Computer-Aided Diagnosis of Benign and Malignant Breast Tissue

Matthew Basso
Ryerson University

***Abstract*—Traditionally, pathologists have used visual perception to analyse histological slides for the diagnosis of various diseases. With technological advancements in the medical field new approaches have been introduced such as Computer-aided Diagnosis (CAD). By using image analysis and machine learning methods for biomedical images, significant diagnosis times can decrease, and higher consistency can be achieved. This paper describes an automated feature extraction and classification approach for breast tissue images. The features extracted from cropped Whole-Slide Images (WSI) include colour histogram features and texture features. Feature pruning was then performed and three different machine learning classifiers were evaluated based on their corresponding classification accuracies. The results show that the Random Forest classifier, classifies benign and malignant breast tumor tissues the best with an accuracy of 80% and AUC of about 0.80.**

***Index Terms*—Digital pathology, machine learning, AUC**

I. Introduction

Healthcare professionals and doctors receive extensive training in order to detect, diagnose, and prevent diseases. Although these professionals are highly qualified, it is important for them to have access to a wide range of technology to further improve diagnostic capabilities. Currently, pathology labs around the world are undergoing a transformation toward a fully digital workflow. Digital pathology is the digitizing of glass slides forming Whole-Slide Images (WSI) with extremely high resolution where they can be viewed, shared, managed, and analyzed using computer software [1]. Traditionally, pathologists observe tissues under a microscope and use human perception to gather insights from the tissue to come to a diagnosis [2]. Due to the variability and high workload of visual inspection, digital algorithms have been introduced that use image analysis techniques and machine learning to automate accurate classification of disease diagnosis [2]. This process of digitizing slides, is similar to that of the digitizing of radiological images, however the difference being that pathology images are tissue images while radiological images are images of gross anatomy [2]. With the advancements in WSI, novel software tools can be used to pull insights from pathology images, to quantify and understand disease etiology while also improving diagnostic accuracy with the help of bio-marker insights.

Image processing and feature extraction algorithms can be performed on medical images which can aid the physician with diagnostic information, helping the physician diagnose the condition with higher accuracy. These technologies are called Computer aided Diagnosis (CAD) systems and can help pathologists in interpreting WSI for classification between benign and malignant breast tumors [2].In Canada, breast cancer is the third most common cancer, and is the first most common cancer among women occurring in 83% of women 50 plus in age [3]. Due to breast cancer being most prevalent amongst women, pathology labs analyze a relatively large percentage of samples [2]. Analysis methods that are routinely used by pathologists such as; hormone receptor status by immunohistochemistry (IHC) and histological grade can be tedious and cause larger inter and intra observer variability [2]. CAD systems can potentially tackle these problems while still accurately diagnosing disease.

In this paper, the classification of benign and malignant breast tumors are investigated, which can be seen in Figure 1. Benign tumors are noncancerous growths in the body that do not proliferate (metastasize) to other parts of the body [1]. Meanwhile, malignant tumors are cancerous growths in the body that have uncontrollable cell division which proliferate and can invade into other places in the body [1]. The algorithm implemented uses handcrafted features and traditional supervised and unsupervised machine learning techniques to automatically differentiate between the two diseases. Different image information such as colour, and texture based features are gathered from the cropped WSI and are then used in a classifier to automatically classify the diseases. The evaluation of the algorithm is obtained using accuracy, specificity, sensitivity, and receiver operating characteristic (ROC) metrics.

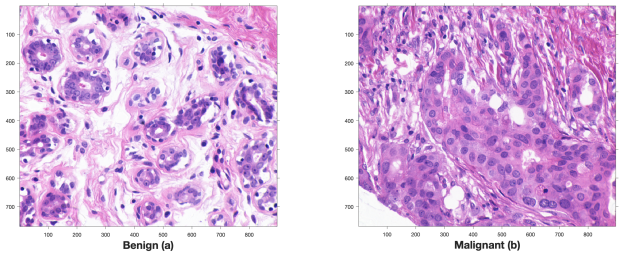

Fig. 1: Benign image (a). Malignant image (b).

II. Materials and Methods

A. Data

The dataset used in this paper is composed of breast benign and malignant digital pathology images. This dataset is composed of 58 images; 32 benign and 26 malignant images. The images in the dataset are in .TIFF format and are sized 896 x 768 pixels in RGB format.

B. Experimental Design

This paper discusses and investigates the classification of benign and malignant breast tumors using handcrafted features and traditional machine learning methods. Figure 2 shows a flowchart representation of the proposed system. In the first step, the images are loaded and vector noise reduction is performed on all the images. Image quality metrics were then gathered to quantify how the pre-processing technique performed. In the second step, feature extraction methods were obtained from all the images. Colour and texture features were extracted from the images and feature extraction validation metrics were gathered using statistical analysis. In the third step, the feature vector was normalized and feature reduction was performed to reduce the number of dimensions in the feature dataset. In the fourth step, a train-test split was performed to separate the dataset. Lastly, three different traditional machine learning classifiers were trained, tested, and evaluated using different metrics. To implement this algorithm, both MATLAB and Python were used.

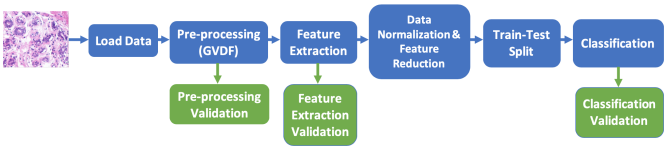

Fig. 2: Experimental design flowchart.

C. Pre-Processing

Pre-processing is an important step when analyzing images. Noise in colour images is a random process and can be caused by image acquisition artifacts, and poor slide preparation techniques [4]. If the noise in a colour image is not correlated across colour channels, traditional scalar noise reduction techniques used in grey-scale images can be performed on each individual channel [4]. However, noise doesn't represent itself in colour medical images in this way. It can be said that noise in medical images is correlated along colour channels and therefore scalar noise removal cannot be used as it will not effectively remove the overall noise in the image [4]. To pre-process a colour image, vector image processing is performed where each colour pixel is treated as a 3-dimensional vector and processing is done in the vector dimension [4]. The advantage to this is that this method does not disregard correlation between colour signal components and tends to be better than scalar methods [4]. In this algorithm, the generalized vector directional filter (GVDF) was used to suppress the noise in the colour images.

1) Generalized vector directional filter (GVDF): A colour image can be thought of as a 2-dimensional grid of 3-dimensional colour vectors R, G, and B [4]. In a uniform colour region the majority of the vectors in that region should be pointing in the same direction and have approximately the same magnitude [4]. If noise is present in that region, the vectors should be pointing in different directions and have different magnitudes [4]. The generalized vector directional filter (GVDF) is a vector image processing method that takes into account the angle and magnitude between colour RGB vectors. A sliding window over the image is performed and a filtered image is outputted. Firstly, a 3-dimensional neighbourhood is gathered and the sum-of-angles between each vector in the neighbourhood is stored [4]. These distances are then ordered from lowest to highest distance. Next, the top five vectors corresponding to the lowest distances are averaged. Lastly, this averaged vector is then the resultant filtered vector. For each neighbourhood over the image, a filtered vector is outputted and the resulting vectors form the filtered image. Figure 3 illustrates both the original and GVDF images. Qualitatively it can be observed that the original image has blurry and non-homogeneous regions while the GVDF image has sharp edges and is more homogeneous in colour. The main disadvantage to this method however is that it is computationally expensive. By reducing the noise in the image, more precise feature extraction metrics can be extracted from the images.

