University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Diabetic Retinopathy Detection Tool (DRDT)

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Christine Eve D. Ocaña

May 2016

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Diabetic Retinopathy Detection Tool (DRDT)" prepared and submitted by Christine Eve D. Ocaña in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Perlita E. Gasmen, M.Sc. (*candidate*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*candidate*)	_____	_____
2. Avegail D. Carpio, M.Sc.	_____	_____
3. Richard Bryann L. Chua, M.Sc.	_____	_____
4. Marvin John C. Ignacio, M.Sc.(*candidate*)	_____	_____
5. Ma. Sheila A. Magboo, M.Sc.	_____	_____
6. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Diabetic retinopathy is one common cause of vision impairment. If not detected and treated, this can also cause blindness. Diabetic Retinopathy Detection Tool (DRDT) is a non-proprietary decision support tool. It classifies fundus/eye images as normal or diabetic retinopathy positive through the use of image processing and support vector machines. The classification is based on the areas of retinal structures and GLCM texture features. DRDT could be used as a starting point for other developers who want to create a decision support tool for DR. Since DRDT's trained classifier gives an average accuracy of 66.6667%, the methods used in the creation of the tool needs to be enhanced to achieve higher accuracy. If the tool is improved, it could be used to provide second opinion to ophthalmologists on detecting DR in patients.

Keywords: diabetic retinopathy, decision support tool, support vector machine, image processing

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Diabetes mellitus, more popularly known as diabetes, is a metabolic disorder of multiple causes characterized by high blood sugar (hyperglycemia). It occurs when there is not enough insulin produced by the pancreas or when the body cannot properly use and respond to the insulin it produces [1]. According to World Health Organization (WHO), there are 347 million people suffering from diabetes worldwide. In 2014, its global occurrence was estimated to be 9% among adults [2]. WHO also projects that diabetes will be the 7th leading cause of death in 2030 [3]. Diabetes can cause complications to the organs, nerves and blood vessels. These are divided into macrovascular complications such as heart attack and stroke, and microvascular complications such as nephropathy (kidney disease), neuropathy (nerve disease), and retinopathy (eye disease).

Diabetic retinopathy (DR) is caused by the damage to blood vessels located in the retina. It is one of the leading causes of gradual vision loss and blindness. Early detection and treatment of DR can prevent or delay the effects of this disease [4]. Both types of diabetes (Type I and Type 2) may result to DR. WHO has estimated that DR is responsible for 4.8% of the millions of cases of blindness in the world [5]. DR is diagnosed with the help of a fundus camera. A fundus camera or retinal camera is a specialized low power microscope with an attached camera that is designed to capture the interior surface of the eye, including the retina, optic disc, macula, and posterior pole [6]. DR patients perceive no symptoms until visual loss develops, which happens at the later stages of the disease where treatment is less effective. The growing incidence of diabetes causes increase in the number of DR cases and images that need to be studied by ophthalmologists. The detection of DR in fundus images is

highly dependent on the physician's observation skills [7].

Technology plays a major role in providing the medical field with fast and accurate diagnosis of diseases. These include early detection of disease in an individual, with the help of computers and images. Image processing is necessary for the analysis of the data contained in the images used in the researches in the medical field. It is used to eliminate erroneous data, enhance the desired parts, and/or output the characteristics of an image by performing operations on it. This results to clearer understanding of the images and more accurate diagnosis of the disease.

Support vector machine (SVM) is a machine learning algorithm used in several pattern recognition problems and classification of objects [8]. It uses a kernel function, which acts on input data [9]. The goal of SVM is to use training data to produce a model which predicts the target values of the test data given the test data attributes [10].

In this study, the researcher created a decision support system for diabetic retinopathy detection using image processing and support vector machine. The inputs used to train the SVM for detection are color fundus images. The images were subjected to processing techniques such as thresholding and feature extraction. The researcher used the extracted features as the training features for the SVM. Based from the model generated by the SVM training process, the SVM was tested to predict which fundus images are normal or diabetic retinopathy positive. After the training and tesing of the SVM, the users of the system are able to input a fundus image to be classified. The system is also able to generate a report for the future analysis of the users of the system.

The data that are used in the study was acquired from the MESSIDOR project. MESSIDOR is a research program which was funded by the French Ministry of

Research and Defense. The database is publicly available, and can be used for research and educational purposes. The downloadable package includes eye fundus color images in TIFF format and excel files with medical diagnosis for each image [11].

B. Statement of the Problem

Early detection of diabetic retinopathy is essential because it is a progressive disease [12]. The problem with DR is that DR patients perceive no symptoms until the changes in the retina of the eye have progressed to a level that treatment will tend to be less effective. It is therefore important for diabetic patients to go through regular screening. Through the analysis of fundus images, the signs of DR in a patient are detected. The examination of these fundus images is observer-driven: it is highly dependent on the observation skills of the physician [7].

Additionally, the diagnosis of DR based only on visual examination poses a challenge because of certain amounts of imaging noise present and other variations in real-life datasets. Medical students/trainees, especially, may not have plenty of experience in interpreting these medical images. It is therefore essential to develop a decision support tool that will overcome these problems and be of help to students/trainees in terms of interpreting fundus images, especially in difficult cases. There is a need for a DR detecting tool to aid analysis of medical students/trainees.

C. Objectives of the Study

Diabetic Retinopathy Detection Tool (DRDT) is a decision support tool for ophthalmologists and medical students/trainees that can aid them in identifying diabetic retinopathy in fundus images. The system detects diabetic retinopathy in

a fundus image by using image processing techniques and a trained support vector machine.

The system is able to do the following:

1. Obtain input data by loading a fundus image into the tool.
2. Process input image by:
    (a) Extracting features using segmentation of retinal structures
        i. Segment blood vessels from the fundus image.
        ii. Segment exudates from the fundus image.
        iii. Segment microaneurysms from the fundus image.
    (b) Extracting textural features using Gray Level Co-occurence Matrix (GLCM)
        i. Compute contrast of the fundus image.
        ii. Compute homogeneity of the fundus image.
        iii. Compute correlation of the fundus image.
        iv. Compute energy of the fundus image.
    (c) Using the extracted features as input for the trained SVM
3. Show the classification results (normal or diabetic retinopathy), that can be exported in a PDF file.
4. Provide viewable tutorials.
    (a) View an introduction on support vector machines.

(b) View an overview on diabetic retinopathy.

D. Significance of the Project

Decision support tools are helpful in the medical industry. They can aid on the classification of many medical cases, even on those which rely heavily on the visual examination and observation skills of the medical professional. Researchers, medical students and trainees on the field of diabetic retinopathy would benefit from the creation of a decision support tool for DR. Such a tool can be used by physicians to provide them a second opinion on the detection of DR cases.

This study could be a basis for a decision support tool for diabetic retinopathy. It can be used by ophthalmology trainees for refining and training their ability to detect DR in fundus images. Since this study is non-proprietary, using it for research, training, or other medical purposes is free of charge. Through this study, DR diagnosis may be a bit closer to having a more automated and standardized process.

E. Scope and Limitations

The following are the scope and limitations of this study:

1. The decision support tool used fundus images from MESSIDOR, a publicly available database. These contain an unknown amount of imaging noise and optical aberrations.

2. The decision support tool does not keep a database of the processed fundus images and related data, and these files are not remembered after closing the tool. The user should use the tool's ability to export the results in a PDF file

for his/her safekeeping and reference.

3. The only image format accepted by the system is TIFF format.

4. The export file generated by the decision support tool is only available in PDF format.

5. The user of the system needs an internet connection to view the other supplementary materials in the tutorials section.

F. Assumptions

The following are the assumptions of this study:

1. The user of the decision support tool is a physician or medical student/trainee.

2. The tool is only used as an aid for medical students/trainees. Final decision and classification of the image is still based on the decision of medical experts.

3. The image that is loaded into the decision support tool is a fundus image.

4. The fundus image loaded into the tool can only contain the possibility of DR and no other diseases.

II. Review of Related Literature

Among the many complications of diabetes, diabetic retinopathy is of particular importance because it may cause blindness if not detected and treated in time. The early detection and diagnosis of this disease is important to avoid the patients' loss of vision [12], hence, there are several studies that tried to develop automated detection of diabetic retinopathy (DR).

One method was done by Priya and Aruna. They used an automated approach for classification of DR disease using fundus images. Image pre-processing was done to enhance the quality of the images. Exudates and haemorrages were also extracted from the images. The feature values: radius, diameter, area, arclength, center angle, and half area, were taken from the blood vessel, haemorrages and exudates detected images. These were fed into the three classification models, namely, probabilistic neural network (PNN), Bayesian classification and support vector machine (SVM), and their performances were compared. The results showed that the SVM outperformed the other models [6].

The approach made by Usher et al. implemented artificial neural network (ANN). Fundus images were pre-processed to standardize color and enhance contrast. Afterwards, the eye structures and candidate lesions were identified. The features of candidate lesions were extracted, then they were identified as true lesion or noise using ANN. The classification of images as normal or DR was done using mathematical rules [13].

Sopharak, Uyyanonvara, Barman and Williamson on the other hand worked on a automatic detector of diabetic retinopathy exudates. Their paper indicated that they focused on exudates, since it is one of the primary signs of diabetic retinopathy, and that detection of exudates requires ophthalmologists to dilate the pupil using a

chemical solution, which takes time and affects patients. They came up with a set of optimally adjusted morphological operators to be used for exudate detection on DR patients' eye images [14].

Another approach for automated DR detection by Selvathi, Prakash, and Balagopal involved the use of feature extraction and SVM. Segmentation followed by extraction was done on retinal images to separate the blood vessels, exudates and microaneurysms (MA) since these are the best indicators of DR. Additionally, Selvathi et al. used texture analysis. The textural features and areas of segmented structures were fed to the SVM which classified the images as normal or DR [15].

Akram, Khalid, and Khan made a three stage system for early detection of MAs using filter banks. The first stage of their work involved extraction and classification of candidate regions as MA or non-MA by formulating a feature vector for each region. The feature vector depended upon properties such as shape, color, intensity and statistics. The classifier used is a hybrid classifier. It is a combination of Gaussian mixture model (GMM), SVM and an extension of multimodal m-Mediod based modelling approach in an ensemble to increase the classification's accuracy [12].

A paper made by García, López, Álvarez and Hornero presented another way of detecting red lesions (RLs) such as MAs in retinal images. After feature extraction was performed on the images, feature selection was made using logistic regression. Four neural network based classifiers (multilayer perceptron (MLP), radial basis function (RBF), SVM, and a combination of these three using a majority voting (MV) schema) were used to get the final segmentation of RLs. The best results were obtained for RDF when it comes to sensitivity, specificity, and accuracy. Their work mentioned that their blood vessel segmentation method should be refined, and that other lesions associated with DR should also be detected because MAs are not usually the ones responsible for vision loss [16].

Meanwhile, Leandro, Cesar and Jelinek's work reports the development of a system for automatic analysis of retinal angiographic images, focusing on the segmentation of blood vessels in these images, starting on the technique called mathematical morphology. They also used an approach called the continuous wavelet transform using the Morlet wavelet. Furthermore, they mentioned that the mathematical morphology method was able to detect finer detail more precisely. Their results suggested that an interesting direction to be investigated is to use both approaches together to obtain better results [17].

Another approach in segmenting blood vessels was done by Mudassar and Butt [18]. They used 4 different techniques to extract blood vessels from retinal images. These were: Image Line Cross-Sections (ILCS), Edge Enhancement and Edge Detection (EEED), Modified Matched Filtering (MMF) and Continuation Algorithm (CA). Their investigations revealed that the four techniques in the order of increasing performance could be arranged as ILCS, MMF, EEED, and CA. They concluded that CA is a promising technique.

Noronha et al. proposed a decision support system for detection of normal and DR classes using discrete wavelet transform (DWT) and SVM. Adaptive histogram equalization was used to remove the non-uniformity of the background of images taken by a retinal camera. The preprocessed images were subjected to second level decomposition using of DWT. The significance of the extracted textural features were evaluated using the Student's t-test. These were then fed to the SVM classifier for classification as normal or DR [19].

Support vector machine is a machine learning algorithm that can be used as classification tool for objects. It maximizes a particular function with respect to a collection of data [8]. SVMs have many applications in several fields.

In Sahin and Duman's work, SVM and decision trees are applied to credit card fraud detection problem. Accounts were monitored separately using suitable descriptors, and the transactions were attempts to be identified and flagged as legitimate. These transactions were identified as fraud or not based on the suspicion score produced by the developed classifier models [20].

