

Computer Vision CW

Abdullah Khalid

F220102

Contents

| | |
|---|----|
| Introduction: | 3 |
| Given Data:..... | 3 |
| Research:..... | 3 |
| Pipeline: | 5 |
| Data Import:..... | 6 |
| Resizing the Images:..... | 6 |
| Image Pre Processing: | 7 |
| Mean Filter..... | 7 |
| Gaussian Filter:..... | 8 |
| Gamma Correction..... | 10 |
| Feature Extraction:..... | 13 |
| Asymmetry: | 13 |
| Boundary Circularity: | 14 |
| Colour Consistency(HSV Colour space): | 16 |
| Groundtruth Variable:..... | 18 |
| SVM: | 18 |
| Failures & Experimentation: | 19 |
| Future Workings: | 21 |
| Bibliography | 22 |

Introduction:

Skin lesions are one of the most common dermatological issues faced by people worldwide. Early detection and identification of these lesions are crucial for timely treatment and management. With the advancements in the field of machine learning, various automated techniques have been developed to assist in the identification and diagnosis of skin lesions. In this report, we present our study of using machine learning techniques with MATLAB to predict the type of skin lesions based on the images of the lesions.

We were provided with a file of skin lesion images and ground truth labels for the same. We experimented with various machine learning techniques, including feature extraction, segmentation, and classification. Some of the techniques worked well in identifying the lesion types, while some did not provide accurate results.

Our study aimed to explore the potential of machine learning algorithms to accurately predict skin lesions, thus aiding in early detection and effective management of skin disorders. The results of our experiments suggest that a combination of various techniques, such as feature extraction and classification, can provide an accurate diagnosis of skin lesions. Overall, this study demonstrates the potential of machine learning algorithms in diagnosing skin disorders and highlights the need for further research in this area.

Given Data:

For a given assignment, we received three files: lesion images, lesion mask images, and a ground truth file. The lesion images and their corresponding masks were zipped files that contained 200 images. However, not all the lesion images were the same size. The lesion label file, which contained the list of all the files and their corresponding labels, was a simple text file.

It is important to note that while the lesion images and their corresponding masks were the same size, the images themselves varied in size. In order to accurately analyse the data, we needed to consider the size of each image when processing the information.

The ground truth was a critical component of the assignment, as it provided the necessary information to identify and label each image. By referencing this file, we were able to accurately match each image with its corresponding label and analyse the data accordingly.

Overall, while the files provided for the assignment presented some challenges due to the variation in image size, the inclusion of the lesion label file was a key factor in ensuring that we could effectively analyse and interpret the data.

Research:

To effectively analyse the lesion images provided for the project, we conducted research on pre-processing techniques and feature extraction methods. The goal was to identify the appropriate pre-processing techniques to enhance the quality of the images and extract key features that would help us accurately predict skin lesions.

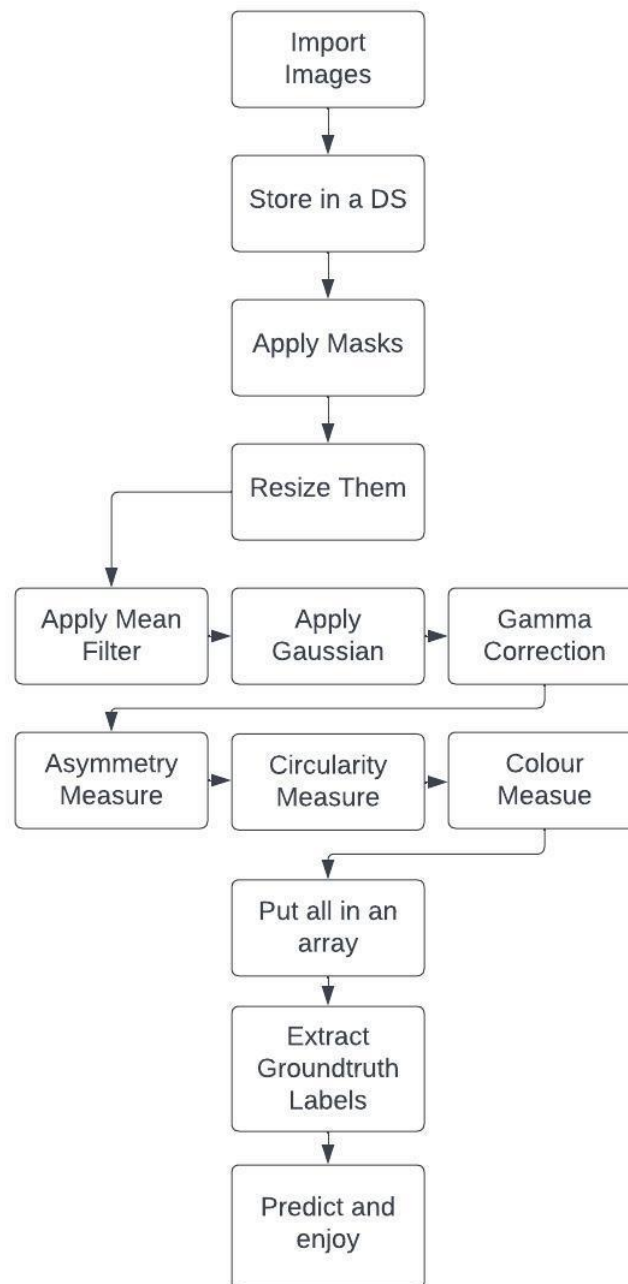
One of the main features we focused on was asymmetry (A), which refers to the unevenness or irregularity of the shape of the lesion. Another feature was circularity (B), which measures the degree to which the lesion resembles a perfect circle. The third feature we considered was colour (C), which was evaluated based on the hue, saturation, and intensity of the lesion's colours.

In order to extract these features, we utilized image processing techniques such as edge detection, smoothing, and colour segmentation. These techniques helped us to isolate the lesion from the surrounding skin and remove any noise or artifacts that could impact our analysis.

After applying these pre-processing techniques, we then extracted the A, B, and C features using a combination of shape analysis and colour analysis. By analysing these features, we were able to make accurate predictions about the nature of the skin lesion, such as whether it was benign or malignant.

Overall, our research on pre-processing and feature extraction techniques was critical in enabling us to accurately analyse and interpret the lesion images provided for the project. By extracting key features such as asymmetry, circularity, and colour, we were able to develop a more nuanced understanding of the nature of each lesion, which in turn helped us to make more informed diagnoses and predictions.

Pipeline:



The pipeline for this project starts with importing the images into the system. Next, the images are resized to a standard size to ensure consistency. Then, a mask is applied to extract the relevant portions of the image. The pre-processed images are then run through a series of filters to enhance their features. Next, features are extracted from the pre-processed images using a variety of techniques such as Histogram extraction, asymmetry, and boundary circularity. Once the model is trained, it can be used to predict values for new images that it has not seen before. The accuracy of the model can be evaluated by comparing its predictions to the actual values of the test dataset.

