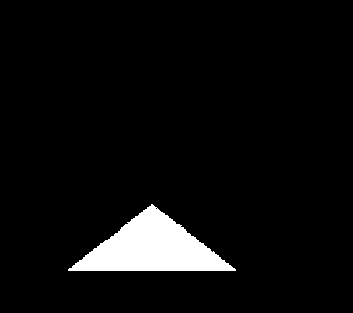
The floodFill.m contains two more functions.

1. boundCheck that checks if the seed is outside the Image boundary or not. This takes in Image and seed as input and returns 0 for outside and 1 for inside.
2. addNeighbour: This function is called whenever a neighbouring pixel of the current seed doesn’t have the same pixel value. This function helps maintain a list of neighbors with their label and count. This function returns the new updated neighbors list.

FloodFill works by maintaining a stack and keeps running until the stack is empty. The stack is empty when all the connected pixels of the first seed have been labeled. At every iteration of the while loop, visitedNodes is checked if the value at current position is not equal to 0 i.e this position has been visited and if it is true we skip the pixel/position. Then we create four more seeds (the algorithm is working in 4 connected region) from the original seed and mark that this position has been visited in the visitedNodes. We increment the area and label the values in labelOut as well. The four created seeds are then pushed on top of the stack based on the following conditions.

1. The boundCheck returns 1.
2. The pixel value at seedX(i.e new seed) is the same as pixel value at current position (i,j).
3. If the pixel value is not the same then that pixel is added to the neighbours list by calling addNeighbour function.

An example of flood fill working is below. The seed value is (262, 103) i.e 262 row and 103 column, which is somewhere inside the triangle shown in the left image. The output is in the folder “Outputs/floodFillImage.png”.

1. **Connected Component Routine.**

*This part is under the Step2.3 in the main.m file.*

The function-connectedComponent, in “Functions/connectedComponent.m”, takes the following inputs:

1. A binary image
2. Starting label for the components to start labeling from
3. A labelImage that is revised and populated inside the function and then returned as output.

ConnectedComponent has two outputs:

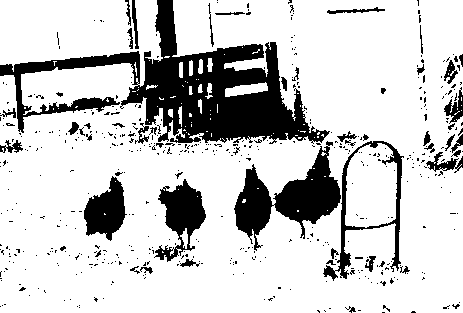
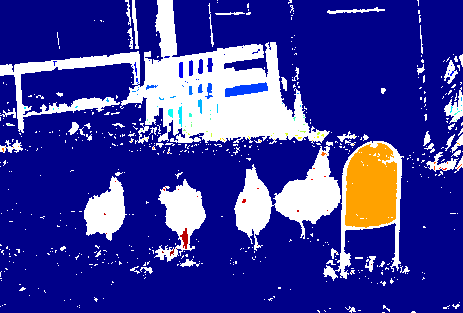
1. A = labeledOut: This is the revised labeled image.
2. B = labelInfo: This contains a list of all the connected regions in the output image. Moreover, each row contains:
   1. The starting position of the labeled region.
   2. The area of the labeled region.
   3. The label number of the region.
   4. The maximum boundary neighbor of the labeled region.

The algorithm starts from the top left pixel and keeps iterating to the right row wise until it encounters a pixel other than 0. It then calls flood fill on that pixel and then resumes iterating the pixels in the same fashion. If the program encounters a pixel that has already been labeled it skips that pixel by checking the visitedNodesG matrix and keeps iterating.

Connected Component maintains a visitedNodesG matrix that keeps track of which pixels have been visited by the floodfill of each label. It iteratively adds the visitedNodes output of the floodFill algorithm.

For a connected region, the output of floodfill will have 0 pixel values in the background except the pixel values of the label. That is only the connected pixels will have a value other than 0, rest all will be 0. Hence, we add the visitedNodes matrix to the visitedNodesG matrix.

An example of connected component labeling. More outputs are in the output folder “Outputs/connectedComponent.png”. There are some extra outputs in the extras folder as well.

**Denoising**

*This part is under Step3 in the main.m file.*

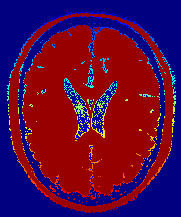
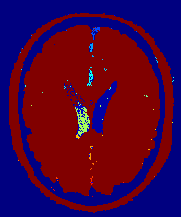
The function denoising, in “Functions/denoising.m”, takes the following inputs:

1. A labeled image.
2. Image Info containing all the connected components in the image, with their starting positions, the label value, area of region and max neighbor.
3. A threshold. All the components having area below the threshold will be denoised.

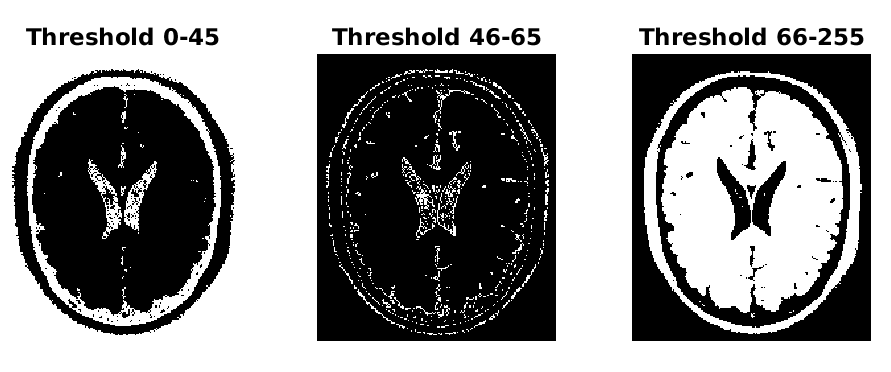
The function first calculates the maximum border neighbor for all the components with area less than the threshold and creates a newImInfo list, containing only those entries whose area is less than the threshold.