Support vector machines are also used for pattern recognition. Munir, Gupta, Nemade and Alam used SVM, eigenface method and a combination of the two for face recognition [21]. Meanwhile, Pan, Shen P. and Shen L proposed a paper on speech emotion recognition using SVM. A German Corpus and a self-built Chinese emotional database were used for training the SVM to classify data into one of the three emotional states: happy, sad or neutral [22].

Several SVM-based systems and tools are being developed to help in the medical field. An example is for detection of breast cancer. Chen, Yang, Liu J. and Liu D. used a rough set based supporting vector machine classifier for breast cancer diagnosis (RS_SVM). The method employed RS reduction algorithm as a feature selection tool in order to eliminate redundant features, thus, increasing the accuracy of the SVM classifier. The proposed method (RS_SVM) achieved very high classification accuracy [23].

Virmani, Kumar, Kalra and Khandelwal proposed a system to characterize normal liver, cirrhotic liver and hepatocellular carcinoma (HCC) evolved on cirrhotic liver. The system consisted of three modules namely: feature extraction, feature selection and classification. Feature extraction used second level decomposition by 2D wavelet packet transform (2D-WPT). Feature selection used genetic algorithm-support vector machine (GA-SVM) to reduce the feature vectors. A multiclass SVM classifier was used in the classification with the aid of LibSVM library [24].

Brain tissues in magnetic resonance images (MRI) were classified using GA and SVM in the work of Kharrat et al. They used spatial gray level dependence method (SGLDM) to extract optimal texture features from normal and tumor regions. Feature selection was done by GA, and SVM classified the brain tissues into normal, benign or malignant tumor [25].

III. Theoretical Framework

A. Diabetic Retinopathy

Diabetic retinopathy (DR) is a disease which damages the blood vessels in the retina due to prolonged diabetes. DR progresses with time without any distinguishing symptoms until damages to the eye such as vision loss or blindness has occurred [19]. Both types if diabetes (Type I and Type II) can cause DR [4]. DR usually affects both eyes and it causes blood vessels to swell and leak fluid, while in some cases, abnormal new blood vessels grow on the surface of the retina [26]. This will produce features such as microaneurysms (MAs), hard exudates and cotton wool spots or soft exudates [19].

Diabetic Retinopathy has four stages [19, 26]:

1. **Mild nonproliferative retinopathy (Mild NPDR)** This is the earliest stage of DR. MAs start to occur at this stage. These are small areas of balloon-like swelling in the retinas tiny blood vessels.

2. **Moderate nonproliferative retinopathy (Moderate NPDR)** In this stage of DR, some blood vessels that nourish the retina are blocked. Several MAs and exudates may be present.

3. **Severe nonproliferative retinopathy (Severe NPDR)** As the disease progress to this stage, More MAs and exudates in the eye become present. More blood vessels are blocked, depriving several areas of the retina with their blood supply. This areas send signals to the body to grow new blood vessels. There is 50% chance of progression of patients with severe NPDR to PDR within 1 year.

4. **Proliferative retinopathy (PDR)** - In this last stage of DR, the signals sent

by the retina for nourishment trigger the growth of new but abnormal and fragile blood vessels. By themselves, these blood vessels do not cause anything, however, due to their thinness, blood leakage may occur, which results may result in severe vision loss and even blindness.

DR is diagnosed with the help of a fundus camera. A fundus camera or retinal camera is a specialized low power microscope with an attached camera that is designed to capture the interior surface of the eye, including the retina, optic disc, macula, and posterior pole [6]. The growing incidence of diabetes increases the number of DR cases and images that need to be studied by ophthalmologists [7].

Figure 1: Fundus image with no DR

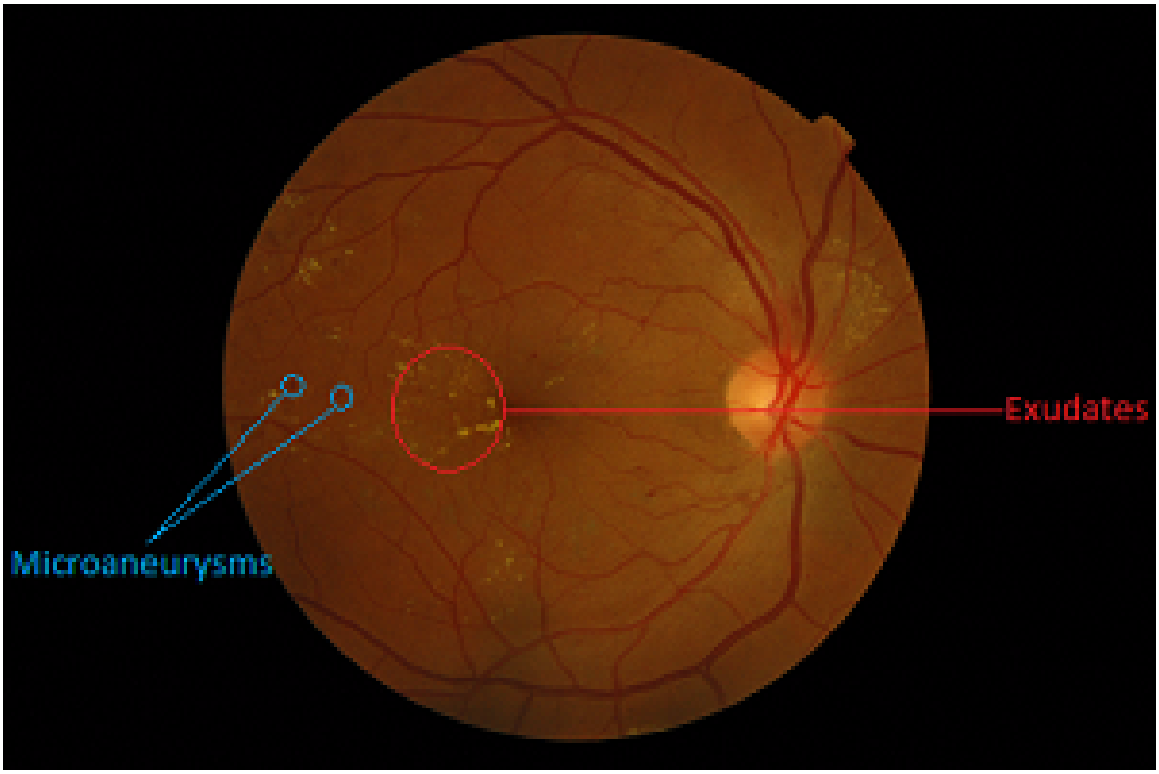

Figure 2: Fundus image exhibiting DR

B. Decision Support System

A decision support system (DSS) is a computer based application that helps people make decisions based on data collected from a wide variety of fields/sources [27]. A benefit of DSS would be speeding up the decision making process in different fields such as in the medical field. Applications of DSS in the clinical setting may improve quality of diagnosis and support in determining the type or severity of disease a person has.

C. Median Filtering

Median filtering is a process used for reducing impulsive, or salt and pepper noise while preserving useful details in an image. Salt and pepper noise can occur due to

errors in a communication channel [28].

The median filter studies each pixel in an image and compares it to its neighboring pixels in order to decide if it is representative of its surroundings or not. It replaces the pixel value with the median intensity value of the neighboring pixels. The median is calculated by sorting the pixel values of the neighborhood into numerical order and replacing the current pixel with the pixels value in the middle (or averaging the two middle pixels values if the number of pixels is even). An example is shown below [29]:

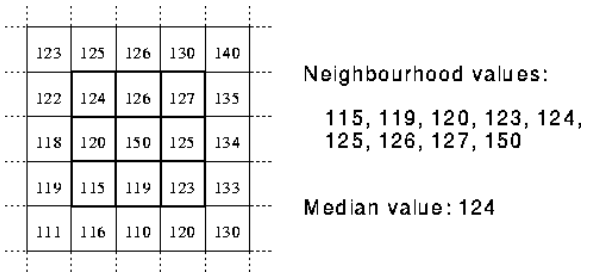

Figure 3: Sample calculation of median pixel value of a neighborhood

D. Contrast Limited Adaptive Histogram Equalization

Adaptive Histogram Equalization (AHE) is a modification of histogram equalization to improve contrast in an image. It yields better results for images which contain local regions having a low contrast bright or dark regions than the normal histogram equalization. AHE considers only small regions, and based on their local cumulative distribution function, does a contrast enhancement of those regions [30].

Ketham, Hummel and Pizer invented the basic form of adaptive histogram equalization. This method involves applying the histogram equalization mapping to each pixel. The histogram equalization mapping is based on the pixels in its contextual region. This simply means that each pixel is mapped to an intensity proportional to its rank in the pixels around it [31]. Figure 4 shows an example of AHE applied in an

image.

Figure 4: Adaptive histogram equalization applied on an image

If a region that is undergoing AHE has a small intensity range, then the noise in that region will be more enhanced, and other artifacts may also appear on those regions. A modification of AHE, **Contrast Limited Adaptive Histogram Equalization (CLAHE)** was therefore done to limit the appearance of such artifacts and noise. Because amount of contrast enhancement for some intensity is directly proportional to the slope of the Cumulative Distribution Function (CDF) at that intensity level, the contrast enhancement can be limited by limiting the slope of the CDF. The slope of CDF is determined by the height of the histogram, hence by limiting the height of the histogram, the amount of contrast enhancement can also be limited [32].

E. Morphological Operations

Morphological operators take a grayscale or binary image and a structuring element (SE) as input and combine them using a set operator. They process the objects in the input image based on shape characteristics, which are encoded in the SE. Morphological operations include erosion, dilation, opening, closing, filling, and many more [33]. Rhody in [34] wrote an overview on morphological image processing.

Erosion, denoted as $A\theta B$ computes for the minimum of each pixels neighborhood

and satisfies the following equation:

$$A\theta B = \{s|(B)_s \subseteq A\}$$

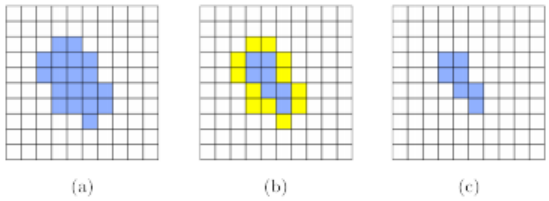

Figure 5: Example of erosion operation

Dilation, denoted as $A \oplus B$, computes the maximum of each pixels neighborhood and satisfies the following equation:

$$A \oplus B = \{s|(B)_s \cap A \neq \emptyset\}$$

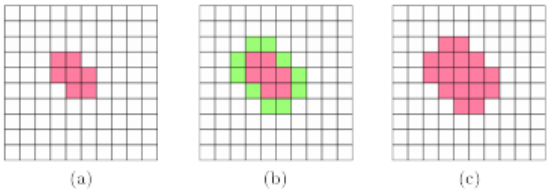

Figure 6: Example of dilation operation

Opening, as discussed in [35] is denoted as $A \circ B$. It is simply an erosion followed by a dilation. It satisfies the following equation:

$$A \circ B = (A \ominus B) \oplus B$$

On the other hand, [35] defines **Closing** to be denoted as $A \bullet B$. It is a dilation followed by an erosion. It satisfies the following equation:

$$A \bullet B = (A \oplus B) \ominus B$$

Region filling is an algorithm to fill in bounded regions with the background color [34]. The image below illustrates filling in connected regions in maze.

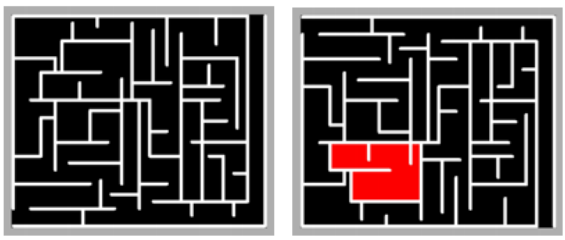

Figure 7: Example of region filling

Other morphological operations include **top-hat** and **bottom-hat**. If intensity values in a gray-level image are thought of as elevations, then the image is composed of mountain tops and valley lows. The top-hat filter is used to preserve bright regions(mountain tops) while bottom-hat is used to preserve dark regions(valley lows), both resulting to enhanced contrast [36].

The morphological operations which are involved in this work are **closing**, **dilation**, **erosion**, **filling**, **opening**, and **top-hat** operation.