Data Import:

So first off we imported data and stored them into data stores. The Code that does so looks something like this.

```
%% Step 1 %%
% First off we will import all the necessary given data and store them in datastores %
Lesion_images = "C:\Users\abdul\Desktop\CV_CW\lesionimages";
Lesion_DS = imageDatastore(Lesion_images);
Mask_images = "C:\Users\abdul\Desktop\CV_CW.masks";
Mask_DS = imageDatastore(Mask_images);
Lesion_label = "C:\Users\abdul\Desktop\CV_CW\groundtruth.txt";
```

From here we'll work on these 2 datastores ,i.e., Lesion_DS and Mask_DS. Note that for lesion label we are only copying it's path, reason being we'll open file and handle labels later on.

Resizing the Images & Masking them:

So upon initial inspection we found out that the images are not the same size like the mask and the corresponding lesion images are same size but not all the lesion images are same. We need to fix that in order to perform proper analysis. Before we do that, we need to put mask images on top of the lesion images so that we have a base to work on.

```
function Masked_DS = maskim(Lesion_DS, Mask_DS)
imgs = readall(Lesion_DS); % read in all Lesion images
imgsm = readall(Mask_DS); % read in all mask images
newDatastore = "C:\Users\abdul\Desktop\CV_CW\Outputds";
outputSize = [224, 334];
for i=1:length(imgs)
    img=imgs{i}-(255-imgsm{i});
    % Resize the image
    %img = imresize(filing, outputSize);
    % Get the filename of the current image
    [~, filename, ext] = fileparts(Lesion_DS.Files{i});
    % Save the resized image with the same filename to the new datastore
    imwrite(img, fullfile(newDatastore, [filename, ext]))
end
Masked_DS=imageDatastore(newDatastore);
end
```

So in this function we are masking the images and storing them in a new datastore. Mind you one may dislike the fact that I am using so may datastores. Reason behind such behaviour is that so I can track and see differences in each and every step.

This function takes in two input datastores: Lesion_DS and Mask_DS, which respectively contain a set of lesion images and their corresponding binary mask images. It then reads all the images from these datastores, subtracts the pixel values of the mask images from 255 to invert the mask, and applies the inverted mask to the corresponding lesion image to generate the masked image. The masked images are then saved in a new output datastore named newDatastore.

The function returns a new imageDatastore Masked_DS that contains the masked images in newDatastore.

When we are storing files in a folder, the code makes sure that all the file names correspond with the original file names. This is to avoid confusion in the later stage.

Next after doing so, we need to resize the images. I wrote a new function that takes masked images and original Lesion images as input and returns resized versions of the images.

```
function [Resized_Lesions,Resized_Masks] = resize(Lesion_DS, Masked_DS)
imgs = readall(Lesion_DS); % read in all Lesion images
imgsm = readall(Masked_DS); % read in all masked images
resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Resiz_legions";
resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Resiz_mask";
outputSize = [350, 450];
for i=1:length(imgs)
    Resized_Lesions = imresize(imgs{i}, outputSize);
    Resized_Masks = imresize(imgsm{i}, outputSize);
    [~, filename, ext] = fileparts(Lesion_DS.Files{i});
    % Save the resized image with the same filename to the new datastore
    imwrite(Resized_Lesions, fullfile(resized_L_DS, [filename, ext]));
    imwrite(Resized_Masks, fullfile(resized_M_DS, [filename, ext]));
end
Resized_Lesions=imageDatastore(resized_L_DS);
Resized_Masks=imageDatastore(resized_M_DS);
end
```

The function takes in two image datastores Lesion_DS and Masked_DS which contain the lesion and mask images respectively. The function first reads in all the images from the two datastores using the readall function.

Next, the output size of the resized images is specified as [350, 450]. Then, a for-loop iterates over each image in the imgs and imgsm arrays and resizes each image using the imresize function to the output size.

After resizing, the function extracts the filename and extension of each image file from the original Lesion_DS and saves the resized images with the same filename to new datastores located in the paths resized_L_DS and resized_M_DS.

Finally, the function creates two new image datastores Resized_Lesions and Resized_Masks from the resized image datastores located at resized_L_DS and resized_M_DS respectively and returns them as output.

Mind you the reason I went for [350 by 450] is because during testing I found this to be the sweet spot. Anything more or less was yielding bad results.

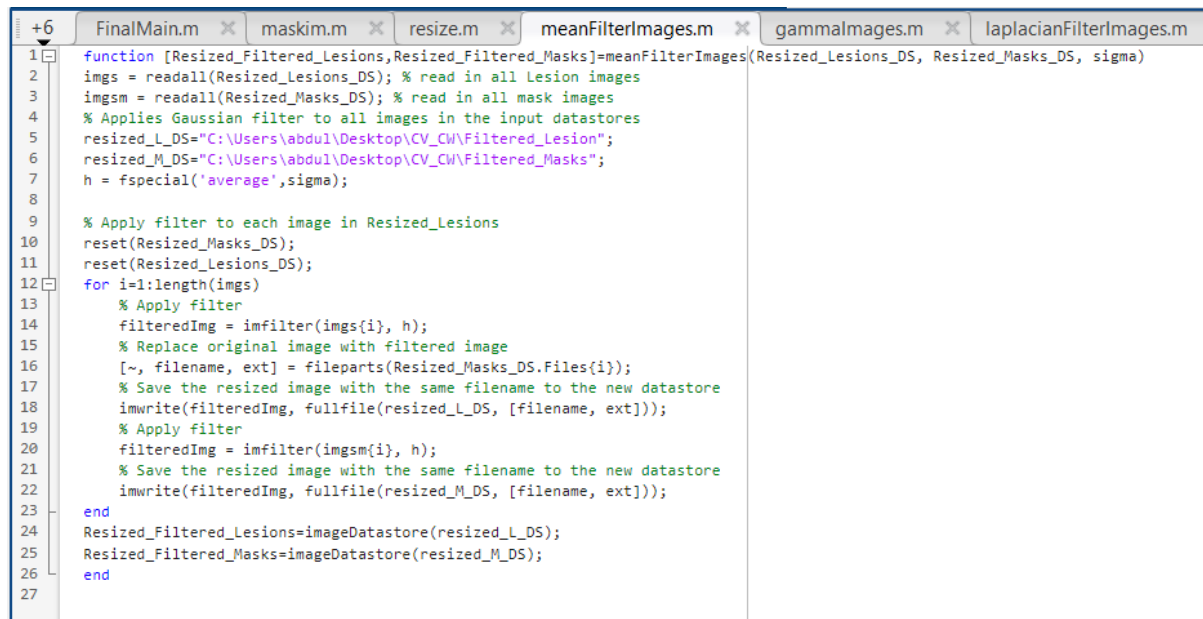
Image Pre Processing:

Now that we have the images resized and masked we need to apply some pre-processing filters that will enhance the image and increase the accuracy.

So, for this project we applied different pre-processing algorithms like gamma correction, mean filter, gaussian filter.

Mean Filter

So first off, we started off with mean filter. This we are using to enhance our image so that it gives us better accuracy.