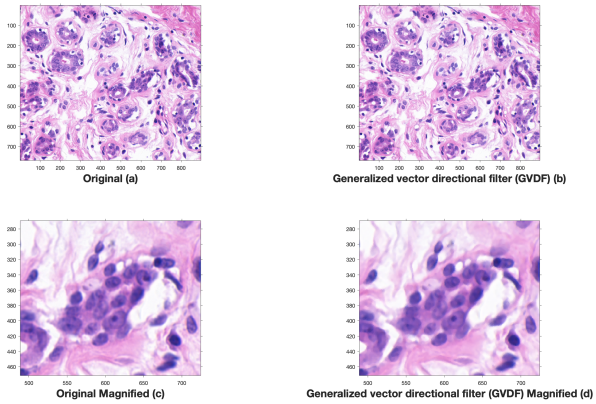

Fig. 3: Original image (a). GVDF image (b). Zoomed in original image (c). Zoomed in GVDF image (d).

2) GVDF Performance Metric: When looking at Figure 3, the qualitative performance of the GVDF filter can be assessed. Quantitative metrics can also be used to assess the performance of the filter such as entropy. Shannon entropy quantifies the pixel randomness in the image [5]. The average entropy was calculated before and average filtering. Before filtering the average entropy was 6.6563 while after GVDF the average entropy was 6.6107. This minute difference in entropy means that the noise was removed in the image while still preserving the image content.

D. Feature Extraction

Features can be used to gather insights from images. In this algorithm colour histogram and texture features such as GLCM, and vector LBP are extracted from the images forming a feature vector.

1) Histogram Features: Digital pathology medical images are most frequently taken in the RGB colour space. To examine the tissue in a laboratory, a pathologist performs staining to enhance the visualization of cells or cellular components under a microscope [2]. In this dataset, hematoxylin and eosin

(H&E) staining was performed. The hematoxylin stains cell nuclei blue, and eosin stains cytoplasm, connective tissue and other extracellular substances pink [2]. These colour structures can be analysed using histogram statistics. In this algorithm, the RGB image was converted to the HSV colour space and only the H channel was analyzed. The H channel represents the perception of colour and histogram statistics were performed. The features gathered from the histogram were; mean, variance, skewness, kurtosis, energy, and entropy. In Appendix A, the boxplots for these histogram features can be seen.

2) Gray-Level Co-Occurrence Matrix (GLCM) Texture Features: In texture analysis, Haralick et al. purposed Gray-Level Co-occurrence Matrices (GLCM) as a tool to extract textural features. GLCM is a statistical method of examining texture that considers the spatial relationship of pixels in the gray-level co-occurrence matrix [6]. The normalized GLCM can be represented where *P(l1,l2)* is the number of occurrences of grey level *l1*, and *l2* at a distance d and angle θ.

$$p\left(l_1, l_2, d, \theta\right) = \frac{P\left(l_1, l_2\right)}{\sum_{l_1=0}^{L-1} \sum_{l_2=0}^{L-1} P\left(l_1, l_2\right)} \tag{1}$$

The co-occurence matrix is then analyzed by extracting Haralick statistical measures which describe the texture. Some examples are contrast, correlation, energy, entropy, and homogeneity. When looking at the texture of benign and malignant breast tissue, there is a noticeable difference in texture. Benign breast tissues are more homogeneous and smooth in appearance, while malignant breast tissues are overall more nonhomogeneous in appearance. Since benign and malignant breast tissues have different textural characteristics, image analysis can be used to gather their textural characteristics so they can be utilized in a classifier. In this algorithm, GLCM was computed on the images at four different directions which are named, horizontal, vertical, left diagonal, and right diagonal respectively, that is, $0°$, $90°$, $45°$, and $135°$ shown in Figure 4. Four different GLCM's were outputted and the mean features were taken. The distance d taken for each direction was 1, with 8 grey levels. Lastly, the Haralick features used that best described homogeneity were: contrast, correlation, energy, entropy, and homogeneity. In Appendix A, the boxplots for these GLCM features can be seen.

Fig. 4: Four different directions GLCM is calculated [6].

3) Vector Local Binary Patterns (LBP) Texture Feature: Local binary patterns (LBP) is a texture descriptor feature used in image analysis. LBP has been successful in describing and classifying textures in general images and in medical images [7]. Since LBP is traditionally used for grey-scale images this algorithm uses a novel vector LBP method which extends it to color. The advantage of using a vector LBP method rather than a grey-scale LBP method is that it takes into account colour texture across R, G, and B channels [8]. Using similar principles as the GVDF, RGB colour channels are converted to a single value by finding the L2 norm or magnitude of the vector [8]. The formula of finding the magnitude of the RGB vector for each pixel is found in Equation (2). After this step, the normal LBP steps for grey-scale images can take place. The first step involves a neighbourhood of pixels which is defined by a radius, R, and a number of points, P to be segmented from the vector magnitude image [7]. In the second step, the center pixel in the neighbourhood is taken and the neighbouring pixels are thresholded [7]. If the center pixel is greater than the neighbouring pixel then the neighbouring pixel is converted to a 0, while if the center pixel is less or equal to the neighbouring pixel it is converted to a 1 [7]. This generates a binary string or a binary pattern [7]. A weighted mask is then multiplied by the binary string and the sum of the neighbourhood is taken [7]. This value is then outputted and a grey-scale LBP texture map image is created. Figure 5 shows the procedure for the novel vector LBP texture descriptor method. Once the LBP texture map image is produced, GLCM is performed on the image and Haralick features contrast, correlation, energy, entropy, and homogeneity were obtained. In Appendix A, the boxplots for these vector-LBP-GLCM features can be seen.

$$D_i = \sqrt{x_r^2 + x_g^2 + x_b^2} \tag{2}$$

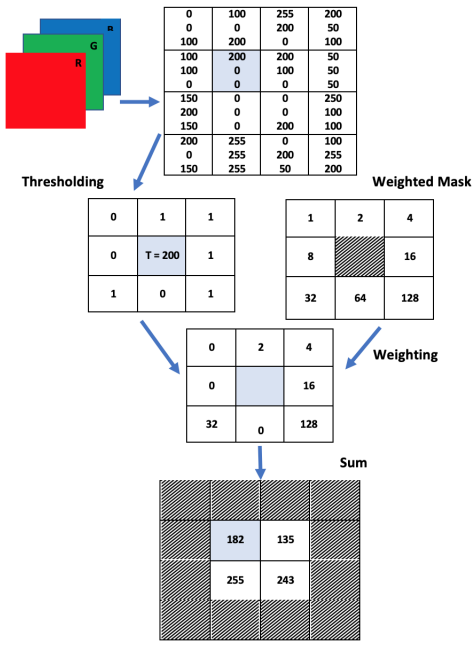

Fig. 5: Vector LBP procedure for colour images.

E. Normalization, Feature Reduction, and Train-test Split

Once the feature vector was obtained for all images, feature normalization was performed using z-score. Z-score subtracts

the data by its mean which centers the data, and then is divided by its standard deviation which performs variance scaling. This feature normalization is important as it puts all the data on the same scale where it can then be analyzed. After this, feature reduction was performed using principal component analysis (PCA) to reduce the number of features. PCA is used as a feature pruning technique to reduce the complexity of the feature dataset while extracting the top features that explain 97% of the variance in the dataset (Figure 6) [9]. Once PCA was performed, it reduced the feature dataset to six features; entropy-glcm, variance-hist, correlation-glcm, contrast-glcm, energy-hist, and kurtosis-hist. This reduced dataset was then split into 67% training and 33% testing before classification. This split was chosen as it gave the classifiers enough information for training and testing.