F. Otsu Thresholding

Binarization of a grayscale image is a common image processing task. One of the ways to convert a greyscale image into monochrome is Otsu's method. This method is named after its inventor, Nobuyuki Otsu [37].

Otsu thresholding is an unparametric and unsupervised method of automatic threshold selection for picture segmentation. This method is different from the other

methods, in such a way that, the optimal threshold is selected by the discriminant criterion so as to maximize the separability of the resultant classes in gray levels. The problem would eventually be reduced to an optimization problem to search for a threshold that maximizes one of the discriminant criterion measures. This standpoint is influenced by a conjecture: a threshold which gives the best separation of classes in gray levels would be the best threshold. Otsu presented several experimental results to support the validity of the method [38].

Otsu's method involves calculating a measure of spread (variances) for the pixels that fall in each side of the threshold, i. e. the pixels that either fall in the foreground or background, throughout all the possible threshold values. The aim is to find the threshold value where the sum of the spread in both the foreground and background is at its minimum [37]. Figures 8 and 9 shows an example of Otsu thresholding algorithm.

Figure 8: Image before Otsu's method

Figure 9: Image after Otsu's method

G. Gray Level Co-occurence Matrix

Gray level co-occurrence matrices (GLCM) is a statistical method of texture analysis. It looks at the spatial distribution of the gray values of the pixels of an image. GLCM defines the texture of an image by calculating the number of times a pair of pixels with specific gray-level occur in a specified spatial relationship in an image, and placing these computed values into a matrix [39].

Gadkaris work in [40] gave a background on how GLCM works. The GLCM of an image is computed using a displacement vector denoted by d, which is defined by its radius δ and orientation θ. GLCM is always of size NN, where N is the number of gray-level values in the image, and its starting index is the lowest gray-level value of the image. Given figure 10 which shows an image whose pixels are represented by gray-level values of 0 through 3, figure 11 gives the general form of the GLCM for that image. The $\#(i, j)$ in each cell represents the number of times the gray-levels i and j have been neighbors with respect to the conditions stated by the displacement

vector d.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Figure 10: Gray-level values of an image

Gray tone	0	1	2	3
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

Figure 11: General form of a GLCM

Assuming that the given radius $\delta = 1$ and orientation $\theta = 0°$, $\#(i, j)$ in each cell is computed by counting the number of instances the gray-level i is 1 pixel and 0° away from gray-level j (that is, i is to the immediate right or left of j). Figure 12 shows the resulting GLCM given this configuration.

The value of cell $(0, 0)$ is 4 because there are four instances where $i = 0$ is to the immediate right or left of $j = 0$. This can be seen in figure 13. The other cells are computed the same way, by using their respective i, j.

The GLCM for radius $\delta = 1$ and orientations $\theta = 45°, 90°, 135°$ are shown in figures 14, 15 and 16, respectively [40].

4	2	1	0
2	4	0	0
1	0	6	1
0	0	1	2

Figure 12: GLCM with $\delta = 1$ and $\theta = 0°$

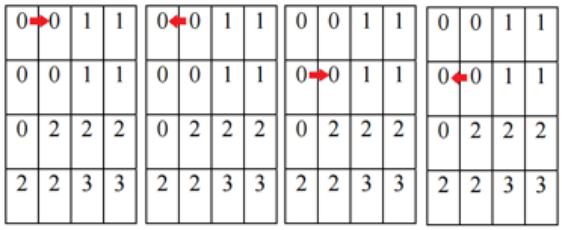

Figure 13: Instances where $i = 0$ is beside $j = 0$

4	1	0	0
1	2	2	0
0	2	4	1
0	0	1	0

Figure 14: GLCM with $\delta = 1$ and $\theta = 45°$

The textural features that are extracted via GLCM in this work are **contrast**, **homogeneity**, **correlation**, and **energy**. These are given by the following [15]:

$Contrast = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (i-j)^2 P_{i,j}$ where $P_{i,j}$ are the elements of the GLCM,

$Homogeneity = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{P_{i,j}}{(1+|i-j|)}$,

$Correlation = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{(i-\mu_i)(j-\mu_j)P_{i,j}}{\sigma_i, \sigma_j}$ where μ_i, μ_j are the means, and σ_i, σ_j

6	0	2	0
0	4	2	0
2	2	2	2
0	0	2	0

Figure 15: GLCM with $\delta = 1$ and $\theta = 90°$

2	1	3	0
1	2	1	0
3	1	0	2
0	0	2	0

Figure 16: GLCM with $\delta = 1$ and $\theta = 135°$

are the standard deviations of $P_{i,j}$, and

$$Energy = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P_{i,j}^2.$$

H. Support Vector Machine

Nobles work in [8] gave a brief introduction on support vector machine. **Support vector machine (SVM)** is a supervised machine learning algorithm used in several pattern recognition problems and classification of objects. Originally developed by Vladimir Vapnik, it is a computer algorithm that learns by example to assign labels to objects. SVM, like other classifier algorithms, has the concept of treating objects to be classified as points in a high dimensional space, and separate them using a line called

the separating hyperplane. The difference between SVM and other hyperplane-based classifiers is the way the separating hyperplane is selected.

SVM selects the maximum-margin separating hyperplane, where the margin of the hyperplane refers to the distance from the separating hyperplane to the nearest expression vector. Selecting this hyperplane maximizes the SVMs ability to predict the correct classification of data. The goal of the SVM is to achieve the largest margin that will classify the data correctly.

Fletchers paper in [41] explained how SVM works:

Given L training points, where each input x_i has D attributes, and is either of class $y_i = -1$ or $+1$. Assuming the data is separable, the hyperplane can be described by the equation

$$w \bullet x + b = 0$$

where w is the normal to the hyperplane and $\frac{b}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin.

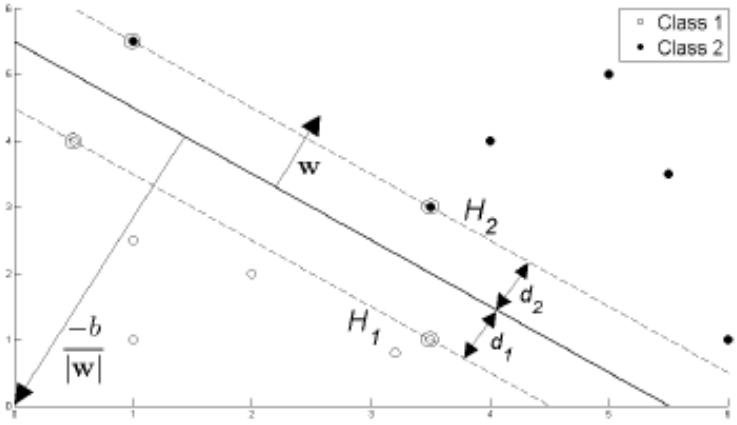

Figure 17: Hyperplane through two linearly separable classes

The encircled points in figure 17 are called support vectors. Support Vectors are the points closest to the separating hyperplane. They contain all the information needed to solve the classification problem. The aim of SVM is to orientate this hyperplane to be as far as possible from the support vectors.

An important factor in SVM is the margin between the support vectors. In figure 17, these are defined as d_1 and d_2. This margin places the hyperplane equidistant from the support vectors ($d_1 = d_2$). To make the hyperplane as far from the support vectors as possible, this margin needs to be maximized. Since the margin can be computed as $\frac{1}{\|w\|}$, maximizing the margin is equivalent to minimizing the value of $\|w\|$. This particular optimization problem is solved by the Langrange function:

$$L_p = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{L} \alpha_i y_i (x_i \bullet w + b) + \sum_{i=1}^{L} \alpha_i \text{ where } \alpha_i \geq 0, \forall_i.$$

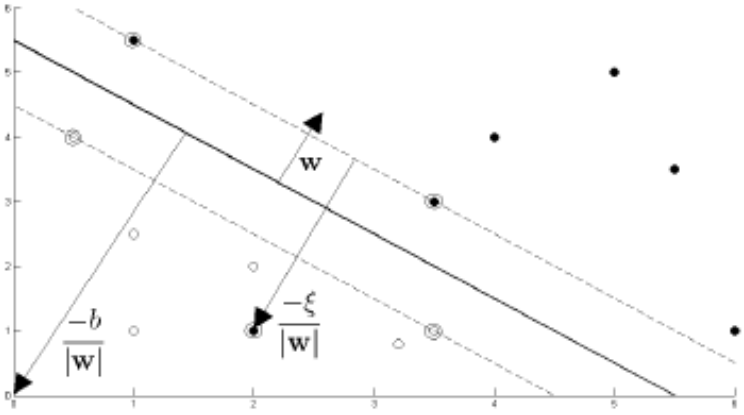

Figure 18: Hyperplane through two non-linearly separable classes

Kernel functions (Φ) map training vectors x_i into a higher dimensional space in order to find a linear separating hyperplane with the maximal margin in this higher dimensional space. There are several kernel functions in SVM, but the popular kernel functions are listed below [42]:

- Linear kernel

$$K(x_i, x_j) = x_i^T x_j$$

- Polynomial kernel

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$$

- Sigmoid kernel

$$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$$

- Radial basis function (RBF) kernel

$$K(x_i, x_j) = exp(-\gamma \left\| x_i - x_j \right\|^2)$$

The SVM type used in the training is **C_SVC** while the kernel used is **Radial Basis Function (RBF)**.

IV. Design and Implementation

A. MESSIDOR Project

The data that are used in the study were acquired from the MESSIDOR project. **MESSIDOR** is a research program which was funded by the French Ministry of Research and Defense. The MESSIDOR database has been established to facilitate studies on computer based diagnoses of diabetic retinopathy. The database is publicly available, and can be used for research and educational purposes [11].

MESSIDOR database contains fundus color numerical images of the posterior pole. These were acquired by three ophthalmologic departments using a color video 3CCD camera on a Topcon TRC NW6 non-mydriatic retinograph, with a 45 degree field of view. The downloadable images are packaged in 3 sets, one per ophthalmologic department, and each set is divided into 4 zipped sub sets which contains images in TIFF format. These zip files also contain an Excel file with medical diagnoses for each image [11].

Two diagnoses have been provided by medical experts tied for each image in [43]: Retinopathy grade and Risk of macular edema.

All of the images in the database were used for actual clinical diagnoses. For privacy purposes, the MESSIDOR team removed patients sensitive information, and no image can be used alone or in combination to identify any subject involved. The download page for the images also include links to other databases with retinal images such as Drive project (retinal color images and results of automatic segmentation of blood vessels) [43].

B. System Design

The Diabetic Retinopathy Detection Tool (DRDT) is a tool to detect the presence of DR given eye images through the use of image processing and support vector machines. The system has two phases: pre-release phase and release phase. The context diagrams for these are shown in figures 19 and 20.

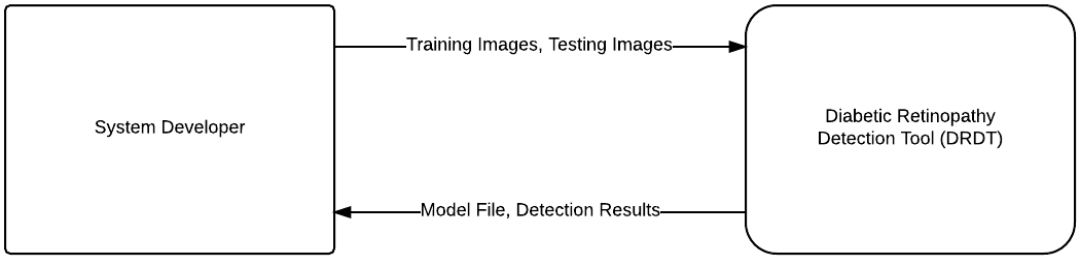

Figure 19: Context Diagram for pre-release phase, DRDT

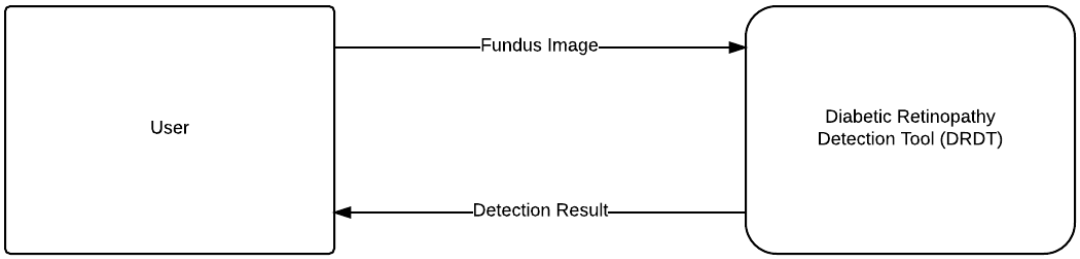

Figure 20: Context Diagram for release phase, DRDT

Figure 21 shows the general (top-level) flow of data in DRDT after its release. The user can load a fundus (eye) image into the tool by placing the path where the file is located. This image is classified as normal or DR positive by the trained system. The results can be exported to a PDF file and downloaded by the user by specifying where to put the file into his/her file system.