The image shows a MATLAB code editor window with several tabs open: FinalMain.m, maskim.m, resize.m, meanFilterImages.m, gammalmImages.m, and laplacianFilterImages.m. The 'meanFilterImages.m' tab is active, displaying the following code:

```
1 function [Resized_Filtered_Lesions,Resized_Filtered_Masks]=meanFilterImages(Resized_Lesions_DS, Resized_Masks_DS, sigma)
2 imgs = readall(Resized_Lesions_DS); % read in all Lesion images
3 imgsm = readall(Resized_Masks_DS); % read in all mask images
4 % Applies Gaussian filter to all images in the input datastores
5 resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Lesion";
6 resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Masks";
7 h = fspecial('average',sigma);
8
9 % Apply filter to each image in Resized_Lesions
10 reset(Resized_Masks_DS);
11 reset(Resized_Lesions_DS);
12 for i=1:length(imgs)
13     % Apply filter
14     filteredImg = imfilter(imgs{i}, h);
15     % Replace original image with filtered image
16     [~, filename, ext] = fileparts(Resized_Masks_DS.Files{i});
17     % Save the resized image with the same filename to the new datastore
18     imwrite(filteredImg, fullfile(resized_L_DS, [filename, ext]));
19     % Apply filter
20     filteredImg = imfilter(imgsm{i}, h);
21     % Save the resized image with the same filename to the new datastore
22     imwrite(filteredImg, fullfile(resized_M_DS, [filename, ext]));
23 end
24 Resized_Filtered_Lesions=imageDatastore(resized_L_DS);
25 Resized_Filtered_Masks=imageDatastore(resized_M_DS);
26 end
27
```

The function takes in three inputs: Resized_Lesions_DS, Resized_Masks_DS, and sigma.

Resized_Lesions_DS and Resized_Masks_DS are imageDatastores that contain the input lesion and mask images, respectively. Sigma is a parameter that determines the size of the filter kernel used in the mean filter.

The function first reads in all the images from the input datastores using the readall function. It then applies a mean or average filter to each image using the imfilter function and a filter kernel created using the fspecial function with the 'average' option and sigma value specified.

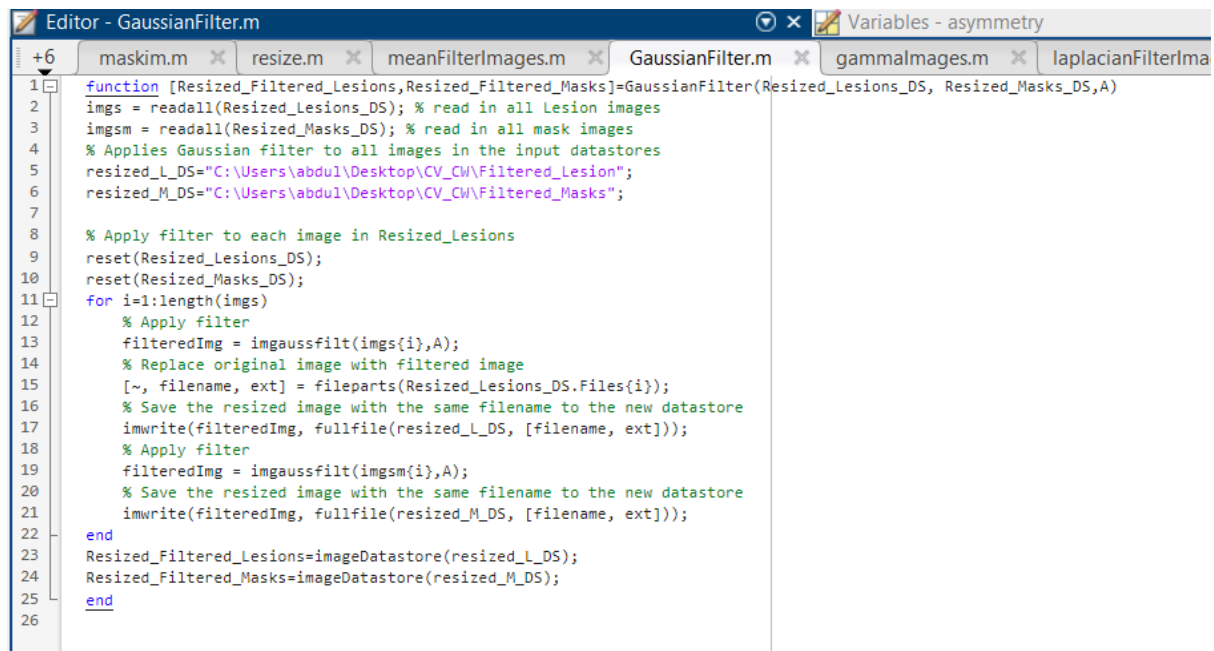
For each filtered image, the function saves the image with the same filename to new datastores for the filtered lesions and masks. The output of the function is the Resized_Filtered_Lesions and Resized_Filtered_Masks imageDatastores that contain the filtered lesion and mask images, respectively.

The function uses a for loop to iterate over each image in the input datastores, applying the filter to each image and saving the resulting filtered image to the new datastores. Finally, the function returns the new datastores containing the filtered images.

From there we move to Gaussian Filter.

Gaussian Filter:

During testing we saw the max accuracy we were getting was around 65%. There was something we needed to do to in order to take it above 70 percent. At this point I figured out that gaussian filter helps enhancing the image so we decided to try that out.



```
1 function [Resized_Filtered_Lesions,Resized_Filtered_Masks]=GaussianFilter(Resized_Lesions_DS, Resized_Masks_DS,A)
2 imgs = readall(Resized_Lesions_DS); % read in all Lesion images
3 imgsm = readall(Resized_Masks_DS); % read in all mask images
4 % Applies Gaussian filter to all images in the input datastores
5 resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Lesion";
6 resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Masks";
7
8 % Apply filter to each image in Resized_Lesions
9 reset(Resized_Lesions_DS);
10 reset(Resized_Masks_DS);
11 for i=1:length(imgs)
12     % Apply filter
13     filteredImg = imgaussfilt(imgs{i},A);
14     % Replace original image with filtered image
15     [~, filename, ext] = fileparts(Resized_Lesions_DS.Files{i});
16     % Save the resized image with the same filename to the new datastore
17     imwrite(filteredImg, fullfile(resized_L_DS, [filename, ext]));
18     % Apply filter
19     filteredImg = imgaussfilt(imgsm{i},A);
20     % Save the resized image with the same filename to the new datastore
21     imwrite(filteredImg, fullfile(resized_M_DS, [filename, ext]));
22 end
23 Resized_Filtered_Lesions=imageDatastore(resized_L_DS);
24 Resized_Filtered_Masks=imageDatastore(resized_M_DS);
25
26
```

The function takes in three inputs: Resized_Lesions_DS, Resized_Masks_DS, and A.

Resized_Lesions_DS and Resized_Masks_DS are imageDatastores that contain the input lesion and mask images, respectively. A is a parameter that determines the standard deviation of the Gaussian filter used.

The function first reads in all of the images from the input datastores using the readall function. It then applies a Gaussian filter to each image using the imgaussfilt function and the standard deviation A value specified.

For each filtered image, the function saves the image with the same filename to new datastores for the filtered lesions and masks. The output of the function is the Resized_Filtered_Lesions and Resized_Filtered_Masks imageDatastores that contain the filtered lesion and mask images, respectively.

The function uses a for loop to iterate over each image in the input datastores, applying the filter to each image and saving the resulting filtered image to the new datastores.

In the for loop, the filteredImg variable is assigned the result of applying the Gaussian filter to the current image. The imgaussfilt function smooths the input image using a Gaussian kernel, which is a bell-shaped curve used to distribute weights to the neighboring pixels. The standard deviation parameter A controls the amount of smoothing applied, with larger values of A producing more blur and smoother images.