Based on the newImInfo list the algorithms floodfills the component regions again with the neighbor label as new label. The output is generated in the finalIm matrix. This finalIm is returned as the denoised image.

The following images show before and after denoising effect. The outputs are in the folder “Outputs/BeforeDenoising.png” and “Outputs/AfterDenoising.png”

**Thresholding:** Before processing an image with denoising it is important to threshold the image based on its gray values. Histogram will be better able to give you that intuition. For the brain image, the threshold values are 45 and 65. So we will have 3 threshold images in the ranges [0, 45], [46, 65], [66, 255]. And then connected components are found on these 3 images separately. The connected component images are then merged into one image having connected components of all the three images. This new image with new image info is then fed to the denoising algorithm. The outputs are in the folder “Outputs/DenoiseThresholds.png”.



**Improvements:**

1. **Smoothing:** One improvement I can think of is that blurring and adding the image to itself will remove the small artifacts present in the final image, although it will add a bloom effect but I think it is interesting nonetheless to give it a try. There are two ways to blur the image 1) Gaussian blurring and 2) Linear Blurring. With Gaussian blurring we will be able preserve the edges and get better smoothing.
2. **Denoise again:** Another improvement I can think of based on the above output is that maybe we can apply the denoising again and the remaining artifacts will go away.
3. **Number of Thresholds:** Another possible improvement to the thresholding step of denoising is that, based on the histogram we can apply multi guassian model to find out the thresholds necessary to segment the image. We might get more than 2 thresholds and apply connected component on all these images.

**Motion Detection:**

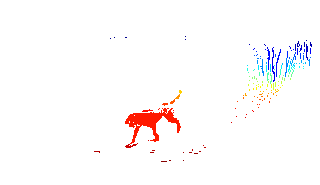
*This part is Step4 of the main.m file.*

In this part of the project there are two images one is background static image and the other is the dynamic image. The aim is to segment out the dog. The following steps are followed:

1. Convert the images from RGB to Grayscale.
2. Subtract the images to know which pixels have changed in intensity. While doing subtraction take the absolute difference. Outputs file: “MotionDetectionGray.png”.
3. Then see the histogram to find the most probable value of threshold. For me it came out to 25. Threshold image is shown below. Output file: “MotionDetectionThreshold.png”.

1. On the threshold image apply the connected component labeling and you will get the following output. Output file: “Outputs/MotionDetectionCCL.png”



1. On the connected component output after applying denoising with the threshold of 10 following output came out. Output file: “MotionDetectionDenoised.png”



On the final image, there is some grass left on the sides, but since the aim is to detect motion in the frames that is acceptable. Otherwise we can maybe remove the grass by checking the shape of the connected component and remove the elongated shapes.

**Challenges**

The main challenge of making this method of motion detection work is that it very sensitive to noise. We are subtracting images and based on that trying to find the motion. With a noisy signal this method might fail horribly. For example, if the given image was shot with a higher ISO (around 1600) then this image will have a lot more noise and motion will be detected pretty much everywhere.

Another challenge to this method is that if the object (to be detected) moves out of the frame then we cannot detect what movement happened, because the movement uncovered the background. This technique will not capture which way the motion is happening.

List of websites/resources used:

1. <https://www.mathworks.com/help/matlab/ref/imwrite.html>
2. <https://www.mathworks.com/help/matlab/ref/saveas.html>
3. <https://www.mathworks.com/help/matlab/creating_plots/add-title-axis-labels-and-legend-to-graph.html>
4. <https://www.mathworks.com/help/matlab/ref/histogram.html>
5. <https://www.mathworks.com/matlabcentral/answers/48938-delete-element-from-vector>
6. <https://www.mathworks.com/help/matlab/ref/abs.html>
7. <https://www.mathworks.com/help/matlab/ref/subplot.html>
8. <https://www.mathworks.com/help/matlab/ref/rgb2gray.html>
9. <https://blogs.mathworks.com/loren/2006/03/29/understanding-persistence/>
10. <https://www.mathworks.com/matlabcentral/answers/16799-best-method-to-clear-persistent-variables>
11. <http://blogs.mathworks.com/loren/2006/05/10/memory-management-for-functions-and-variables/>
12. <https://www.mathworks.com/help/images/getting-information-about-the-pixels-in-an-image.html>
13. <https://www.reddit.com/r/matlab/comments/25i2nr/recursion_with_a_global_variablepotentially_dumb/>
14. <https://www.mathworks.com/matlabcentral/answers/101556-is-it-possible-to-define-a-variable-as-static-within-a-matlab-matlab-file>
15. <http://www.matlabtips.com/recursive-functions/>
16. <https://www.mathworks.com/help/images/ref/imshow.html>
17. <https://www.mathworks.com/help/matlab/ref/ceil.html>
18. <https://www.mathworks.com/help/images/ref/mat2gray.html>
19. <https://www.mathworks.com/help/matlab/ref/figure.html>
20. <https://www.mathworks.com/help/matlab/learn_matlab/array-indexing.html>
21. <https://www.mathworks.com/matlabcentral/answers/328959-how-to-call-functions-from-another-m-file>
22. <https://www.mathworks.com/help/matlab/ref/addpath.html>
23. <https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html>
24. <https://www.mathworks.com/help/matlab/ref/cd.html>
25. <https://www.mathworks.com/help/stats/normal-distribution.html>
26. <https://www.cse.unr.edu/~fredh/papers/conf/034-asoidt/paper.pdf>