The process of detection of DR in fundus image includes segmentation of retinal structures done simultaneously with feature extraction using GLCM. The features are

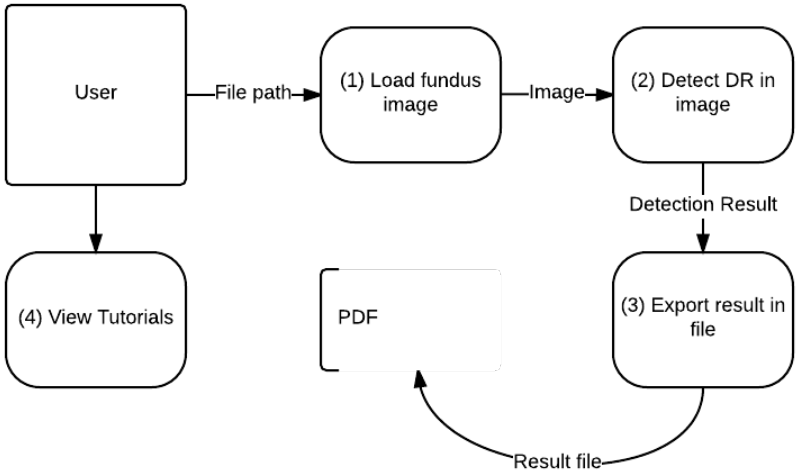

Figure 21: Top-level data flow diagram, DRDT

then used by the trained SVM for prediction of the input image. This is shown in figure 22.

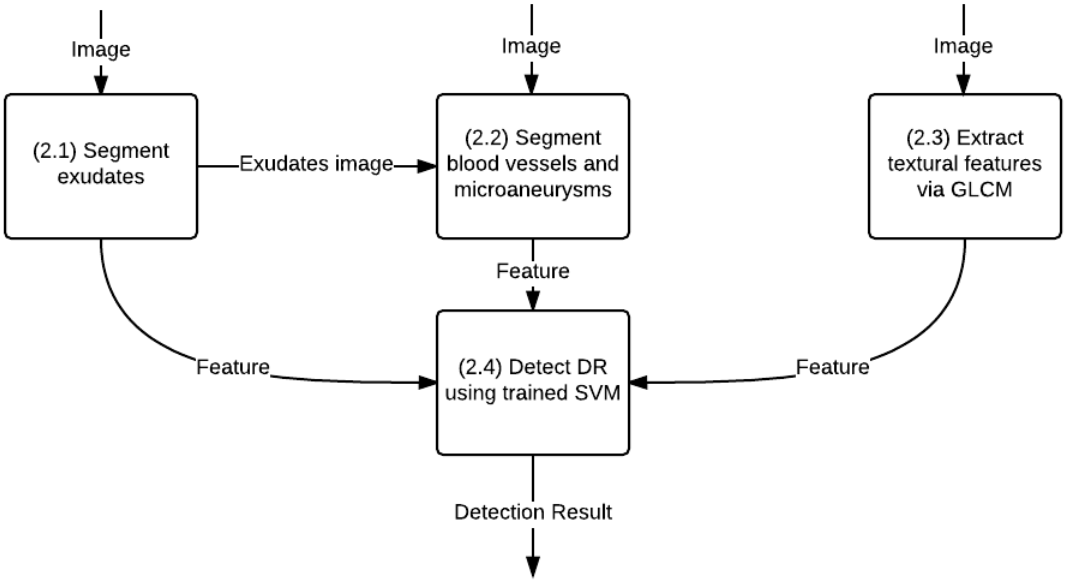

Figure 22: Sub-explosion of process 2 (Image classification)

The process for exudates segmentation is shown in figure 23. The steps are as follows: The intensity channel is extracted from the input image, then median filter is

applied to it. The median filter image undergoes contrast limited adaptive histogram equalization (CLAHE), then a process to isolate the optic disc. Meanwhile, closing operation is applied on the CLAHE image. Then, the standard deviation image is obtained. The resulting image is thresholded at automatically selected grey levels using the Otsu algorithm. The image undergoes dilation to include the neighboring pixels of the thresholded result. The enclosed areas were then filled so that not only the borders of exudates are included. Afterwards, the optic disc is removed from the image using the previously detected optic disc. To create a marker image, the image without optic disc is subtracted from the CLAHE image. This is morphologically reconstructed, using the original CLAHE image as mask. The final exudates image is obtained by applying an Otsu thresholding on the difference between the CLAHE image and the reconstructed image. After the exudates image is obtained, the area of the exudates is computed by finding the total number of white pixels in the image. This is used as a feature for DR detection.

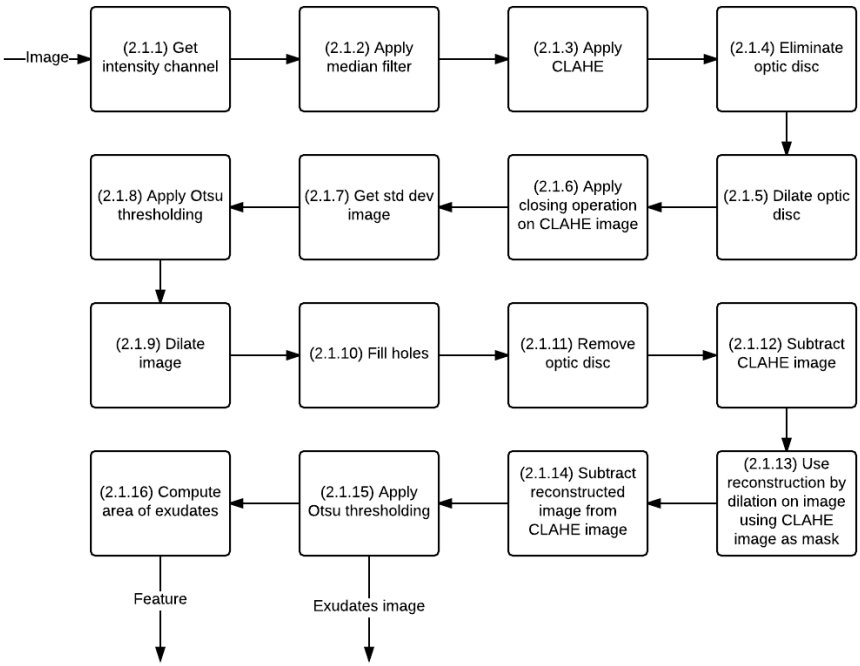

Figure 23: Sub-explosion of process 2.1 (Segmentation of exudates)

Figure 24 shows the details of the segmentation of blood vessels and microaneurysms. The basic structure of the blood vessels image and the smaller blood vessels and the microaneurysms image are obtained simultaneously. Using these two images and the exudates image, the blood vessels of the original image are segmented. After the blood vessels image is obtained, the area of the blood vessels is computed by finding the total number of white pixels in the blood vessels image. This is a feature used for the detection of DR on the fundus image. Using the segmented blood vessels, exudates image and the small blood vessels and microaneurysms image, the microaneurysms of the original image are segmented. The area of the microaneurysms is computed by finding the total number of white pixels in the microaneurysms image. This is a feature used for the detection of DR on the fundus image.

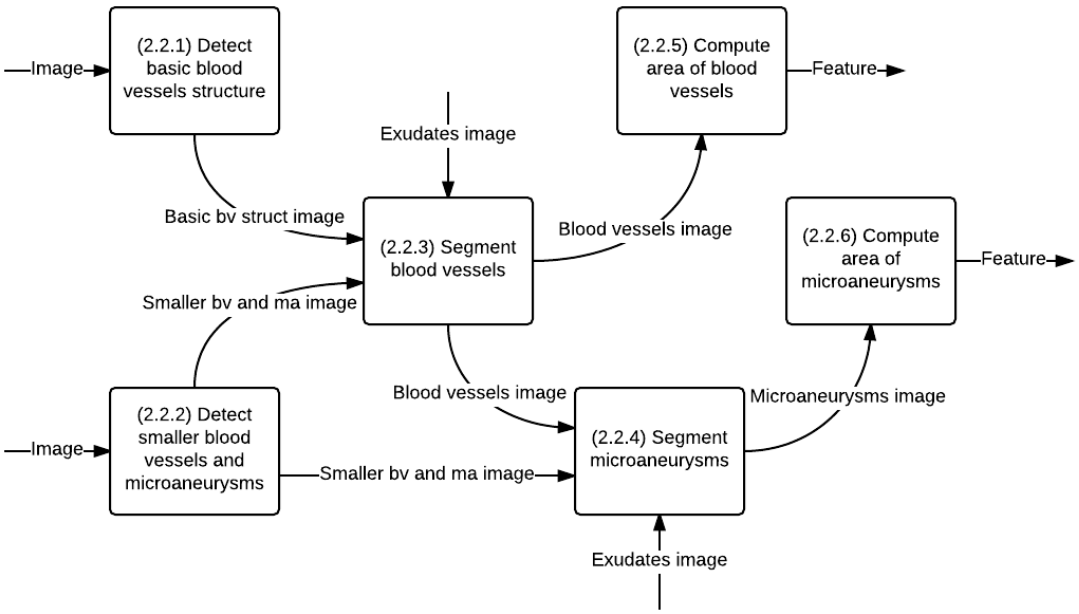

Figure 24: Sub-explosion of process 2.2 (Segmentation of blood vessels and microaneurysms)

The process for the detection of the basic structure of blood vessels is detailed on figure 25, while figure 26 details the detection of smaller vessels and microaneurysms. Figure 27 illustrates the process of segmentation of blood vessels, and figure 28 shows

the process of segmentation of microaneurysms.

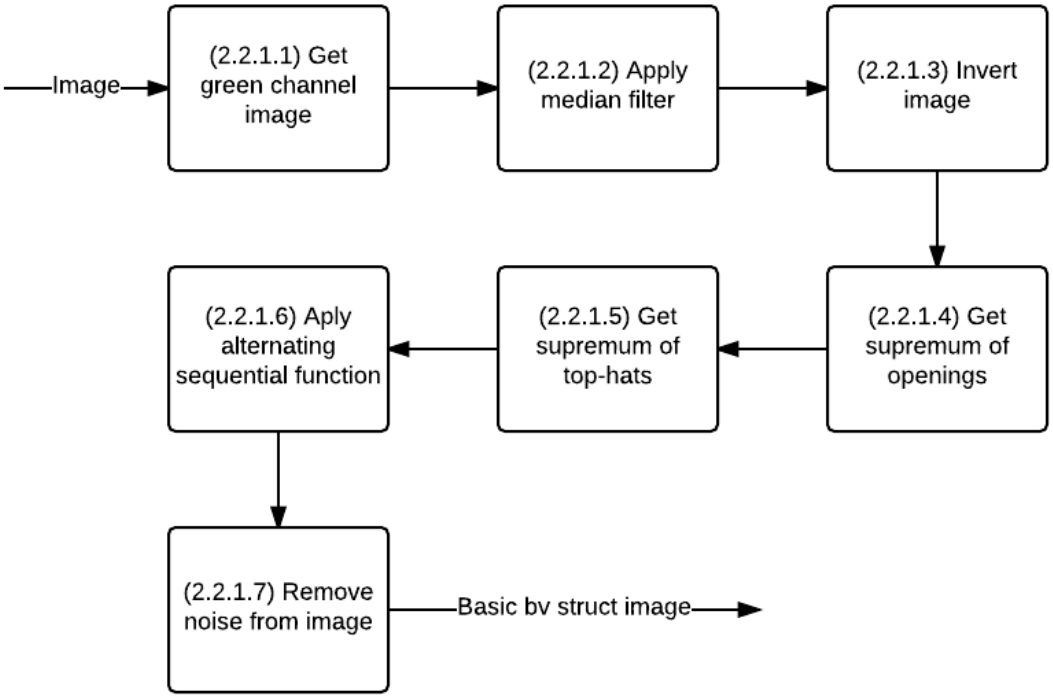

Figure 25: Sub-explosion of process 2.2.1 (Detection of basic blood vessels structure)