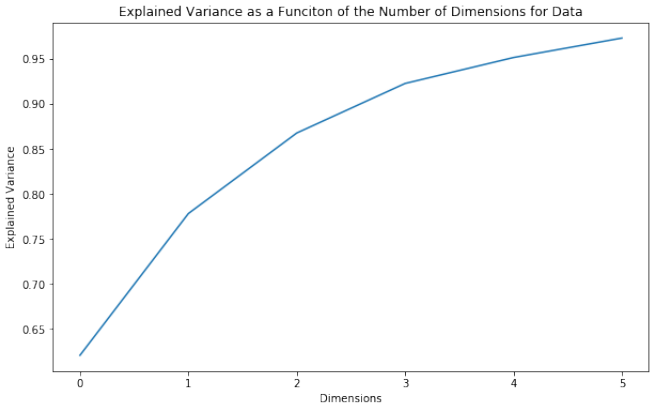

Fig. 6: Explained variance of PCA.

F. Classification

In this algorithm, three different traditional machine learning classifiers were used to classify between benign and malignant breast tumors. The classifiers analyzed were, k-Nearest Neighbour (k-NN), support vector machines (SVM), and Random Forest. All three of these classifiers are supervised machine learning algorithms which rely on labeled data to train the model. For each classifier a grid search was performed to find the optimal hyper-parameters for each classifier. Metrics for classification were then obtained and compared in the results.

1) k-Nearest Neighbour (k-NN): The first classifier used in this algorithm was the k-Nearest Neighbour (k-NN). This machine learning classifier is a supervised method that works by comparing the query instance's distance to the other training samples and selecting the k-NN [10]. It then takes the majority of these k-neighbour classes to be the prediction of the query instances. This model, ultimately states that similar class features exist in close proximity [10]. The parameter for the number of neighbours is very important as it can dictate whether the model is under or overfitting. To obtain proper hyper-parameters for this model, a grid search was performed and the following parameters were obtained: leaf-size = 1, n-neighbours = 3, and p = 1.

2) Support Vector Machines (SVM): The second classifier used in this algorithm was support vector machines (SVM). This machine learning classifier is a supervised method that works by optimally separating data by drawing a line between the two classes which is then used to predict future data [10]. This classifier is fast and performs well on a limited amount of data [10]. SVM's models can be linear to classify data or may need some manipulation using the kernel trick as the data might not be linearly separable [10]. Besides the linear kernel, some of the non-linear kernels include the radial basis function (RBF) and the polynomial kernel [10]. Choosing which kernel to use is important when developing the SVM classifier. The C parameter is also an important parameter as it is known as the penalty parameter of the error term [10]. This means that it controls the trade-off between the smooth decision boundary and classifying the training points correctly [10]. The last parameter is called the Gamma and it defines how far the influence of a single training example reaches [10]. These three parameters were fine tuned using grid search and the optimal parameters obtained for the SVM model were: kernel = rbf, C = 1, and gamma = 0.1.

3) Random Forest: The last classifier used in this algorithm was Random Forest. This machine learning classifier is a supervised method that is an ensemble of Decision Trees, generally trained via the bagging method [11]. Each individual tree in the Random Forest outputs a class prediction and the class with the most votes becomes the model's prediction [11]. The Random Forest model, introduces extra randomness when growing trees as it searches for the best feature among a random subset of features when splitting a node [11]. This ultimately results in a greater tree diversity yielding an overall better model [11]. Bootstrap is an important parameter when building this model [11]. If bootstrap is selected then the Random Forest is trained on different random subsets of the training data [11]. When using grid search optimal parameters for the Random Forest were found: bootstrap = True, max-depth = 100, max-features = sqrt, min-samples-leaf = 4, min-samples-spilt = 5, and n-estimators = 50.

G. Classification Validation

When observing each classifiers output, performance metrics can gather information on which classifier performed the best. In this dataset, the images were labeled as benign or malignant by an expert. These labels were then compared to those outputted by the classifiers. Different metrics such as accuracy, sensitivity, specificity and receiver operating characteristic (ROC) were obtained.

1) Accuracy: The accuracy of a model is a fraction of the number of correct predictions over the total number of predictions [12]. The accuracy formula can be seen in Equation (3).

$$ACC = \frac{TP + TN}{TP + FP + TP + FN} \tag{3}$$

2) Sensitivity: The sensitivity of a model is the proportion of observed positives that were predicted to be positive [12]. The sensitivity formula can be seen in Equation (4).

$$Sen = \frac{TP}{TP + FN} \tag{4}$$

3) Specificity: The specificity of a model is the proportion of observed negatives that were predicted to be negatives [12]. The specificity formula can be seen in Equation (5).

$$Spec = \frac{TN}{TN + FP} \tag{5}$$

III. Results

A. Feature Extraction Validation

Based on the 16 features gathered a *t*-test was performed to determine if there is a significant difference between the means of two groups, benign versus malignant breast tumors. If the null hypothesis in the statistical test is rejected then the means of the two groups are different. This statistical test gives some inference of which features are good in differentiating the two diseases. Table I, illustrates the *t*-test performed on all the features gathered with an α = 0.05. If the hypothesis test was 1 then the null hypothesis was rejected and the means were found to be different. p-values were also obtained which show how different the two diseases actually are for each feature.

TABLE I: T-test statistics for feature extraction.

Feature	Hypothesis Test	p-value
Mean Hist	1	9.7093e-06
Variance Hist	0	0.39307
Skewness Hist	1	1.6998e-06
Kurtosis Hist	1	0.013548
Energy Hist	1	0.00040799
Entropy Hist	1	0.00019081
Contrast GLCM	0	0.081131
Correlation GLCM	1	0.015246
Energy GLCM	1	4.8941e-05
Entropy GLCM	1	0.0001332
Homogeneity GLCM	1	0.00071357
Contrast LBP-GLCM	1	0.0084119
Correlation LBP-GLCM	1	0.023843
Energy LBP-GLCM	0	0.29594
Entropy LBP-GLCM	0	0.15336
Homogeneity LBP-GLCM	0	0.089332

B. Classification

Based on the evaluation metrics for each classifier, it can be seen that the Random Forest classifier performed the best. The accuracy using 5-fold cross validation for training and testing were as follows, 84.2% and 80%. The model also had a sensitivity and specificity of 0.833% and 0.79%. Lastly, the area under the curve (AUC) ROC metric, is one of the most important evaluation metrics for checking any classification model's performance. It tells how much a model is capable of distinguishing between classes. A higher AUC means that the model is better in distinguishing between diseases. When comparing the AUC between all the classifiers, the Random Forest classifier had the highest of 0.80 which means that it did the best job in classifying benign versus malignant breast tumors. It can be said that the Random Forest model had good generalization as the results are repeatable through cross validation. Also, the training and testing scores obtained were similar, indicating that the model was not overfitting. Looking at the f1-score, it is seen that the model is able to classify benign cases better than malignant. One reason for this is due to the class imbalance since there are more benign cases than malignant.

TABLE II: Comparing different classifier metrics.

Classifiers	Train Accuracy	Test Accuracy	Test Sensitivity	Test Specificity
k-NN	0.815789	0.650000	0.833333	0.571429
SVM	0.815789	0.700000	0.833333	0.642857
Random Forest	**0.842105**	**0.800000**	**0.833333**	**0.785714**

TABLE III: Classification report showing the main classification metrics.

	Precision	Recall	f1-score	Support
Benign	0.92	0.79	0.85	14
Malignant	0.62	0.83	0.71	6
Accuracy			0.80	20
Macro Accuracy	0.77	0.81	0.78	20
Weighted Average	0.83	0.80	0.81	20

TABLE IV: Confusion matrix for benign or malignant breast tumors.

	Predicted Benign	Predicted Malignant
Actual Benign	11	3
Actual Malignant	1	5

Fig. 7: Learning curve for Random Forest model.