The function then extracts the filename and extension of the current image and saves the filtered image with the same filename to the new datastores for the filtered lesions and masks using the imwrite function.

Finally, the function returns the new datastores containing the filtered images.

In summary, this function applies a Gaussian filter to a set of input images, saves the filtered images to new datastores, and returns these new datastores. The Gaussian filter is applied using the imgaussfilt function with a standard deviation parameter A, which controls the amount of smoothing applied to the input images. The function uses a for loop to iterate over each image in the input

datastores and applies the filter to each image before saving the filtered image to the new datastores.

Lastly, we'll move on to the Gamma Correction.

Gamma Correction

```
function [Resized_Filtered_Lesions_2,Resized_Filtered_Masks_2]=gammaImages(Resized_Filtered_Lesions, Resized_Filtered_Masks,gamma)
imgs = readall(Resized_Filtered_Lesions); % read in all Lesion images
imgsm = readall(Resized_Filtered_Masks); % read in all mask images
% Applies Gaussian filter to all images in the input datastores
resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Lesion";
resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Masks";

% Apply filter to each image in Resized_Lesions
reset(Resized_Filtered_Lesions);
reset(Resized_Filtered_Masks);
for i=1:length(imgs)
    % Apply filter
    filteredImg = imadjust(imgs{i},[],[],gamma);
    % Replace original image with filtered image
    [~, filename, ext] = fileparts(Resized_Filtered_Lesions.Files{i});
    % Save the resized image with the same filename to the new datastore
    imwrite(filteredImg, fullfile(resized_L_DS, [filename, ext]));
    % Apply filter
    filteredImg = imadjust(imgsm{i},[],[],gamma);
    % Save the resized image with the same filename to the new datastore
    imwrite(filteredImg, fullfile(resized_M_DS, [filename, ext]));
end
Resized_Filtered_Lesions_2=imageDatastore(resized_L_DS);
Resized_Filtered_Masks_2=imageDatastore(resized_M_DS);
end
```

The `gammaImages` function takes in two `imageDatastores` `Resized_Filtered_Lesions` and `Resized_Filtered_Masks` which contain filtered lesion images and their corresponding masks respectively. The function then applies gamma correction to the images and saves them in new `imageDatastores` `Resized_Filtered_Lesions_2` and `Resized_Filtered_Masks_2`.

Gamma correction is a non-linear operation that adjusts the brightness of an image. In this function, the `imadjust` function is used to apply gamma correction to each image in the input `imageDatastores`. The `imadjust` function adjusts the intensities of the input image to enhance the contrast. The gamma input parameter is used to adjust the gamma value of the image. A higher gamma value will result in a darker image while a lower gamma value will result in a brighter image.

The function reads all images from the input `imageDatastores` using the `readall` function. It then loops through each image and applies gamma correction using the `imadjust` function. The resulting image is saved in a new directory for lesion images and their corresponding masks respectively. The `fileparts` function is used to extract the file name and extension from the input `imageDatastore`. The resulting filtered and adjusted images are saved in the new directory with the same file name and extension as the original image.

Finally, the function creates new `imageDatastores` `Resized_Filtered_Lesions_2` and `Resized_Filtered_Masks_2` that contain the filtered and adjusted images in the new directory. These `imageDatastores` can be used for further analysis.

In summary, the `gammaImages` function applies gamma correction to lesion images and their corresponding masks and saves them in a new directory. The function can be used to enhance the contrast of the images and improve the accuracy of lesion detection algorithms.

So, in order to get the context, we'll see before and after images.

Before:



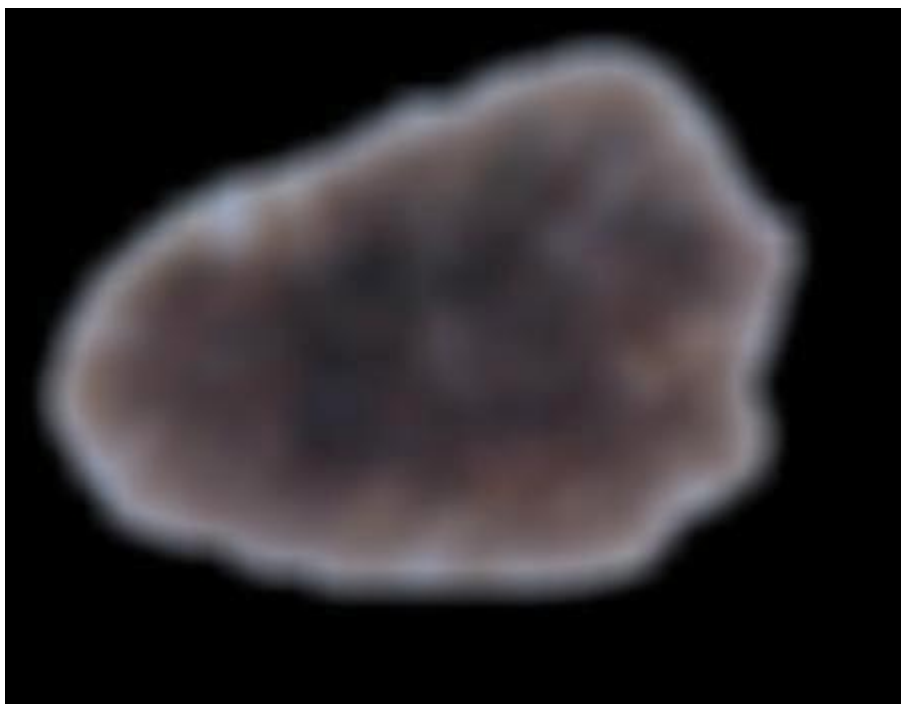
After:



Masked Before:



Masked After:



So we can see the difference, maybe for human eye it may be a wee bit weird but computer sees things different than us.

After all that lets move to feature extraction.

Feature Extraction:

So the main part of the whole project is the feature extraction part. In this part we extract the features and put them into the imfeature matrix.

First off, we'll start with the Asymmetry of the picture.

Asymmetry:

```
function asymmetry= asymmetry_measure(Resized_Filtered_Masks)
imgs = readall(Resized_Filtered_Masks);
asymmetry = zeros(numel(Resized_Filtered_Masks),1);
for i = 1:numel(Resized_Filtered_Masks.Files)
    % Calculate the asymmetry
    img_rotated = imrotate(imgs{i}, 180,'crop');
    img_flipped = flip(img_rotated, 2);
    diff = abs(double(imgs{i}) - double(img_flipped));
    asymmetry(i,:) = sum(diff(:)) / (numel(imgs{i}) * 255);
end
asymmetry_norm = (asymmetry - min(asymmetry)) / (max(asymmetry) - min(asymmetry));
asymmetry = asymmetry_norm;
```

This code defines a function named `asymmetry_measure` that takes in a parameter `Resized_Filtered_Masks`. The function reads all images in the given folder using `readall` and stores them in the variable `imgs`. The number of images found in the folder is determined using `numel`.

The function then initializes a zero-filled matrix called `asymmetry` with the same size as the number of images found. It then enters a loop that processes each image in the folder one at a time.

For each image, the code first rotates the image by 180 degrees using `imrotate`, then flips it horizontally using `flip`. It then calculates the absolute difference between the original image and the flipped image and stores the result in a variable called `diff`.