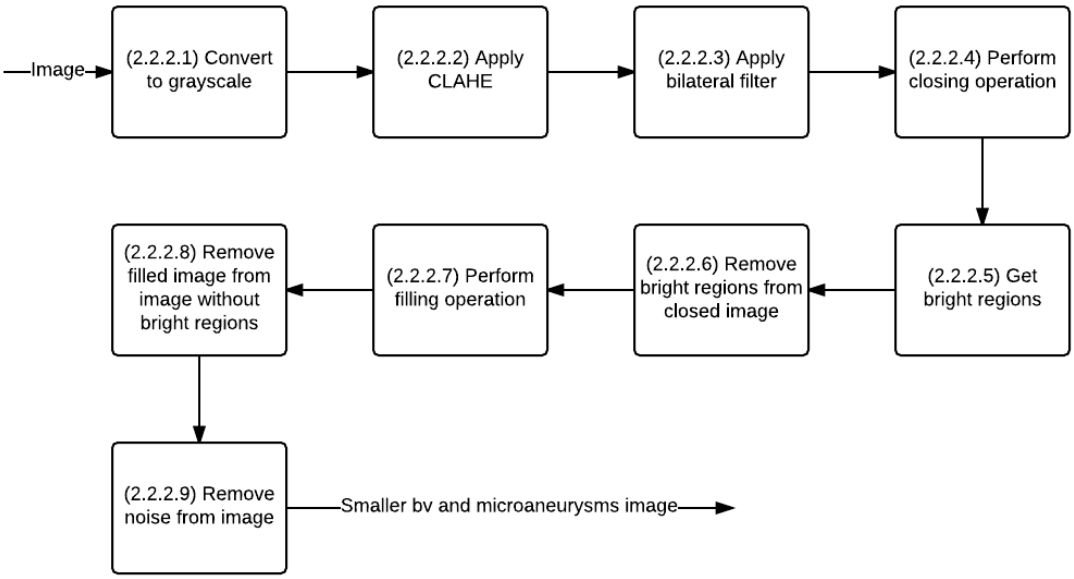

Figure 26: Sub-explosion of process 2.2.2 (Detection of smaller blood vessels and microaneurysms)

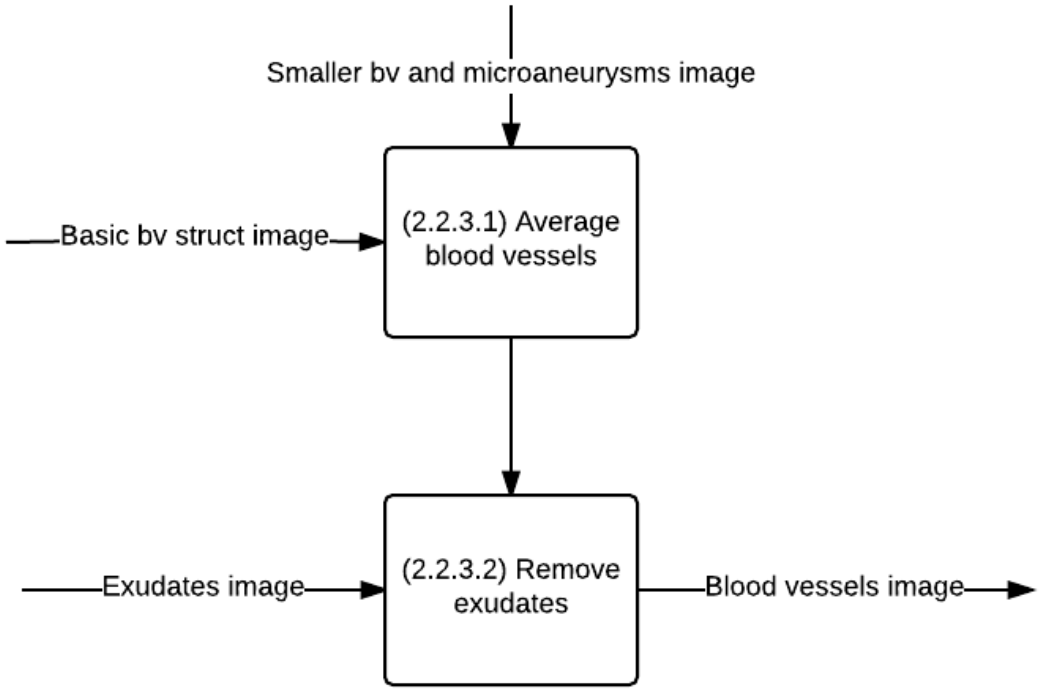

Figure 27: Sub-explosion of process 2.2.3 (Segmentation of blood vessels)

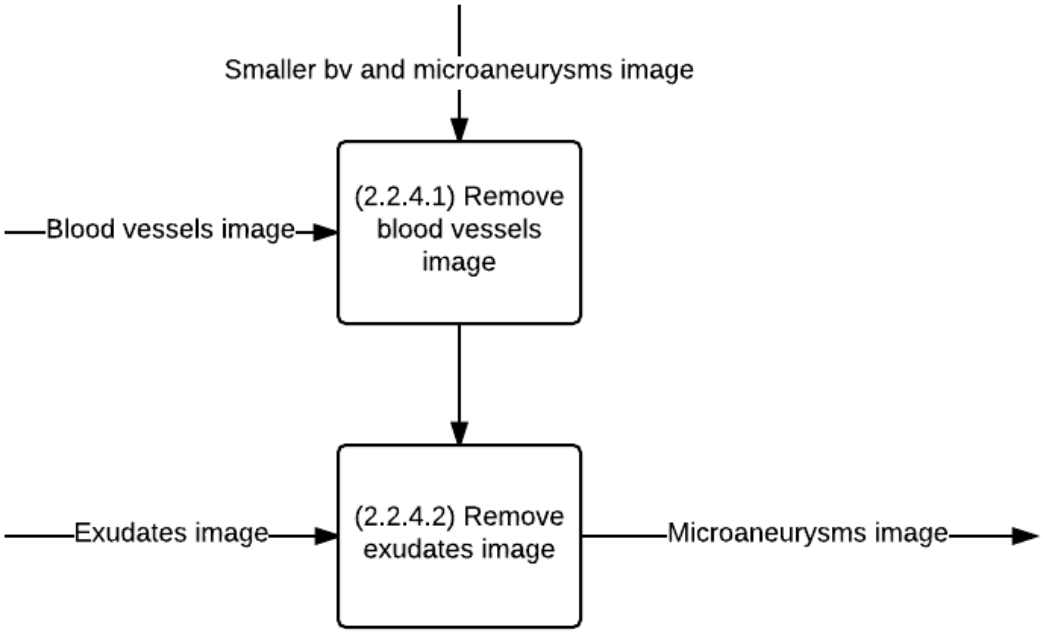

Figure 28: Sub-explosion of process 2.2.4 (Segmentation of microaneurysms)

Textural features are also used to classify the fundus images based on the presence of DR. GLCM is used in order to obtain these textural features. Figure 29 illustrates the process of extracting these features using GLCM. The textural features that are used in the tool are contrast, homogeneity, correlation, and energy.

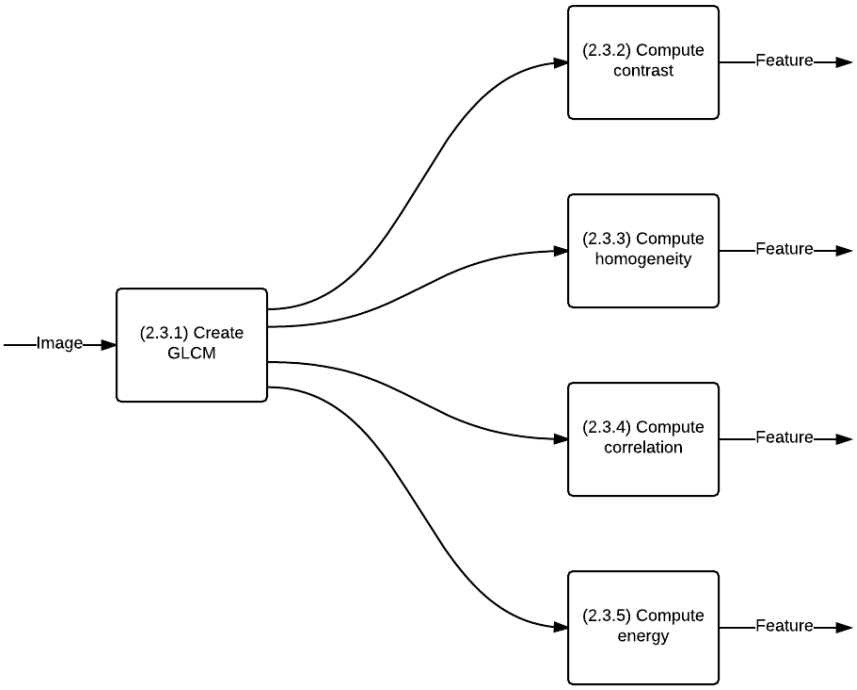

Figure 29: Sub-explosion of process 2.3 (Feature extraction using GLCM)

Medical students and trainees might be interested in how the tool works. Similarly, some developers who wanted to improve the system might be interested in the SVM algorithm used in this paper. Hence, videos and other learning materials are available for user's viewing. Figure 30 shows the viewable tutorials.

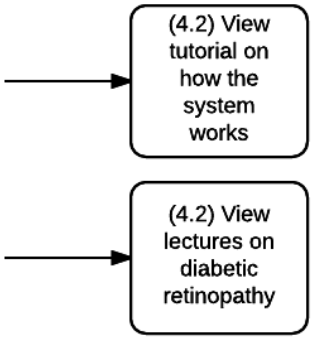

Figure 30: Sub-explosion of process 4 (Feature extraction using GLCM)

C. Technical Architecture

It is recommended that the machine where DRDT will be installed has at least the same specifications as the machine that is used in the study. Table 1 shows these specifications.

Component	Description
Operating system	Windows 8
Processor	Intel ®Core(TM) i3-4005U CPU @1.70GHz
Installed memory (RAM)	4.00 GB
System type	64-bit operating system

Table 1: Specifications of the machine that is used

V. Results

Diabetic Retinopathy Detection Tool (DRDT) is developed using **C++ programming language** and **Microsoft Visual Studio 2013**. It used the open-source libraries **Qt** (for the user interface) and **Open Source Computer Vision** or **OpenCV** (for the image processing techniques and support vector classification part of the DRDT, such as training of the system). The target users of this tool are physicians and medical students/trainees involved in the DR field.

A. Pre-release Phase

This phase is the development phase. In this phase, the program was used for training and testing the SVM classifier that is used in the next phase. The 300 fundus images that were used in this phase, were downloaded from MESSIDOR, and their locations were specified in the text files labeled *DRPositive* and *DRNegative* which were both placed in the Model folder in the same directory as the executable file of DRDT. Both DR-positive and normal have 150 images (50%) each. Before the training and testing of the SVM, the images contained in the text files were split into training and testing datasets. The program randomly selected without replacement 210 images (70%) of the dataset for training (105 images for each class), leaving the remaining 90 images (30%) for testing (45 images for each class) the accuracy of the SVM classifier. Before the training of the classifier, the training dataset had to undergo image processing for feature extraction as shown in figure 31.

The training of the classifier generated an XML file containing the SVM model. This file was created in the Model folder in the same directory as the executable file of DRDT. Figure 32 shows the generated model file.