IV. Discussion & Conclusion

In this paper, a complete workflow for benign versus malignant breast tissue tumor classification was designed and developed. The images from the dataset were first pre-processed using the GVDF which is a filter that removes noise

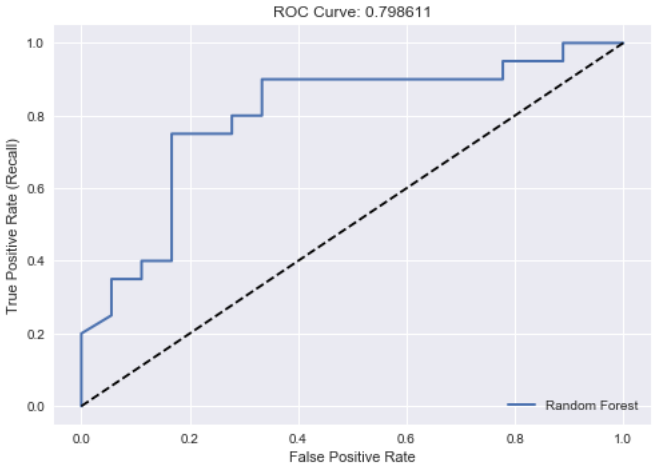

Fig. 8: ROC curve for Random Forest model.

from colour images using vector processing. Next, 16 different features were extracted from each image which include colour histogram and texture features. Once these features were obtained, feature normalization and feature pruning were performed using PCA. The dataset was then split into training and testing for classification. Next, three different classifiers were examined and classification metrics were obtained. For each model, a grid search was performed to obtain the best performance for each model. It was observed that out of all the models, the Random Forest classifier had the highest accuracy of 80% and AUC of about 0.80. Good precision and recall were obtained too. The reported results suggest that the purposed CAD algorithm is highly reliable and robust for the classification of benign and malignant breast tissue tumors. However, there is still a high miss-classification of malignant images classified as benign. The reason for this is due to the class imbalance and the small dataset used. If a larger dataset was obtained and had an equal representation of both benign and malignant breast tissue tumor images, the metrics obtained would be more accurate and would give a better understanding of the model.

In future work, an investigation of other texture feature methods would be analyzed such as wavelet features and Gabor filters. These texture features are widely used for texture classification and perform localized and orientated frequency analysis on the images [13]. Also, other feature pruning methods can be assessed such as Minimum Redundancy Maximum Relevance (mRMR) which looks at the correlation between features. Other classification methods such as Convolutional Neural Networks (CNN's) which are a deep learning algorithms can be assessed. Finally, the designed algorithm can be integrated into a complete CAD tool for pathologists to use.

References

[1] L. Pantanowitz, P. N. Valenstein, A. J. Evans, K. J. Kaplan, J. D. Pfeifer, D. C. Wilbur, L. C. Collins, and T. J. Colgan, "Review of the current state of whole slide imaging in pathology," *Journal of pathology informatics*, vol. 2, 2011.

[2] M. Veta, J. P. Pluim, P. J. Van Diest, and M. A. Viergever, "Breast cancer histopathology image analysis: A review," *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 5, pp. 1400–1411, 2014.

[3] P. H. A. of Canada, "Government of canada," Dec 2019. [Online]. Available: https://www.canada.ca/en/public-health/services/chronic-diseases/cancer/breast-cancer.html

[4] A. Khademi, "Lecture 6 notes," Feb 2020.

[5] ——, "Lecture 4 notes," Feb 2020.

[6] R. M. Haralick and L. G. Shapiro, *Computer and robot vision*. Addison-wesley Reading, 1992, vol. 1.

[7] S. Morales, K. Engan, V. Naranjo, and A. Colomer, "Retinal disease screening through local binary patterns," *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 184–192, 2015.

[8] A. Porebski, N. Vandenbroucke, and L. Macaire, "Haralick feature extraction from lbp images for color texture classification," in *2008 First Workshops on Image Processing Theory, Tools and Applications*. IEEE, 2008, pp. 1–8.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] W.-M. Lee, *Python Machine Learning*. John Wiley & Sons, 2019.

[11] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[12] R. Parikh, A. Mathai, S. Parikh, G. C. Sekhar, and R. Thomas, "Understanding and using sensitivity, specificity and predictive values," *Indian journal of ophthalmology*, vol. 56, no. 1, p. 45, 2008.

[13] E. Cernadas, M. Fernández-Delgado, E. González-Rufino, and P. Carrión, "Influence of normalization and color space to color texture classification," *Pattern Recognition*, vol. 61, pp. 120–138, 2017.

Appendix A
Feature Boxplots

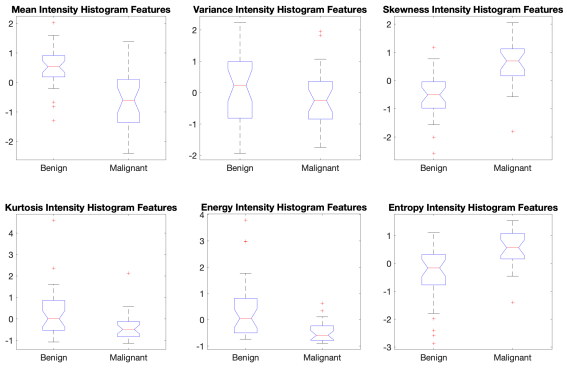

Fig. 9: Histogram features.

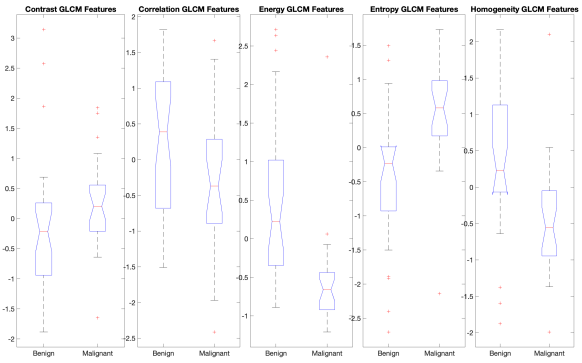

Fig. 10: GLCM features.

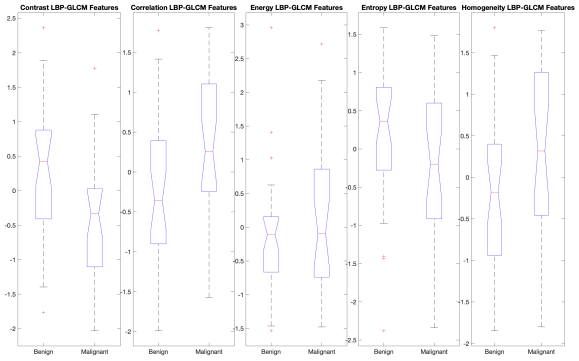

Fig. 11: Vector LBP-GLCM features.