Finally, the code calculates the asymmetry of the image by summing all the values in the difference matrix and dividing the result by the total number of pixels in the image multiplied by 255. This value is then stored in the corresponding index of the `asymmetry` matrix.

After processing all images, the `asymmetry` matrix is normalized between 0 and 1 and returned as the output of the function.

The result of this function results in a matrix that looks something like as shown below.

| asymmetry | |
|--------------|--------|
| 200x1 double | |
| | 1 |
| 1 | 0.3733 |
| 2 | 0.0386 |
| 3 | 0.3845 |
| 4 | 0.3131 |
| 5 | 0.0660 |
| 6 | 0.0794 |
| 7 | 0.2472 |
| 8 | 0.2333 |
| 9 | 0.0201 |
| 10 | 0.1601 |
| 11 | 0.4706 |
| 12 | 0.0564 |
| 13 | 0.2822 |
| 14 | 0.2702 |
| 15 | 0.3450 |

After this we'll move to B, which is Boundary circularity.

Boundary Circularity:

This function helps us find a critical feature which is how circular is the lesion part of the image.

```

Editor - circularity_measure.m
+7 meanFilterImages.m x GaussianFilter.m x gammalmImages.m x laplacianFilterImages.m x asymmetry_measure.m x circularity_measure.m x
1 function circularity = circularity_measure(Resized_Filtered_Masks)
2 imgs = readall(Resized_Filtered_Masks);
3 circularity = zeros(numel(Resized_Filtered_Masks),1);
4 for i = 1:numel(Resized_Filtered_Masks.Files)
5     % Calculate the circularity
6     % Convert the image to grayscale
7     gray_img = rgb2gray(imgs{i});
8     % Threshold the image using Otsu's method
9     threshold = graythresh(gray_img);
10    binary_img = imbinarize(gray_img, threshold);
11    % Get the properties of the binary image regions
12    stats = regionprops(binary_img, 'Area', 'Perimeter');
13    % Calculate the circularity using the area and perimeter
14    circls=(4 * pi * [stats.Area] ./ ([stats.Perimeter].^2));
15    circls(isinf(circls)) = 1;
16    % Calculate the mean circularity of all the regions
17    circularity(i,:) = mean(circls);
18 end
19 circularity_norm = (circularity - min(circularity)) / (max(circularity) - min(circularity));
20 circularity = circularity_norm;

```

The input argument Resized_Filtered_Masks is a datastore object containing the paths of the binary mask images. The output of the function is a normalized circularity measure for each image in the input datastore.

The function first reads all the images in the input datastore using the readall function and stores them in the imgs cell array. It then initializes a zero vector of size equal to the number of images in the input datastore to store the circularity measure of each image.

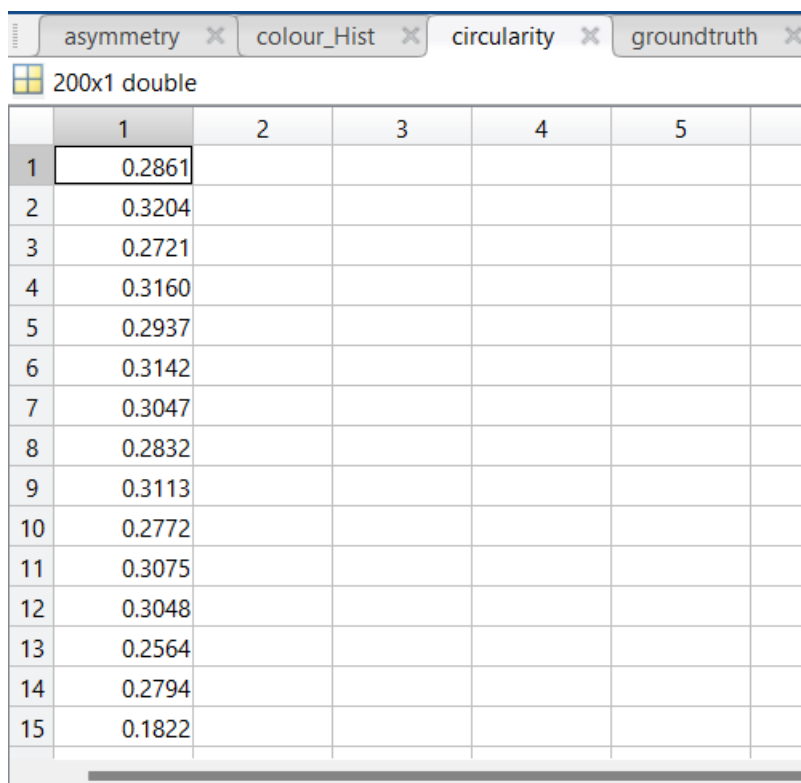
The code then loops through each image in the input datastore and performs the following steps to calculate its circularity measure:

Convert the image to grayscale using the `rgb2gray` function.

- Threshold the grayscale image using Otsu's method to convert it into a binary image.
- Compute the region properties (area and perimeter) of the binary image regions using the `regionprops` function.
- Calculate the circularity measure of each region using the formula $(4 * \pi * \text{area} / \text{perimeter}^2)$, where area and perimeter are the region properties computed in the previous step.
- Calculate the mean circularity measure of all the regions in the image and store it in the circularity vector for that image.
- Finally, the circularity measures are normalized to the range [0,1] using the min-max normalization method and returned as the output of the function.

In summary, this function provides a quick and efficient way to compute the circularity measure of a set of binary mask images.

The output of this function looks something like this.

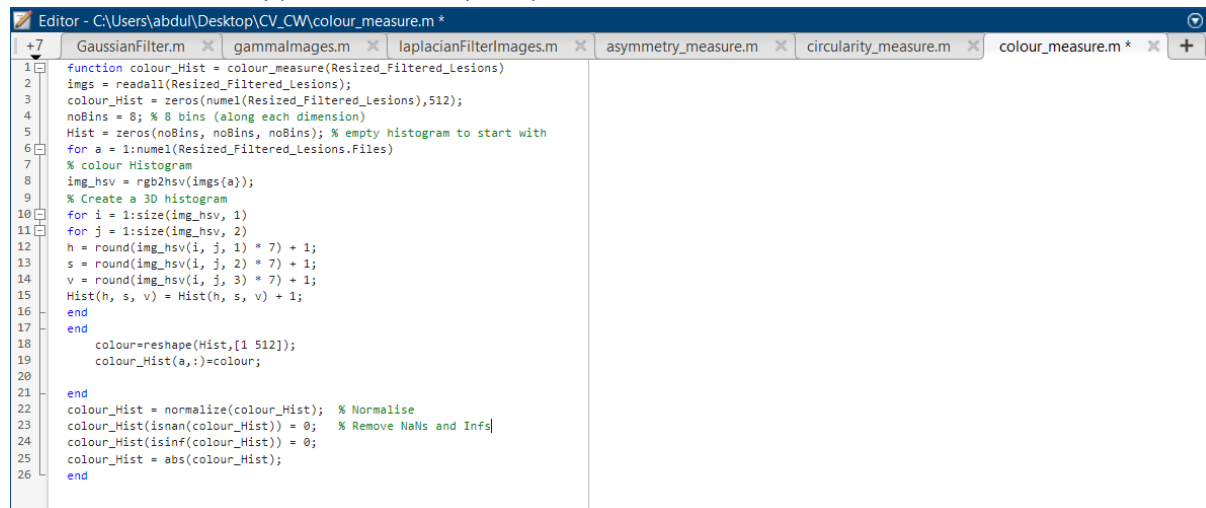


| | 1 | 2 | 3 | 4 | 5 | |
|----|--------|---|---|---|---|--|
| 1 | 0.2861 | | | | | |
| 2 | 0.3204 | | | | | |
| 3 | 0.2721 | | | | | |
| 4 | 0.3160 | | | | | |
| 5 | 0.2937 | | | | | |
| 6 | 0.3142 | | | | | |
| 7 | 0.3047 | | | | | |
| 8 | 0.2832 | | | | | |
| 9 | 0.3113 | | | | | |
| 10 | 0.2772 | | | | | |
| 11 | 0.3075 | | | | | |
| 12 | 0.3048 | | | | | |
| 13 | 0.2564 | | | | | |
| 14 | 0.2794 | | | | | |
| 15 | 0.1822 | | | | | |

Note: I tried not using `regionprops` but had little to no luck what so ever.