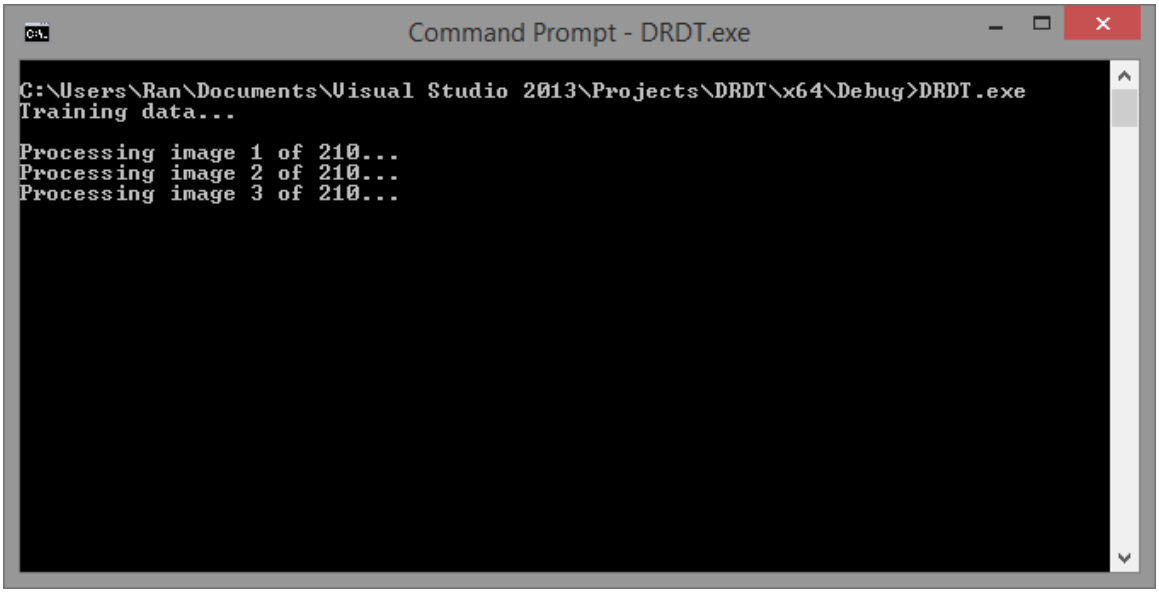

Figure 31: Image processing of images for training set, DRDT (pre-release)

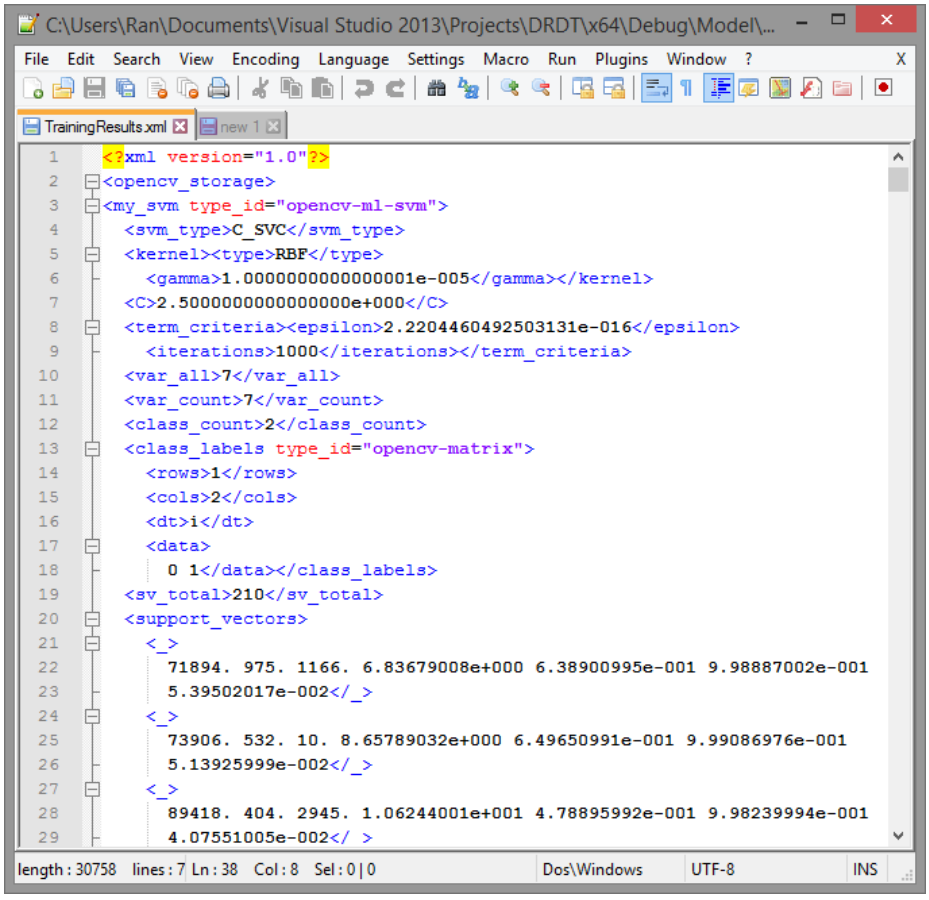

Figure 32: Model file generated by the tool after training, DRDT(pre-release)

After the training, image processing was done on the testing images as well, as shown in figure 33. The model file created after the training process was loaded into the tool during testing.

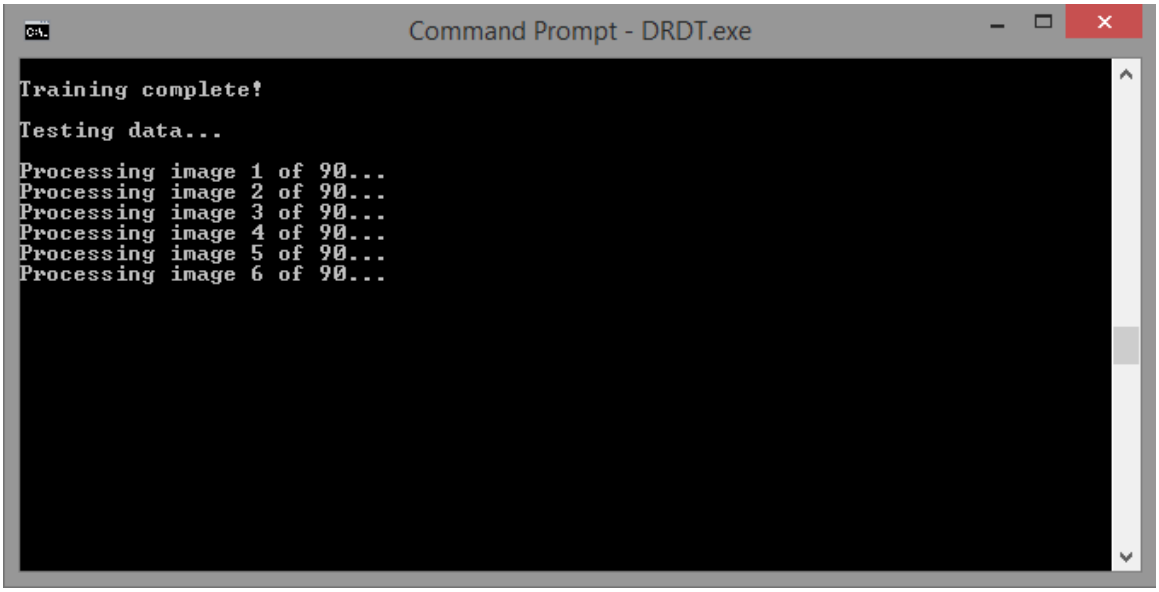

Figure 33: Image processing of images for testing set, DRDT (pre-release)

After the training and testing of the SVM, the results and details of the training (such as the number of training and testing images, accuracy of the SVM, other SVM parameters used) were shown to the developer via the command prompt. This is illustrated in figure 34.

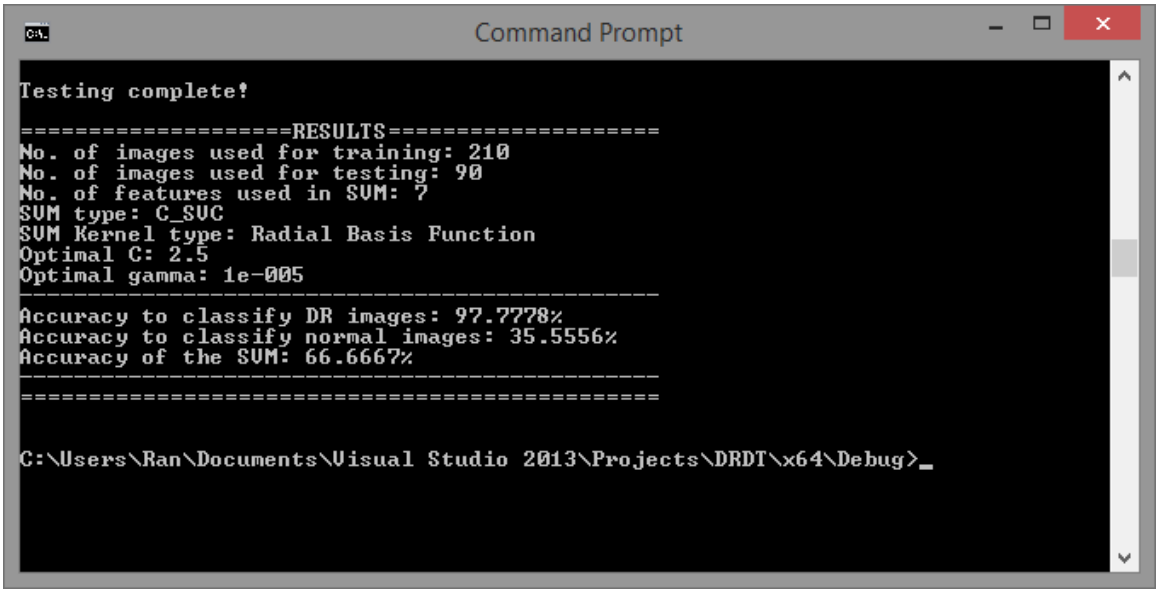

Figure 34: Results of training and testing, DRDT(pre-release)

As seen from the screenshot, the accuracy of classifying DR positive images is *97.7778%*, while the accuracy of classifying normal images is only *35.5556%*. Hence the average accuracy of the SVM is *66.6667%*.

B. Release Phase

DRDT's user interface is divided into the following tabs: the *Home* tab, the *Classify* tab, the *Tutorials* tab, and the *FAQ* tab. Figure 35 shows the user interface of DRDT after opening. It has clickable buttons that lets the user navigate the tabs. Each of the tabs has a *Home* button if in case the user wants to return to this screen.

Figure 35: *Home* tab, DRDT(release)

The main functionality of the DRDT lies on the *Classify* tab. Figure 36 shows how the *Classify* tab looks like.

The tab is accessible by either clicking the *Start Classification* button in the *Home* tab or clicking the *Classify* tab itself. It is where the user may select an image to classify as DR-positive or normal.

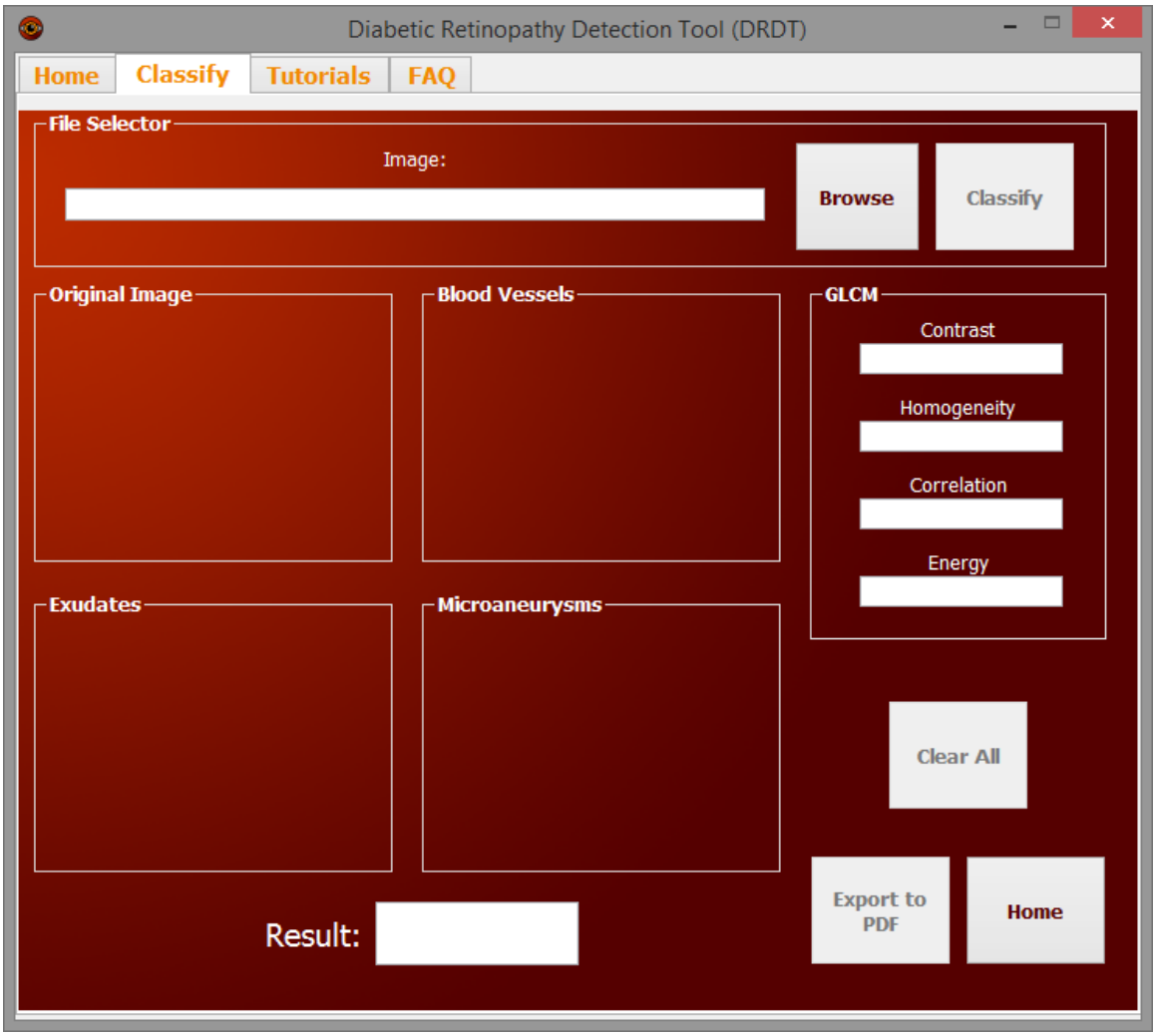

Figure 36: *Classify* tab, DRDT(release)

In order to classify an image, the user must select an image by clicking the *Browse* button, then selecting which image the user wants to classify. Figure 37 shows the dialog box that opens when *Browse* button is clicked.