Code Appendix:

Main:

```
% BE8105 - Advanced Medical Image Analysis

% Assignment 2: Computer-Aided Diagnosis for Digital Pathology Images
% Matthew Basso, 500686499

%% Clear, close all, clc

clear;
close all;
clc;

%% Load Dataset

% Adding all folders in assignment 2 to path
addpath(genpath('/Users/matthewbasso/Desktop/MASc Courses/BE8105- Advanced Medical Image Analysis/Assignments/Assignment 2'));

% Dataset directory
Img_dir_images = '/Users/matthewbasso/Desktop/MASc Courses/BE8105- Advanced Medical Image Analysis/Assignments/Assignment 2/BE8105 - Adv. Medical Image Analysis - W2020 - 362020 - 1149 AM/Breast Cancer Images';

[P,n] = LoadFiles(Img_dir_images);

cd('/Users/matthewbasso/Desktop/MASc Courses/BE8105- Advanced Medical Image Analysis/Assignments/Assignment 2/Tools');

%% Pre-processing

disp('...Pre-processing');

for i = 1:length(P)

    disp(['...GVDF Image:', num2str(i)]);
    [P(i).GVDF] = GVDF(P(i).Image,[3,3]);

end

% Pre-processing Metrics
[Noise_metrics] = colour_noise_metrics(P);

%% Feature Extraction

[features] = feature_extraction(P);

%% Features to Array

features_array = struct('Histogram', {features.Histogram}, 'GLCM', {features.GLCM}, 'LBP_features', {features.LBP_features});
features_array = struct2table(features_array,'AsArray',true);
features_array = table2array(features_array);

%% Save as CSV

label = [P.Label]';
formatOut = 'mm_dd_yy';
date = datestr(now,formatOut);

csvwrite(['features_',date,'.csv'], features_array);
csvwrite(['label_',date,'.csv'], label);
```

```
%%

X = categorical({'Original' 'GVDF'});
X = reordercats(X,{'Original' 'GVDF'});

x = mean([Noise_metrics.Entropy_Original]);
y = mean([Noise_metrics.Entropy_Filtered]);

figure
b = bar(X,[x y]);
title('Average Entropy of Original vs GVDF','fontsize',20,'fontweight','bold');
ylabel('Entropy');
set(gca,'FontSize',20)


xtips1 = b(1).XEndPoints;
ytips1 = b(1).YEndPoints;
labels1 = string(b(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom','fontsize',20,'fontweight','bold')

%% Loading features

features = csvread('features_03_19_20.csv');

%% Plotting Boxplots of features

% Z-score normalization
features_norm = zscore(features);

Label1 = find(~[P.Label]');
Label2 = find([P.Label]');

% Boxplot
figure('DefaultAxesFontSize',18);

label = {'Benign','Malignant'};
grp = [zeros(1,length(Label1)),ones(1,length(Label2))];

subplot(2,3,1)
C = [features_norm(Label1,1) ; features_norm(Label2,1)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Mean Intensity Histogram Features');

subplot(2,3,2)
C = [features_norm(Label1,2) ; features_norm(Label2,2)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Variance Intensity Histogram Features');

subplot(2,3,3)
C = [features_norm(Label1,3) ;features_norm(Label2,3)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Skewness Intensity Histogram Features');

subplot(2,3,4)
C = [features_norm(Label1,4) ;features_norm(Label2,4)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Kurtosis Intensity Histogram Features');

subplot(2,3,5)
```

```
C = [features_norm(Label1,5) ;features_norm(Label2,5)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Energy Intensity Histogram Features');

subplot(2,3,6)
C = [features_norm(Label1,6) ;features_norm(Label2,6)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Entropy Intensity Histogram Features');

% Boxplot GLCM
figure('DefaultAxesFontSize',16);

subplot(1,5,1)
C = [features_norm(Label1,7) ; features_norm(Label2,7)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Contrast GLCM Features');

subplot(1,5,2)
C = [features_norm(Label1,8) ; features_norm(Label2,8)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Correlation GLCM Features');

subplot(1,5,3)
C = [features_norm(Label1,9) ;features_norm(Label2,9)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Energy GLCM Features');

subplot(1,5,4)
C = [features_norm(Label1,10) ;features_norm(Label2,10)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Entropy GLCM Features');

subplot(1,5,5)
C = [features_norm(Label1,11) ;features_norm(Label2,11)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Homogeneity GLCM Features');

% Boxplot LBP
figure('DefaultAxesFontSize',14);

subplot(1,5,1)
C = [features_norm(Label1,12) ; features_norm(Label2,12)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Contrast LBP-GLCM Features');

subplot(1,5,2)
C = [features_norm(Label1,13) ; features_norm(Label2,13)]';
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Correlation LBP-GLCM Features');

subplot(1,5,3)
C = [features_norm(Label1,14) ;features_norm(Label2,14)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Energy LBP-GLCM Features');

subplot(1,5,4)
C = [features_norm(Label1,15) ;features_norm(Label2,15)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Entropy LBP-GLCM Features');

subplot(1,5,5)
```

```
C = [features_norm(Label1,16) ;features_norm(Label2,16)];
boxplot(C,grp,'Notch','on','Labels',label,'Whisker',1);
title('Homogeneity LBP-GLCM Features');


%% T-Test
featnames =
["Mean_hist","Variance_hist","Skewness_hist","Kurtosis_hist","Energy_hist","Entropy_hist","Contrast_glcm","Correlation_glcm","Energy_glcm
","Entropy_glcm","Homogeneity_glcm","Contrast_lbp","Correlation_lbp","Energy_lbp","Entropy_lbp","Homogeneity_lbp"];

% Two Sample t-test
[t_test] = two_sample_ttest(features_norm,featnames,Label1,Label2,1);

function [Noise_metrics] = colour_noise_metrics(P)

% colour_noise_metrics
%   This function calculates the perfromance of removing noise in colour
%   images

%% Struct with noise metrics

disp('... Image Noise Metrics');

Noise_metrics(1:length(P)) = struct('Name','','Image',[],'Filtered',[],'Entropy_Original',[],'Entropy_Filtered',[],'Sharpness',[]);

for i = 1:length(P)

   Noise_metrics(i).Name = P(i).Name;
   Noise_metrics(i).Image = P(i).Image;
   Noise_metrics(i).Filtered = P(i).GVDF;

end

%% Entropy

for i = 1:length(P)

   q = Noise_metrics(i).Filtered;
   f = Noise_metrics(i).Image;

   Noise_metrics(i).Entropy_Original = (entropy(f(:,:,1)) + entropy(f(:,:,2)) + entropy(f(:,:,3)))./3;
   Noise_metrics(i).Entropy_Filtered = (entropy(q(:,:,1)) + entropy(q(:,:,2)) + entropy(q(:,:,3)))./3;

end

%% Image Sharpness

for i = 1:length(P)

   q = rgb2gray(Noise_metrics(i).Filtered);
   f = rgb2gray(Noise_metrics(i).Image);

   Noise_metrics(i).Sharpness = sharpness_metric(f,q);

end

end


function [avg_vector] = distance_sum_angle(vectors)
%UNTITLED2 Summary of this function goes here
```

```
%   Detailed explanation goes here


vectors = double(vectors);

for i = 1:length(vectors)

    ref = vectors(i,:);

    idx = ones(1,length(vectors))';
    idx(i,:) = 0;
    idx = idx.*(1:length(vectors))';
    idx(idx==0) = [];

    x = vectors(idx,:);

    for k = 1:length(x)

        vect = x(k,:);

        num = ref(1)*vect(1) + ref(2)*vect(2) + ref(3)*vect(3);