Next up we have colour consistency. We used 3D histograms for that purpose.

Colour Consistency(HSV Colour space):



```
1 function colour_Hist = colour_measure(Resized_Filtered_Lesions)
2   imgs = readall(Resized_Filtered_Lesions);
3   colour_Hist = zeros(numel(Resized_Filtered_Lesions),512);
4   noBins = 8; % 8 bins (along each dimension)
5   Hist = zeros(noBins, noBins, noBins); % empty histogram to start with
6   for a = 1:numel(Resized_Filtered_Lesions.Files)
7       % colour Histogram
8       img_hsv = rgb2hsv(imgs{a});
9       % Create a 3D histogram
10      for i = 1:size(img_hsv, 1)
11          for j = 1:size(img_hsv, 2)
12              h = round(img_hsv(i, j, 1) * 7) + 1;
13              s = round(img_hsv(i, j, 2) * 7) + 1;
14              v = round(img_hsv(i, j, 3) * 7) + 1;
15              Hist(h, s, v) = Hist(h, s, v) + 1;
16          end
17      end
18      colour = reshape(Hist, [1 512]);
19      colour_Hist(a,:) = colour;
20
21  end
22  colour_Hist = normalize(colour_Hist); % Normalise
23  colour_Hist(isnan(colour_Hist)) = 0; % Remove NaNs and Infs
24  colour_Hist(isinf(colour_Hist)) = 0;
25  colour_Hist = abs(colour_Hist);
26  end
```

The code is implementing a function called "colour_measure" that takes in a set of images of skin lesions, "Resized_Filtered_Lesions", and outputs a color histogram values for each image. The purpose of the function is to capture the distribution of colors in the images, which can be useful for characterizing skin lesions.

First, the function reads in all the images in "Resized_Filtered_Lesions" using the "readall" function, which returns a cell array of image data. Then, an empty matrix called "colour_Hist" is initialized with the size of the number of images in "Resized_Filtered_Lesions" by 512, where 512 is the total number of bins in the color histogram (8 bins for each of the three-color channels: hue, saturation, and value).

The variable "noBins" is set to 8, which means that the range of each color channel is divided into 8 equal-sized bins. An empty 3D matrix called "Hist" is initialized with dimensions "noBins" x "noBins" x "noBins" to hold the histogram data.

A loop is then set up to iterate through each image in "Resized_Filtered_Lesions". For each image, it is converted from RGB to the HSV color space using the "rgb2hsv" function. The code then loops through each pixel in the image and calculates the index of the corresponding bin in the histogram for the pixel's hue, saturation, and value values. The counts for each bin are accumulated in the "Hist" matrix.

After looping through all the pixels in the image, the histogram data is reshaped into a 1D row vector called "colour". This vector is then stored in the corresponding row of "colour_Hist" for the current image.

The final step is to normalize "colour_Hist" using the "normalize" function to scale the data so that the median is centered at 0 and the data is scaled to have a median absolute deviation of 1. Any NaN or Inf values are set to 0, and the absolute value of the resulting matrix is taken. The function then returns the "colour_Hist" matrix.

In summary, the "colour_measure" function takes a set of skin lesion images, converts them to the HSV color space, and calculates a color histogram for each image using 8 bins for each of the three-color channels. The resulting histogram data is then normalized and returned as a matrix.

Let's see what the output looks like.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | : |
|----|--------|---|---|---|---|---|---|---|---|--------|--------|--------|--------|--------|--------|----|--------|--------|--------|--------|--------|---|
| 1 | 1.6343 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.0419 | 2.4724 | 2.2872 | 1.2004 | 1.4480 | 2.2939 | 0 | 1.0289 | 1.2909 | 2.8831 | 2.7319 | 0.9577 | |
| 2 | 1.6343 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.0419 | 2.4724 | 2.2872 | 1.2004 | 1.4480 | 2.2939 | 0 | 1.0289 | 1.2909 | 2.8831 | 2.7319 | 0.9577 | |
| 3 | 1.6298 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.9868 | 2.4724 | 2.2140 | 1.1515 | 1.4240 | 2.2939 | 0 | 1.0289 | 1.2875 | 2.8048 | 2.7089 | 0.8903 | |
| 4 | 1.0918 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.8515 | 2.3185 | 1.8300 | 1.1271 | 1.1113 | 1.3843 | 0 | 1.0289 | 1.0900 | 1.7675 | 1.2084 | 0.8565 | |
| 5 | 1.0828 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.8232 | 1.5931 | 1.7446 | 1.0124 | 1.1035 | 1.3843 | 0 | 1.0289 | 1.0766 | 1.6305 | 1.2084 | 0.8533 | |
| 6 | 1.0759 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7854 | 1.5711 | 1.7325 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0577 | 1.6110 | 1.2084 | 0.7778 | |
| 7 | 1.0759 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7854 | 1.5711 | 1.7325 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0577 | 1.6110 | 1.2084 | 0.7778 | |
| 8 | 1.0758 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7854 | 1.5711 | 1.7325 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0577 | 1.6110 | 1.2084 | 0.7778 | |
| 9 | 1.0758 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7854 | 1.5711 | 1.7325 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0577 | 1.6110 | 1.2084 | 0.7778 | |
| 10 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 11 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 12 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 13 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 14 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 15 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 16 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 17 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 18 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 19 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 20 | 1.0174 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7020 | 1.5491 | 1.4612 | 0.9196 | 1.0877 | 1.3373 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0877 | 0.7778 | |
| 21 | 0.9810 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.6847 | 1.5491 | 1.3880 | 0.8903 | 1.0662 | 1.1173 | 0 | 1.0289 | 1.0342 | 1.5718 | 1.0590 | 0.7650 | |
| 22 | 0.9454 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5226 | 1.5491 | 1.2204 | 0.8829 | 1.0649 | 0.8281 | 0 | 1.0289 | 0.9885 | 1.5718 | 0.9727 | 0.7569 | |
| 23 | 0.9454 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5226 | 1.5491 | 1.2204 | 0.8829 | 1.0649 | 0.8281 | 0 | 1.0289 | 0.9885 | 1.5718 | 0.9727 | 0.7569 | |
| 24 | 0.9454 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5226 | 1.5491 | 1.2204 | 0.8829 | 1.0649 | 0.8281 | 0 | 1.0289 | 0.9885 | 1.5718 | 0.9727 | 0.7569 | |
| 25 | 0.9454 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5226 | 1.5491 | 1.2204 | 0.8829 | 1.0649 | 0.8281 | 0 | 1.0289 | 0.9885 | 1.5718 | 0.9727 | 0.7569 | |
| 26 | 0.9454 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.5226 | 1.5491 | 1.2204 | 0.8829 | 1.0649 | 0.8281 | 0 | 1.0289 | 0.9885 | 1.5718 | 0.9727 | 0.7569 | |

Now that we have all 3 infinity stones, now is the time to put them infinity gauntlet. Oh, I mean put all 3 features to imfeature matrix. Let's recall what's going on We have asymmetry matrix, which is 200x1, next up we have circularity which is also 200x1 and then we have colour consistency matrix which is 200x512.