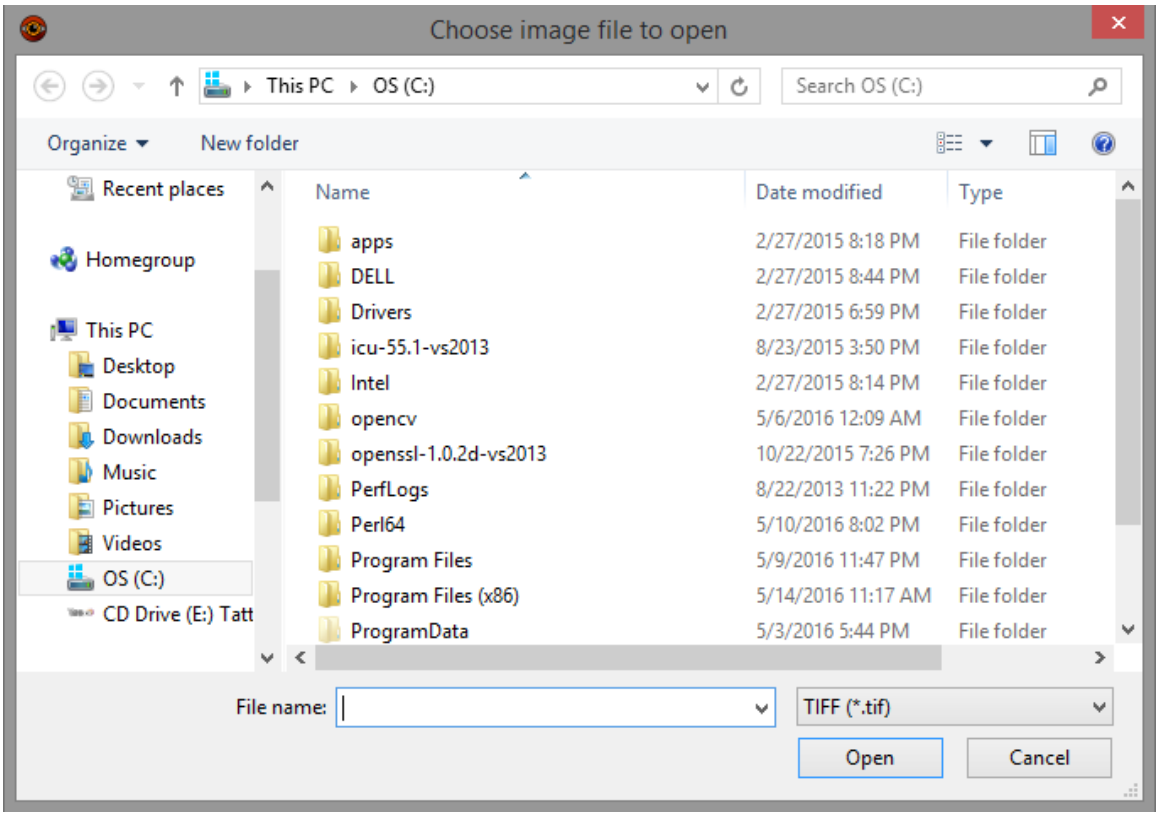

Figure 37: Browse dialog box, DRDT(release)

After selecting an image using the *Browse* button, the image path is displayed into the textbox inside the *File selector* section. Figure 38 illustrates the interface when an image is selected.

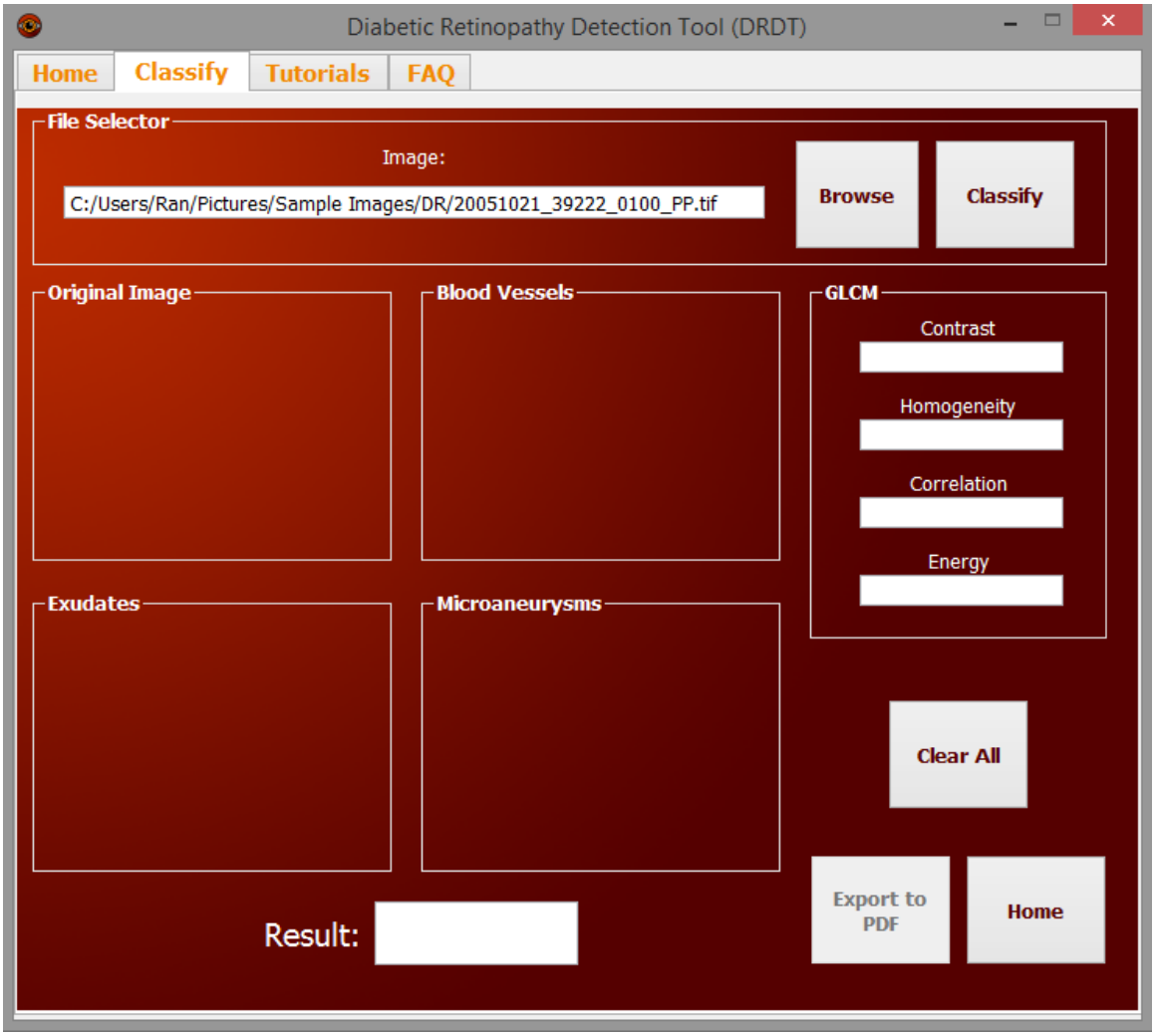

Figure 38: User selected an image, DRDT(release)

Afterwards, the *Classify* button becomes clickable. The user should click the *Classify* button to start classifying the image. This process may take a while especially if the image is big. Figure 39 illustrates this.

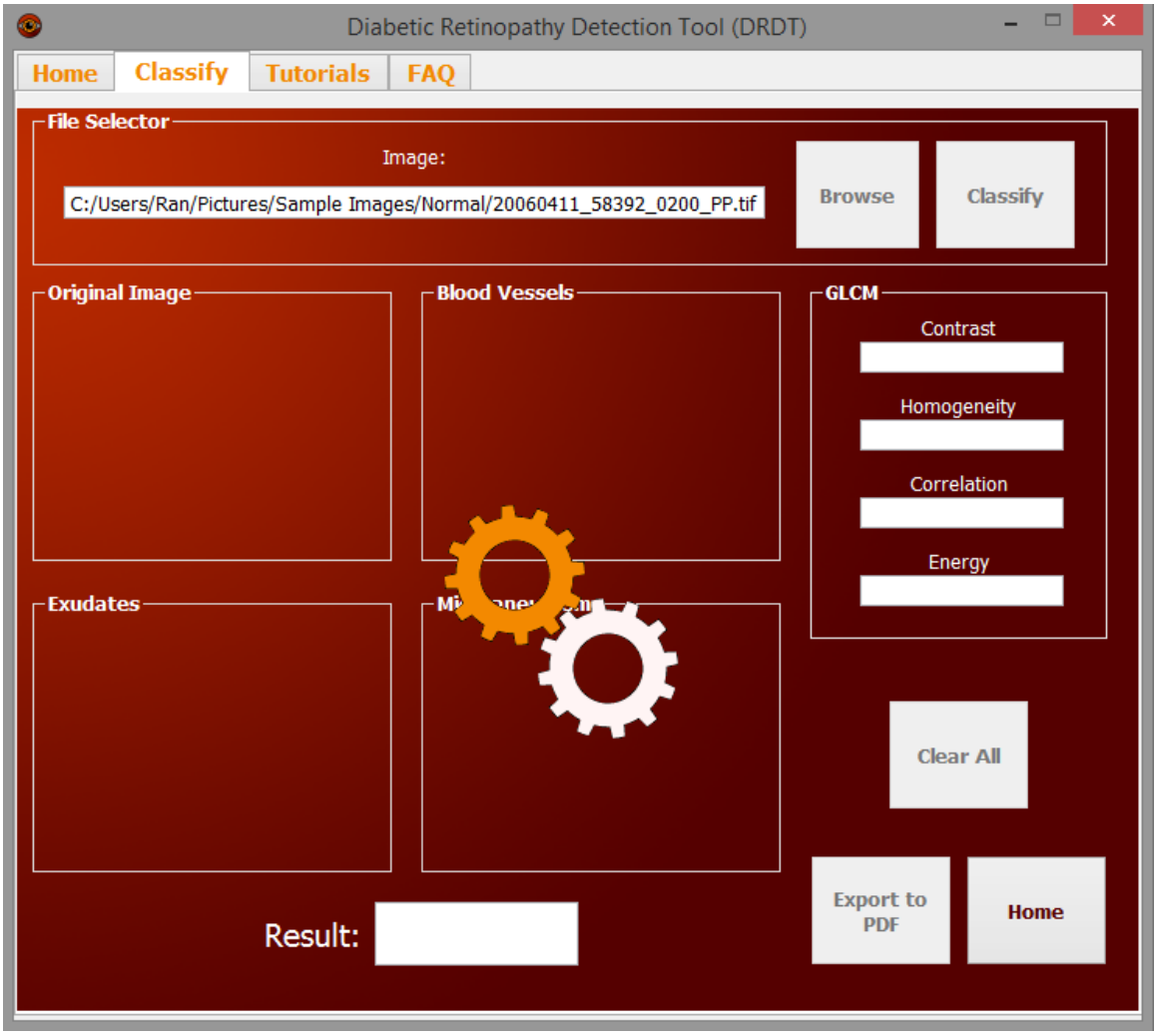

Figure 39: User clicked the *Classify* button, DRDT(release)

The results of the process done are displayed in fields and sections in the *Classify* tab. These are the images section, which displays the original image, segmented blood vessels, segmented exudates, and segmented microaneurysms, the GLCM section, which displays the contrast, homogeneity, correlation, and energy of the image, and lastly, the main result box, which displays the classification of the image. There are only two possible values that could be displayed here, which are *DR* and *Normal*. Figure 40 shows an example when the result is *DR* (the eye has diabetic retinopathy).