        den = sqrt((ref(1)).^2 + (ref(2)).^2 + (ref(3)).^2) .* sqrt((vect(1)).^2 + (vect(2)).^2 + (vect(3)).^2) ;

        d(k,:,i) = acos(num./den);

    end

    d_sum(i,:) = real(sum(d(:,:,i)));

end

[~,I] = sort(d_sum);

m = floor(length(vectors)/2)+1;

avg_vector = mean(vectors(I(1:m),:),1);

end


function [out] = entropy_color(I,win_size)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

ksize = floor(win_size(1)./2);
I_pad = double(padarray(I,[ksize ksize],'symmetric','both'));

m = win_size(1);
n = win_size(2);

C = round(win_size(1)./2);
F = round((n.*m)./2);

row_max = size(I_pad,1)-m+1;
col_max = size(I_pad,2)-n+1;

out = zeros([row_max, col_max]);

for i = 1:row_max
    for j = 1:col_max
```

```
        Temp = I_pad(i:i+m-1,j:j+n-1,:);
        Temp = reshape(Temp,[],3);

        for k = 1:length(Temp)

            Temp_d(k) = sqrt(Temp(k,1).^2 + Temp(k,2).^2 + Temp(k,3).^2);

        end

        Temp_d = reshape(Temp_d,m,n);
        out(i,j) = entropy(Temp_d/max(Temp_d,[],'all'));

    end
end

out = uint8(out);

end


function [features] = feature_extraction(P)
%feature_extraction
%   In this function, different feature extraction metrics will be gathered

tic;

disp('...Feature Extraction');

%% Feature Struct

features(1:length(P)) = struct('Name','','Image',[]);

for i = 1:length(P)

    features(i).Name = P(i).Name;
    features(i).Image = P(i).GVDF;

end

%% Histogram Features

disp('...Histogram Based Features');
for i = 1:length(P)

    [features(i).Histogram] = hist_features(rgb2gray(features(i).Image));

end

%% GLCM

disp('...GLCM Features');
for i = 1:length(P)

    I = rgb2gray(features(i).Image);
    [features(i).GLCM] = table2array(struct2table(glcm(I,[0 1; -1 1; -1 0; -1 -1],[], 9,false)));

end

%% Wavelet Transform

% disp('...Wavelet Transform Features');
```

```
%
% level = 3;
% wname = 'sym2';
% show = false;
%
% for i = 1:length(P)
%
%     % Image pre-processing
%     I = rgb2gray(features(i).Image);
%
%     % 2-D Wavelet Decomposition
%     [c,s] = wavedec2(I,level,wname);
%
%     for k = 1:level
%
%         LL{k} = appcoef2(c,s,wname,k); % approx
%         [LH{k}, HL{k}, HH{k}] = detcoef2('all',c,s,k); % detail
%
%     end
%
%     [features(i).wavelet_features] = WaveletAnalysis(c,s,LL,LH,HL,HH,level,wname,show,gray,'tree');
%
% end

%% LBP

disp('...LBP Features');
for i = 1:length(P)

    [features(i).LBP] = lbp_color(P(i).Image,[3,3]);
    [features(i).LBP_features] = table2array(struct2table(glcm(features(i).LBP,[0 1; -1 1; -1 0; -1 -1],[], 9,false)));

end

toc;
end


function [GLCM] = glcm(I,offset,graylim, numlevels,symmetric)

%% GLCM Struct

GLCM(1) = struct('Contrast',[],'Correlation',[],'Energy',[],'Entropy',[], 'Homogeneity',[]);

%% GLCM

[glcm,~] = graycomatrix(I, 'Offset',offset, 'GrayLimits', graylim, 'NumLevels',numlevels, 'Symmetric', symmetric);
glcm_stats = graycoprops(glcm,'all');

%% Mean Compressed Haralick texture Features

GLCM.Contrast = mean(glcm_stats.Contrast);
GLCM.Correlation = mean(glcm_stats.Correlation);
GLCM.Energy = mean(glcm_stats.Energy);
GLCM.Entropy = mean(glcm_stats.Entropy);
GLCM.Homogeneity = mean(glcm_stats.Homogeneity);

end


function [GLCMS,SI] = glcm3D(I, NL, GL, Offset)
```

```
% Matthew Basso, IAMLAB, Ryerson University
%glcm3D

%   GLCMS = glcm3D(I) analyzes pairs of horizontally adjacent pixels
%   in a scaled version of I.  If I is a binary image, it is scaled to 2
%   levels. If I is an intensity image, it is scaled to 8 levels. In this
%   case, there are 8 x 8 = 64 possible ordered combinations of values for
%   each pixel pair. GRAYCOMATRIX accumulates the total occurrence of each
%   such combination, producing a 8-by-8 output array, GLCMS. The row and
%   column subscripts in GLCMS correspond respectively to the first and
%   second (scaled) pixel-pair values.
%
%   GLCMS = GRAYCOMATRIX(I,PARAM1,VALUE1,PARAM2,VALUE2,...) returns one or
%   more gray-level co-occurrence matrices, depending on the values of the
%   optional parameter/value pairs. Parameter names can be abbreviated, and
%   case does not matter.

% Scale I so that it contains integers between 1 and NL.
if GL(2) == GL(1)
  SI = ones(size(I));
else
    slope = NL / (GL(2) - GL(1));
    intercept = 1 - (slope*(GL(1)));
    SI = floor(imlincomb(slope,I,intercept,'double'));
end

% Clip values if user had a value that is outside of the range, e.g.,
% double image = [0 .5 2;0 1 1]; 2 is outside of [0,1]. The order of the
% following lines matters in the event that NL = 0.
SI(SI > NL) = NL;
SI(SI < 1) = 1;

numOffsets = size(Offset,1);

if NL ~= 0

  % Create vectors of row and column subscripts for every pixel and its
  % neighbor.
  s = size(I);
  [r,c] = meshgrid(1:s(1),1:s(2));
  r = r(:);
  c = c(:);

  % Compute GLCMS

  GLCMS = zeros(NL,NL,numOffsets);

  for k = 1 : numOffsets

    GLCMS(:,:,k) = computeGLCM3D(r,c,Offset(k,:),SI,NL);

  end

end


function stats = graycoprops(varargin)
%GRAYCOPROPS Properties of gray-level co-occurrence matrix.
%   STATS = GRAYCOPROPS(GLCM,PROPERTIES) normalizes the gray-level
%   co-occurrence matrix (GLCM) so that the sum of its elements is one. Each
```

```
%   element in the normalized GLCM, (r,c), is the joint probability occurrence
%   of pixel pairs with a defined spatial relationship having gray level
%   values r and c in the image. GRAYCOPROPS uses the normalized GLCM to
%   calculate PROPERTIES.
%
%   GLCM can be an m x n x p array of valid gray-level co-occurrence
%   matrices. Each gray-level co-occurrence matrix is normalized so that its
%   sum is one.
%
%   PROPERTIES can be a comma-separated list of strings or char vectors, a
%   cell array containing strings or char vectors, the string "all", char
%   vector 'all', or a space separated string or char vector. They can be
%   abbreviated, and case does not matter.
%
%   Properties include:
%
%   'Contrast'      the intensity contrast between a pixel and its neighbor
%                   over the whole image. Range = [0 (size(GLCM,1)-1)^2].
%                   Contrast is 0 for a constant image.
%
%   'Correlation'   statistical measure of how correlated a pixel is to its
%                   neighbor over the whole image. Range = [-1 1].
%                   Correlation is 1 or -1 for a perfectly positively or
%                   negatively correlated image. Correlation is NaN for a
%                   constant image.
%
%   'Energy'        summation of squared elements in the GLCM. Range = [0 1].
%                   Energy is 1 for a constant image.
%
%   'Homogeneity'   closeness of the distribution of elements in the GLCM to
%                   the GLCM diagonal. Range = [0 1]. Homogeneity is 1 for
%                   a diagonal GLCM.
%
%   If PROPERTIES is the string "all" or char vector 'all', then all of the