HSV histogram was performing better so I used it instead of RGB histogram.

```
%% Step 4 %%
% Start working on imfeature Matrix %
% Calculate Asymmetry %
asymmetry = asymmetry_measure(Resized_Filtered_Masks_2);
% Calculate circularity %
circularity = circularity_measure(Resized_Filtered_Masks_2);
% Colour Values using 3d histogram %
colour_Hist = colour_measure(Resized_Filtered_Lesions1);
imfeature = [asymmetry,circularity,colour_Hist];
%% Step 5 %%
```

In the main part of the function we are just calling the functions and storing the returned values into a matrix called imfeature. The resulting imfeature matrix is a 200x514 matrix.

Before we run SVM we need to store labels into a vector. For SVM the labels should be in a vector rather than a matrix or anything. So, for that we created a function.

Groundtruth Variable:

```
gammalimages.m x laplacianFilterImages.m x asymmetry_measure.m x circularity_measure.m x co
function groundtruth = grt_to_vec(Lesion_label)
% Open the file containing the groundtruth labels
fid = fopen(Lesion_label, 'r');
% Initialize an empty cell array to store the labels
groundtruth = cell(200, 1);
% Read the file line by line and extract the label from each line
for i = 1:200
    line = fgetl(fid);
    comma_idx = strfind(line, ',');
    label = line(comma_idx+1:end);
    groundtruth{i} = label;
end
% Close the file
fclose(fid);
groundtruth=groundtruth(:);
end
```

In this function we are opening the 'Lesion_Label' file, creating a ground_truth empty array and then with the help of for loop we are going through every line of the text file. So basically in the function we are reading a line, and storing whatever is next to a ',' in our case it's the label. We are storing the label into a matrix and then closing the file. To convert the matrix to a vector we are just doing `groundtruth=groundtruth(:)`

Now since our imfeature matrix and groundtruth vector is ready we'll run the SVM.

SVM:

So, this SVM implementation is a very textbook implementation.

```
%% Step 6 %%
rng(1);% let's all use the same seed for the random number generator
svm = fitcsvm(imfeature, groundtruth);
cvsvm = crossval(svm);
pred = kfoldPredict(cvsvm);
[cm, order] = confusionmat(groundtruth, pred);
```

Here's a brief explanation of each line of the code:

- **rng(1);**: This sets the seed for the random number generator to ensure reproducibility of the results across different runs of the code.
- **svm = fitcsvm(imfeature, groundtruth);**: This line trains an SVM classifier using the `fitcsvm` function. `imfeature` is a matrix of features (predictors) and `groundtruth` is a vector of class labels (0 or 1) for the binary classification task.
- **cvsvm = crossval(svm);**: This line performs k-fold cross-validation on the trained SVM model using the `crossval` function. The default number of folds is 10.

- **pred = kfoldPredict(cvsvm);** : This line generates predictions for the test data using the trained SVM model and cross-validation. The kfoldPredict function outputs a vector of predicted class labels (0 or 1) for each test instance.
- **[cm, order] = confusionmat(groundtruth, pred);** This line computes the confusion matrix cm and the order of class labels order using the confusionmat function. The confusion matrix is a table that shows the number of true positives, true negatives, false positives, and false negatives, which can be used to evaluate the performance of the classifier.

Our results were rather amazing.

| | 1 | 2 |
|---|----|----|
| 1 | 95 | 5 |
| 2 | 12 | 88 |
| 3 | | |

We got an accuracy of **91.5%** which is very high. But we got here after some experimentation and testing.

Our sensitivity is 95.

Specificity is 88.

If you are curious our Main Function looks like this.

```
%% Welcome to the Program %%
% Made By Abdullah Khalid %
% F220102 %

%% Step 1 %%
% First off we will import all the necessary given data and store them in datastores %
Lesion_images = "C:\Users\abdul\Desktop\CV_CW\lesionimages";
Lesion_DS = imageDatastore(Lesion_images);
Mask_images = "C:\Users\abdul\Desktop\CV_CW\masks";
Mask_DS = imageDatastore(Mask_images);
Lesion_label = "C:\Users\abdul\Desktop\CV_CW\groundtruth.txt";

%% Step 2 %%
% Mask and resize all the images %
Masked_DS=maskim(Lesion_DS,Mask_DS);
[Resized_Lesions,Resized_Masks]=resize(Lesion_DS, Masked_DS);

%% Step 3 %%
% Apply Gaussian Filter & laplacian Filter to all the images %
% CC_DS = applyCC(Resized_Lesions);
[Resized_Filtered_Lesions,Resized_Filtered_Masks]=meanFilterImages(Resized_Lesions, Resized_Masks, [20 20] );
[Resized_Filtered_Lesions1,Resized_Filtered_Masks1]=GaussianFilter(Resized_Filtered_Lesions,Resized_Filtered_Masks,1);
[Resized_Filtered_Lesions_2,Resized_Filtered_Masks_2]=gammaImages(Resized_Filtered_Lesions1, Resized_Filtered_Masks,0.9);

%% Step 4 %%
% Start working on infeature Matrix %
% Calculate Asymmetry %
asymmetry = asymmetry_measure(Resized_Filtered_Masks_2);
% Calculate circularity %
circularity = circularity_measure(Resized_Filtered_Masks_2);
% Colour Values using 3d histogram %
colour_Hist = colour_measure(Resized_Filtered_Lesions1);
infeature = [asymmetry,circularity,colour_Hist];

%% Step 5 %%
groundtruth = grt_to_vec(Lesion_label);

%% Step 6 %%
rng(1);% let's all use the same seed for the random number generator
svm = fitcsvm(infeature, groundtruth);
cvsvm = crossval(svm);
pred = kfoldPredict(cvsvm);
[cm, order] = confusionmat(groundtruth, pred);
Sensitivity=(cm(1,1)/(cm(1,1)+cm(1,2)))*100;
```

Failures & Experimentation:

"It is impossible to live without failing at something unless you live so cautiously that you might as well not have lived at all, in which case you have failed by default." — J.K. Rowling

During the course of this project we faced some major challenges and failures. Let's talk about some of the major failures we faced during this project.