%   above properties are calculated. This is the default behavior. Please
%   refer to the Documentation for more information on these properties.
%
%   STATS is a structure with fields that are specified by PROPERTIES. Each
%   field contains a 1 x p array, where p is the number of gray-level
%   co-occurrence matrices in GLCM. For example, if GLCM is an 8 x 8 x 3 array
%   and PROPERTIES is 'Energy', then STATS is a structure containing the
%   field 'Energy'. This field contains a 1 x 3 array.
%
%   Notes
%   -----
%   Energy is also known as uniformity, uniformity of energy, and angular second
%   moment.
%
%   Contrast is also known as variance and inertia.
%
%   Class Support
%   -------------
%   GLCM can be logical or numeric, and it must contain real, non-negative, finite,
%   integers. STATS is a structure.
%
%   Examples
%   --------
%   GLCM = [0 1 2 3;1 1 2 3;1 0 2 0;0 0 0 3];
%   stats = graycoprops(GLCM)
%
%   I = imread('circuit.tif');
```

```
%   GLCM2 = graycomatrix(I,'Offset',[2 0;0 2]);
%   stats = graycoprops(GLCM2,{'contrast','homogeneity'})
%
%   See also GRAYCOMATRIX.

% Copyright 2003-2016 The MathWorks, Inc.

allStats = {'Contrast','Correlation','Energy','Entropy', 'Homogeneity'};

[glcm, requestedStats] = ParseInputs(allStats, varargin{:});

% Initialize output stats structure.
numStats = length(requestedStats);
numGLCM = size(glcm,3);
empties = repmat({zeros(1,numGLCM)},[numStats 1]);
stats = cell2struct(empties,requestedStats,1);

for p = 1 : numGLCM

 if numGLCM ~= 1 %N-D indexing not allowed for sparse.
   tGLCM = normalizeGLCM(glcm(:,:,p));
 else
   tGLCM = normalizeGLCM(glcm);
 end

 % Get row and column subscripts of GLCM.  These subscripts correspond to the
 % pixel values in the GLCM.
 s = size(tGLCM);
 [c,r] = meshgrid(1:s(1),1:s(2));
 r = r(:);
 c = c(:);

 % Calculate fields of output stats structure.
 for k = 1:numStats
   name = requestedStats{k};
   switch name
    case 'Contrast'
     stats.(name)(p) = calculateContrast(tGLCM,r,c);

    case 'Correlation'
     stats.(name)(p) = calculateCorrelation(tGLCM,r,c);

    case 'Energy'
     stats.(name)(p) = calculateEnergy(tGLCM);

    case 'Entropy'
     stats.(name)(p) = calculateEntropy(tGLCM);

    case 'Homogeneity'
     stats.(name)(p) = calculateHomogeneity(tGLCM,r,c);
   end
 end

end


%--------------------------------------------------------------------------
function glcm = normalizeGLCM(glcm)

% Normalize glcm so that sum(glcm(:)) is one.
if any(glcm(:))
```

```
  glcm = glcm ./ sum(glcm(:));
end

%-------------------------------------------------------------------------
function C = calculateContrast(glcm,r,c)
% Reference: Haralick RM, Shapiro LG. Computer and Robot Vision: Vol. 1,
% Addison-Wesley, 1992, p. 460.
k = 2;
l = 1;
term1 = abs(r - c).^k;
term2 = glcm.^l;

term = term1 .* term2(:);
C = sum(term);

%-------------------------------------------------------------------------
function Corr = calculateCorrelation(glcm,r,c)
% References:
% Haralick RM, Shapiro LG. Computer and Robot Vision: Vol. 1, Addison-Wesley,
% 1992, p. 460.
% Bevk M, Kononenko I. A Statistical Approach to Texture Description of Medical
% Images: A Preliminary Study., The Nineteenth International Conference of
% Machine Learning, Sydney, 2002.
% http://www.cse.unsw.edu.au/~icml2002/workshops/MLCV02/MLCV02-Bevk.pdf, p.3.

% Correlation is defined as the covariance(r,c) / S(r)*S(c) where S is the
% standard deviation.

% Calculate the mean and standard deviation of a pixel value in the row
% direction direction. e.g., for glcm = [0 0;1 0] mr is 2 and Sr is 0.
mr = meanIndex(r,glcm);
Sr = stdIndex(r,glcm,mr);

% mean and standard deviation of pixel value in the column direction, e.g.,
% for glcm = [0 0;1 0] mc is 1 and Sc is 0.
mc = meanIndex(c,glcm);
Sc = stdIndex(c,glcm,mc);

term1 = (r - mr) .* (c - mc) .* glcm(:);
term2 = sum(term1);

Corr = term2 / (Sr * Sc);

%-------------------------------------------------------------------------
function S = stdIndex(index,glcm,m)

term1 = (index - m).^2 .* glcm(:);
S = sqrt(sum(term1));

%-------------------------------------------------------------------------
function M = meanIndex(index,glcm)

M = index .* glcm(:);
M = sum(M);

%-------------------------------------------------------------------------
function E = calculateEnergy(glcm)
% Reference: Haralick RM, Shapiro LG. Computer and Robot Vision: Vol. 1,
% Addison-Wesley, 1992, p. 460.

foo = glcm.^2;
```

```
E = sum(foo(:));

%--------------------------------------------------------------------------
function En = calculateEntropy(glcm)
% Reference: Haralick RM, Shapiro LG. Computer and Robot Vision: Vol. 1,
% Addison-Wesley, 1992, p. 460.

% remove zero entries in glcm

glcm(glcm==0) = [];

En = -sum(glcm(:).*log2(glcm(:)));

%--------------------------------------------------------------------------
function H = calculateHomogeneity(glcm,r,c)
% Reference: Haralick RM, Shapiro LG. Computer and Robot Vision: Vol. 1,
% Addison-Wesley, 1992, p. 460.

term1 = (1 + abs(r - c));
term = glcm(:) ./ term1;
H = sum(term);

%--------------------------------------------------------------------------
function [glcm,reqStats] = ParseInputs(allstats,varargin)

numstats = length(allstats);
narginchk(1,numstats+1);

reqStats = '';
glcm = varargin{1};

% The 'nonnan' and 'finite' attributes are not added to validateattributes because the
% 'integer' attribute takes care of these requirements.
validateattributes(glcm,{'logical','numeric'},{'real','nonnegative','integer'}, ...
              mfilename,'GLCM',1);

if ndims(glcm) > 3
  error(message('images:graycoprops:invalidSizeForGLCM'))
end

% Cast GLCM to double to avoid truncation by data type. Note that GLCM is not an
% image.
if ~isa(glcm,'double')
  glcm = double(glcm);
end

list = varargin(2:end);

if isempty(list)
  % GRAYCOPROPS(GLCM) or GRAYCOPROPS(GLCM,PROPERTIES) where PROPERTIES is empty.
  reqStats = allstats;
else
  if iscell(list{1}) || numel(list) == 1
    % GRAYCOPROPS(GLCM,{...})
    list = list{1};
  end

  if ischar(list)
    %GRAYCOPROPS(GLCM,SPACE-SEPARATED STRING)
    C = textscan(list, '%s');
    list = C{1};
```

```
 end

 anyprop = allstats;
 anyprop{end+1} = 'all';

 for k = 1 : length(list)
  match = validatestring(list{k}, anyprop, mfilename, 'PROPERTIES', k+1);
  if strcmp(match,'all')
   reqStats = allstats;
   break;
  end
  reqStats{k} = match;
 end

end

% Make sure that reqStats are in alphabetical order.
reqStats = sort(reqStats);

if isempty(reqStats)
 error(message('images:graycoprops:internalError'))
end

function [out] = GVDF(I,win_size)

% Generalized vector directional filter (GVDF)
%   This filter removes noise in colour images using vector image
%   processing
%   This filter uses vector angles to filter the image

%% GVDF sliding window

tic;

ksize = floor(win_size(1)./2);
I_pad = padarray(I,[ksize ksize],0,'both');

m = win_size(1);
n = win_size(2);

row_max = size(I_pad,1)-m+1;
col_max = size(I_pad,2)-n+1;

out = zeros([row_max, col_max, 3]);

for i = 1:row_max
   for j = 1:col_max

       Temp = I_pad(i:i+m-1,j:j+n-1,:);
       vectors = reshape(Temp,[],3);

       [avg_vector] = distance_sum_angle(vectors);
       avg_vector = reshape(avg_vector,1,1,3);
       out(i,j,:) = avg_vector;

   end
end

out = uint8(out);

toc;
```

```
end


function [stats] = haralick(I)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

%% Struct

stats(1) = struct('Contrast',[],'Correlation',[],'Energy',[],'Entropy',[], 'Homogeneity',[]);

%% Haralick Features

texture_stats = graycoprops(I,'all');

%% Mean Compressed Haralick texture Features

stats.Contrast = mean(texture_stats.Contrast);
stats.Correlation = mean(texture_stats.Correlation);
stats.Energy = mean(texture_stats.Energy);
stats.Entropy = mean(texture_stats.Entropy);
stats.Homogeneity = mean(texture_stats.Homogeneity);

end


function [Features] = hist_features(I)
%HIST_FEATURES Summary of this function goes here
%   Detailed explanation goes here

I = uint8(I);
s = size(I);
[counts,Gray_vector] = imhist(I);
Prob = counts./(s(1)*s(2));

%% Mean
Mean = sum(Prob.*Gray_vector);

%% Variance
Variance = sum(Prob.*(Gray_vector-Mean).^2);

%% Skewness
Skewness  = calculateSkewness(Gray_vector,Prob,Mean,Variance);

%% Kurtosis
Kurtosis = calculateKurtosis(Gray_vector,Prob,Mean,Variance);

%% Energy
Energy = sum(Prob.*Prob);

%% Entropy
Entropy = -nansum(Prob.*log(Prob));
%% Feature Struct

Features = [Mean,Variance,Skewness,Kurtosis,Energy,Entropy];

function Skewness = calculateSkewness(Gray_vector,Prob,Mean,Variance)
% Calculate Skewness
term1    = Prob.*(Gray_vector-Mean).^3;
term2    = sqrt(Variance);
```

```
Skewness = term2^(-3)*sum(term1);
end

function Kurtosis = calculateKurtosis(Gray_vector,Prob,Mean,Variance)
% Calculate Kurtosis
term1    = Prob.*(Gray_vector-Mean).^4;
term2    = sqrt(Variance);
Kurtosis = term2^(-4)*sum(term1);
end

end


function [HistDistH] = HistDistHist(P, TargetImage, n, verbose)

HistDistH = array2table(zeros(n,3),'VariableNames',{'HI_Hue','HI_Sat','HI_Val'});


for i = 1:n

    hsvImage = rgb2hsv(P(i).Image_Norm_Hist);
    hImage = hsvImage(:, :, 1);
    sImage = hsvImage(:, :, 2);
    vImage = hsvImage(:, :, 3);

    hsvTargetImage = rgb2hsv(TargetImage);
    hTargetImage = hsvTargetImage(:, :, 1);
    sTargetImage = hsvTargetImage(:, :, 2);
    vTargetImage = hsvTargetImage(:, :, 3);

    [hSource] = histcounts(hImage,255);
    [hTarget] = histcounts(hTargetImage,255);
    [sSource] = histcounts(sImage,255);
    [sTarget] = histcounts(sTargetImage,255);
    [vSource] = histcounts(vImage,255);
    [vTarget] = histcounts(vTargetImage,255);

%     figure('visible',visibility);
%     subplot(1,3,1);
%     [hSource] = histcounts(hImage,255);
%     histogram(hImage,255);
%     grid on;
%     title('Hue Histogram');
%
%     hold on
%     [hTarget] = histcounts(hTargetImage,255);
%     histogram(hTargetImage,255);
%     hold off
%
%     subplot(1,3,2);
%     [sSource] = histcounts(sImage,255);
%     histogram(sImage,255);
%     grid on;
%     title('Saturation Histogram');
%
%     hold on
%     [sTarget] = histcounts(sTargetImage,255);
%     histogram(sTargetImage,255);
%     hold off
%
%     subplot(1,3,3);
```

```
%     [vSource] = histcounts(vImage,255);
%     histogram(vImage,255);
%     grid on;
%     title('Value Histogram');
%
%     hold on
%     [vTarget] = histcounts(vTargetImage,255);
%     histogram(vTargetImage,255);
%     hold off

    HistDistH.HI_Hue(i) = HistInter(hSource,hTarget);
    HistDistH.HI_Sat(i) = HistInter(sSource,sTarget);
    HistDistH.HI_Val(i) = HistInter(vSource,vTarget);


end


end


function [LL_L1,LH_L1,HL_L1,HH_L1] = L1norm_wavelet(LL,LH,HL,HH)
%UNTITLED7 Summary of this function goes here
%   Detailed explanation goes here


%% L1-norm

[m,n] = size(LL);

LL_L1 = sum(LL(:))./(m.*n);
LH_L1 = sum(LH(:))./(m.*n);
HL_L1 = sum(HL(:))./(m.*n);
HH_L1 = sum(HH(:))./(m.*n);

end


function [out] = lbp_color(I,win_size)
%UNTITLED7 Summary of this function goes here
%   Detailed explanation goes here

ksize = floor(win_size(1)./2);
I_pad = double(padarray(I,[ksize ksize],'symmetric','both'));

m = win_size(1);
n = win_size(2);

C = round(win_size(1)./2);
F = round((n.*m)./2);

row_max = size(I_pad,1)-m+1;
col_max = size(I_pad,2)-n+1;

out = zeros([row_max, col_max]);

for i = 1:row_max
    for j = 1:col_max

        Temp = I_pad(i:i+m-1,j:j+n-1,:);
        Temp = reshape(Temp,[],3);
```

```
        for k = 1:length(Temp)

            Temp_d(k) = sqrt(Temp(k,1).^2 + Temp(k,2).^2 + Temp(k,3).^2);

        end

        thresh = Temp(F,:);
        thresh_d = sqrt(thresh(1).^2 + thresh(2).^2 + thresh(3).^2);

        label = Temp_d-thresh_d;
        label(label>0) = 1;
        label = reshape(uint8(label),m,n);

        out(i,j) = label(C,win_size(1)) + label(win_size(1),win_size(1))*2 + label(win_size(1),C)*4 + label(win_size(1),1)*8 + label(C,1)*16 +
label(1,1)*32 + label(1,C)*64 + label(1,win_size(1))*128;

    end
end

out = uint8(out);

end



function [P,n] = LoadFiles(Img_dir_images)

% LoadFiles
% In this function data is loaded from dataset path
% Images are sorted into desease class and images and info are placed into struct

%% Display

disp('...Loading Data');

%% Image Dataset File information

num_img = dir(Img_dir_images);
num_img = num_img(~ismember({num_img.name},{'.','..','.DS_Store'}));

%% Struct of Data

n = length(num_img);
P(1:n) = struct('Name','','Label',[],'Disease','','Image', []);

%% Read query image set into a struct

for i = 1:n
    if contains(num_img(i).name,'benign')

        P(i).Name = num_img(i).name;
        P(i).Label = 0;
        P(i).Disease = 'Benign';
        P(i).Image = imread(num_img(i).name);

    elseif contains(num_img(i).name,'malignant')

        P(i).Name = num_img(i).name;
        P(i).Label = 1;
        P(i).Disease = 'Malignant';
```

```
        P(i).Image = imread(num_img(i).name);

    else

        error('Error: No image found');

    end
end

end



function [T] = two_sample_ttest(Features,featnames,Label1,Label2,k)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

disp('...Two-sample t-test');

if isstruct(Features) == 1
    Features = cell2mat((struct2cell(Features)))';
end


FSGS_Stats = (Features(Label1,:,k));
MN_Stats = (Features(Label2,:,k));
ttest = zeros(size(Features,2),2);

for i = 1:size(Features,2)

    [h,p] = ttest2(FSGS_Stats(:,i,k),MN_Stats(:,i,k),'Vartype','unequal');
    ttest(i,:,k) = cat(2, h, p);

end

featnames = cellstr(featnames);

%Ttest table
T = array2table(ttest(:,:,k),'VariableNames',{'Hypothesis Test Result','p-value'},'RowNames',featnames);

% Display table
disp(T);

end
```