Our first failure was when we tried to use Laplacian Filter.

```
function [Resized_Filtered_Lesions_3,Resized_Filtered_Masks_3]=laplacianFilterImages(Resized_Filtered_Lesions_2,Resized_Filtered_Masks_2)
imgs = readall(Resized_Filtered_Lesions_2); % read in all Lesion images
imgsm = readall(Resized_Filtered_Masks_2); % read in all mask images
% Applies Gaussian filter to all images in the input datastores
resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Lesion";
resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Masks";
laplacian_filter = fspecial('laplacian',0.9);
for i=1:length(imgs)
    filtered_image = imfilter(imgs{i}, laplacian_filter);
    subtracted_image = imsubtract(imgs{i}, filtered_image);
    % Replace original image with filtered image
    [~, filename, ext] = fileparts(Resized_Filtered_Lesions_2.Files{i});
    % Save the resized image with the same filename to the new datastore
    imwrite(subtracted_image, fullfile(resized_L_DS, [filename, ext]));
    filtered_image = imfilter(imgsm{i}, laplacian_filter);
    subtracted_image = imsubtract(imgsm{i}, filtered_image);
    % Save the resized image with the same filename to the new datastore
    imwrite(subtracted_image, fullfile(resized_M_DS, [filename, ext]));
end
Resized_Filtered_Lesions_3=imageDatastore(resized_L_DS);
Resized_Filtered_Masks_3=imageDatastore(resized_M_DS);
end
```

I created this function placed it before mean function, but the results were bad. The accuracy fell to around 50% which is bad. To set a benchmark for accuracy We ran the code without any pre-processing and the results were around 65%. Okay dropping the Laplacian filter I decided to go with Mean and Gamma correction only. It worked well, Increased the accuracy but I felt as if I could do more to increase the accuracy. The accuracy with mean filter and gamma correction was around 67%. One thing I had to play with for hours was the filter size in mean filter. After vigorous testing I found the best size to be [20 20]. After all this I decided to add Gaussian Filter.

```

Editor - GaussianFilter.m
+6 maskim.m x resize.m x meanFilterImages.m x GaussianFilter.m x gammalimages.m x laplacianFilterma
1 function [Resized_Filtered_Lesions,Resized_Filtered_Masks]=GaussianFilter(Resized_Lesions_DS, Resized_Masks_DS,A)
2 imgs = readall(Resized_Lesions_DS); % read in all Lesion images
3 imgsm = readall(Resized_Masks_DS); % read in all mask images
4 % Applies Gaussian filter to all images in the input datastores
5 resized_L_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Lesion";
6 resized_M_DS="C:\Users\abdul\Desktop\CV_CW\Filtered_Masks";
7
8 % Apply filter to each image in Resized_Lesions
9 reset(Resized_Lesions_DS);
10 reset(Resized_Masks_DS);
11 for i=1:length(imgs)
12     % Apply filter
13     filteredImg = imgaussfilt(imgs{i},A);
14     % Replace original image with filtered image
15     [~, filename, ext] = fileparts(Resized_Lesions_DS.Files{i});
16     % Save the resized image with the same filename to the new datastore
17     imwrite(filteredImg, fullfile(resized_L_DS, [filename, ext]));
18     % Apply filter
19     filteredImg = imgaussfilt(imgsm{i},A);
20     % Save the resized image with the same filename to the new datastore
21     imwrite(filteredImg, fullfile(resized_M_DS, [filename, ext]));
22 end
23 Resized_Filtered_Lesions=imageDatastore(resized_L_DS);
24 Resized_Filtered_Masks=imageDatastore(resized_M_DS);
25 end
26

```

When I was applying the gaussian filter, at first, I was using fspecial to apply the filter but then I found out that using imgaussfilt is a better solution (fspecial Create predefined 2-D filter, n.d.). Standard deviation around 1 yielded the best results. Next, gamma correction was simple. The only thing I had to play with was the sigma value and I found that the best sigma value was 0.9.

After this we moved on to the feature extraction part.

In feature extraction we hit fairly low issues and there was not much room to play with in the first place. So I started off with asymmetry, everything worked well. When I got to circularity, I had the issue of NaNs and INFs. With some clever programming skills I tackled those issues and then comes the elephant in the room which was colour histograms. I remember we did colour histograms in Lab, but I wanted to create my own function.

```

function colour_Hist = colour_measure(Resized_Filtered_Lesions)
    imgs = readall(Resized_Filtered_Lesions);
    colour_Hist = zeros(numel(Resized_Filtered_Lesions),512);
    noBins = 8; % 8 bins (along each dimension)
    Hist = zeros(noBins, noBins, noBins); % empty histogram to start with
    for a = 1:numel(Resized_Filtered_Lesions.Files)
        % colour Histogram
        img_hsv = rgb2hsv(imgs{a});
        % Create a 3D histogram
        for i = 1:size(img_hsv, 1)
            for j = 1:size(img_hsv, 2)
                h = round(img_hsv(i, j, 1) * 7) + 1;
                s = round(img_hsv(i, j, 2) * 7) + 1;
                v = round(img_hsv(i, j, 3) * 7) + 1;
                Hist(h, s, v) = Hist(h, s, v) + 1;
            end
        end

        colour=reshape(Hist,[1 512]);
        colour_Hist(a,:)=colour;
    end

    colour_Hist = normalize(colour_Hist,'center','median','scale','mad');
    colour_Hist(isnan(colour_Hist)) = 0;
    colour_Hist(isinf(colour_Hist)) = 0;
    colour_Hist = abs(colour_Hist);
end

```

I opted for HSV rather than RGB reason being RGB was giving me very low accuracy like in sub 60s.

In this part things worked well until I hit the normalisation. NANs and INFs were stopping me from running the SVM. Plus, I got this error too at some point.

```
>> finalMain
Requested 350x16777216 (43.8GB) array exceeds maximum array size preference (13.9GB). This might cause MATLAB to become unresponsive.
```

Anyways, if I don't normalise the histogram I get a accuracy of around **71%** which I think is more realistic. But since I had to normalise everything and deal with NANs and INFs it somehow increased my accuracy.

For the circularity measure there was this issue I was facing. In the paper Classifying Mammographic Lesions Using Computerized Image Analysis. The formula for circularity was $C = \text{Perimeter}^2 / \text{Area}$. This formula was causing too many issues for me.

Well, due to my vast knowledge I was trying to run SVM with my labels in a matrix. It threw me an error then with the help of internet I figured it out and used a vector.

Future Workings:

Due to upcoming class test I was not able to go on with this project. If I was able, I would have added more functionalities like PCA etc. I know for a fact that PCA will improve the accuracy of the whole program. LPB would also be a great addition but again due to time constraint I wasn't able to do that.

In the end to summarise all this, the accuracy I got was 91.5% but I think if I don't normalise the histogram the accuracy of 71%.

Bibliography

fspecial Create predefined 2-D filter. (n.d.). Retrieved from MathWorks:

<https://uk.mathworks.com/help/images/ref/fspecial.html>

<https://uk.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html>

<https://uk.mathworks.com/company/newsletters/articles/image-overlay-using-transparency.html>

<https://uk.mathworks.com/help/images/ref/fspecial.html>

<https://uk.mathworks.com/help/stats/fitcsvm.html#d124e378495>

https://uk.mathworks.com/matlabcentral/fileexchange/50903-novel-method-for-determining-symmetry-of-skin-lesions-using-the-jaccard-index?s_tid=FX_rc3_behav

<https://uk.mathworks.com/help/vision/ug/perform-gamma-correction-image.html>

<https://uk.mathworks.com/help/images/ref/imgaussfilt.html>

<https://uk.mathworks.com/help/matlab/ref/isnan.html>