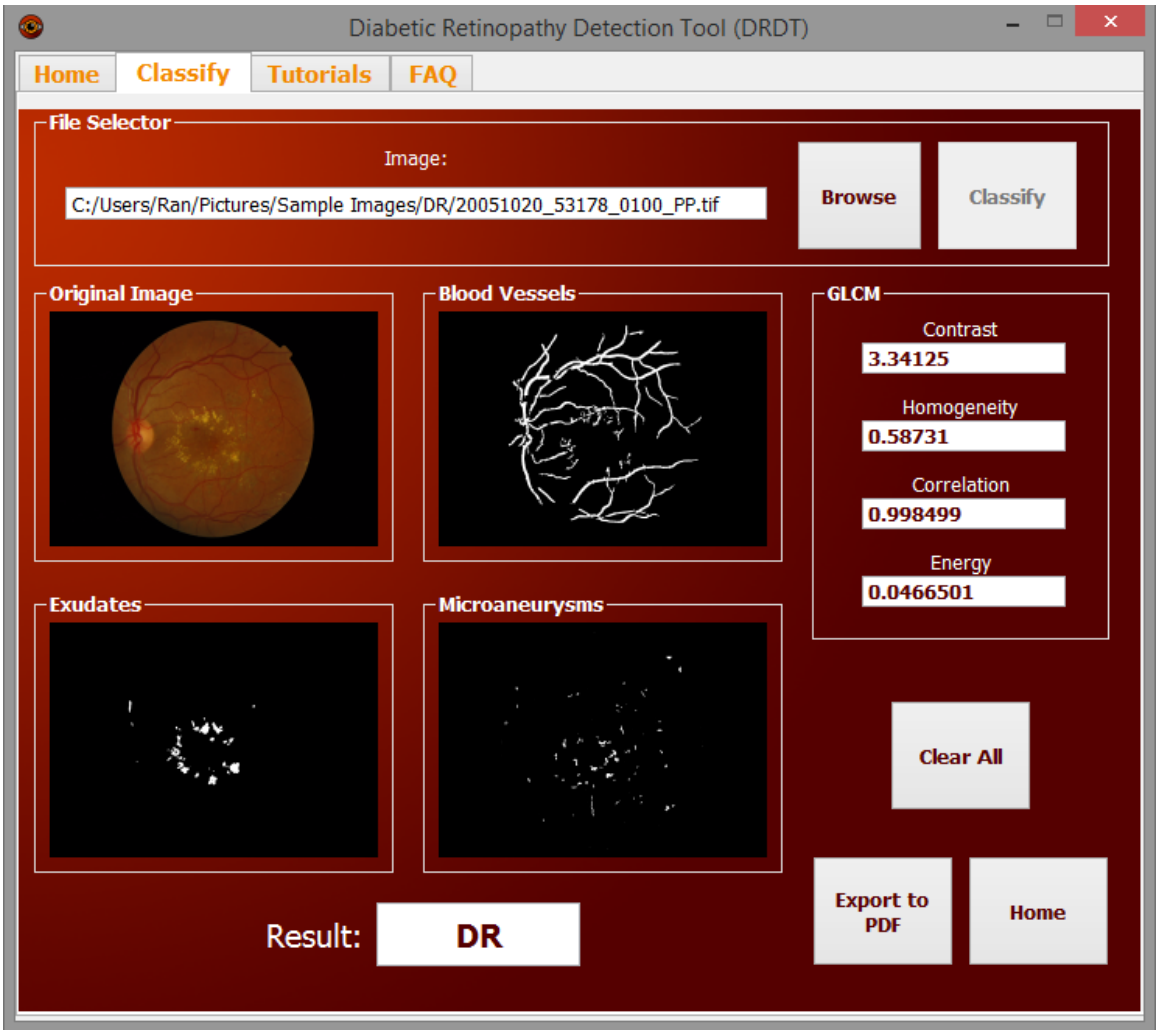

Figure 40: DR-positive result sample, DRDT(release)

Figure 41 on the other hand, shows a sample when the result of the detection is *Normal* (the eye does not have diabetic retinopathy).

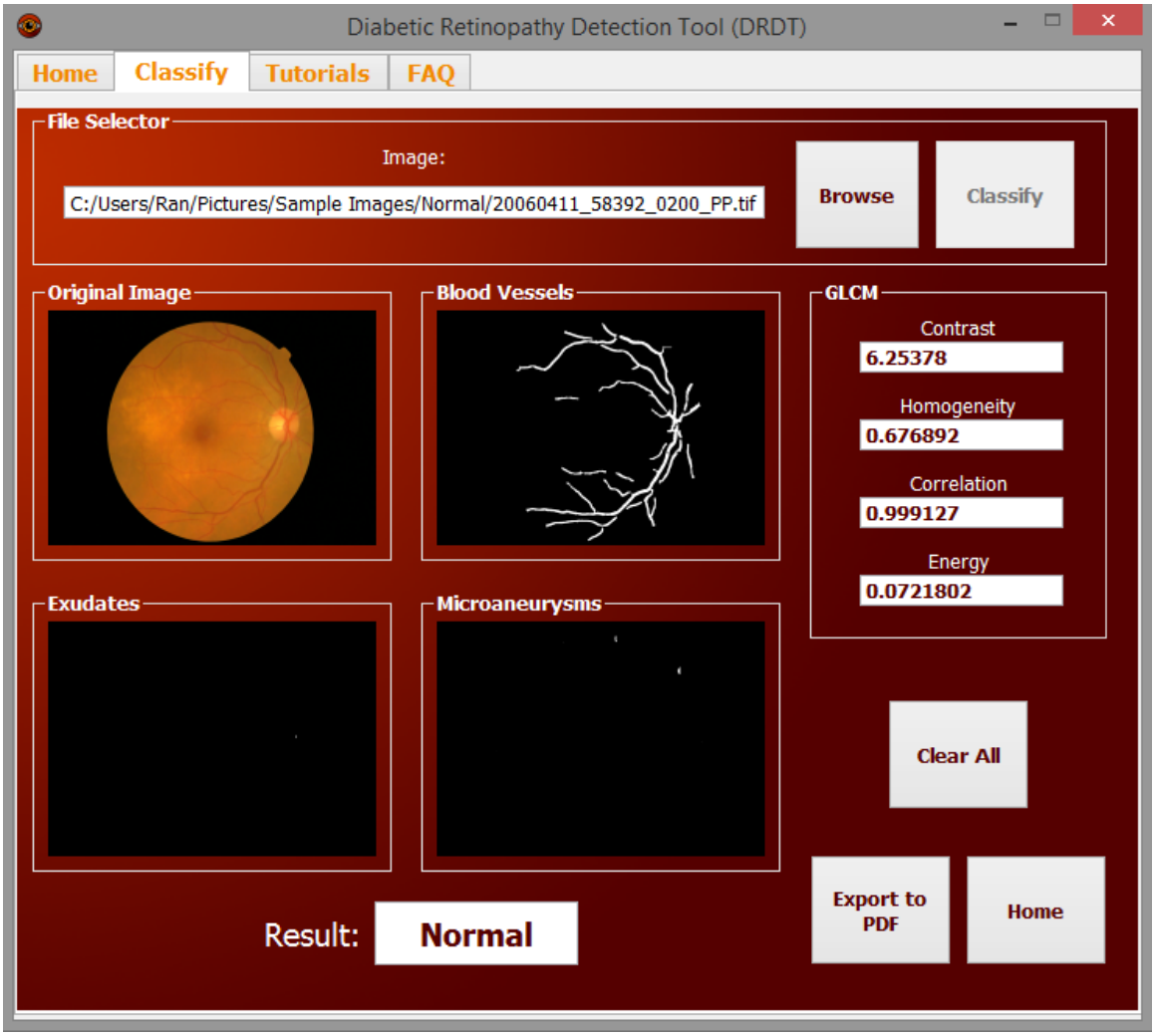

Figure 41: Normal result sample, DRDT(release)

There are other things that could be done in this tab. The user may clear everything on the tab by clicking the *Clear All* button. The user may also opt to save the results of the classification for future use. The *Export to PDF* button provides this functionality. The user may input the path where he/she wants to save the PDF file. Figure 42 shows the dialog box that is opened when you click the *Export to PDF* button.

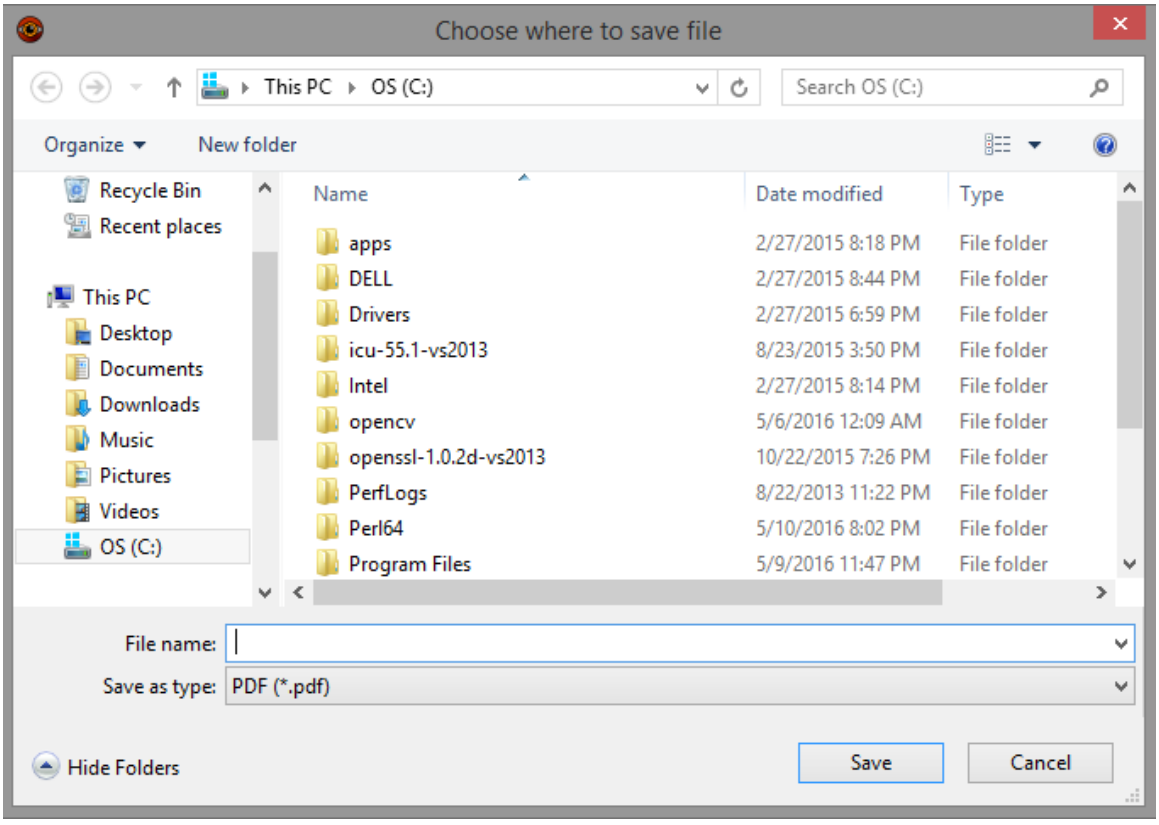

Figure 42: Export dialog box, DRDT(release)

The exported PDF will contain all the details displayed previously on the *Classify* tab: location of the image, original image, blood vessels image, exudates image, microaneurysms image, GLCM descriptors contrast, homogeneity, correlation and energy, some of the parameters used in the training, and the detection result which is *DR* or *Normal*. Figure 43 shows a sample PDF on the user's file system.

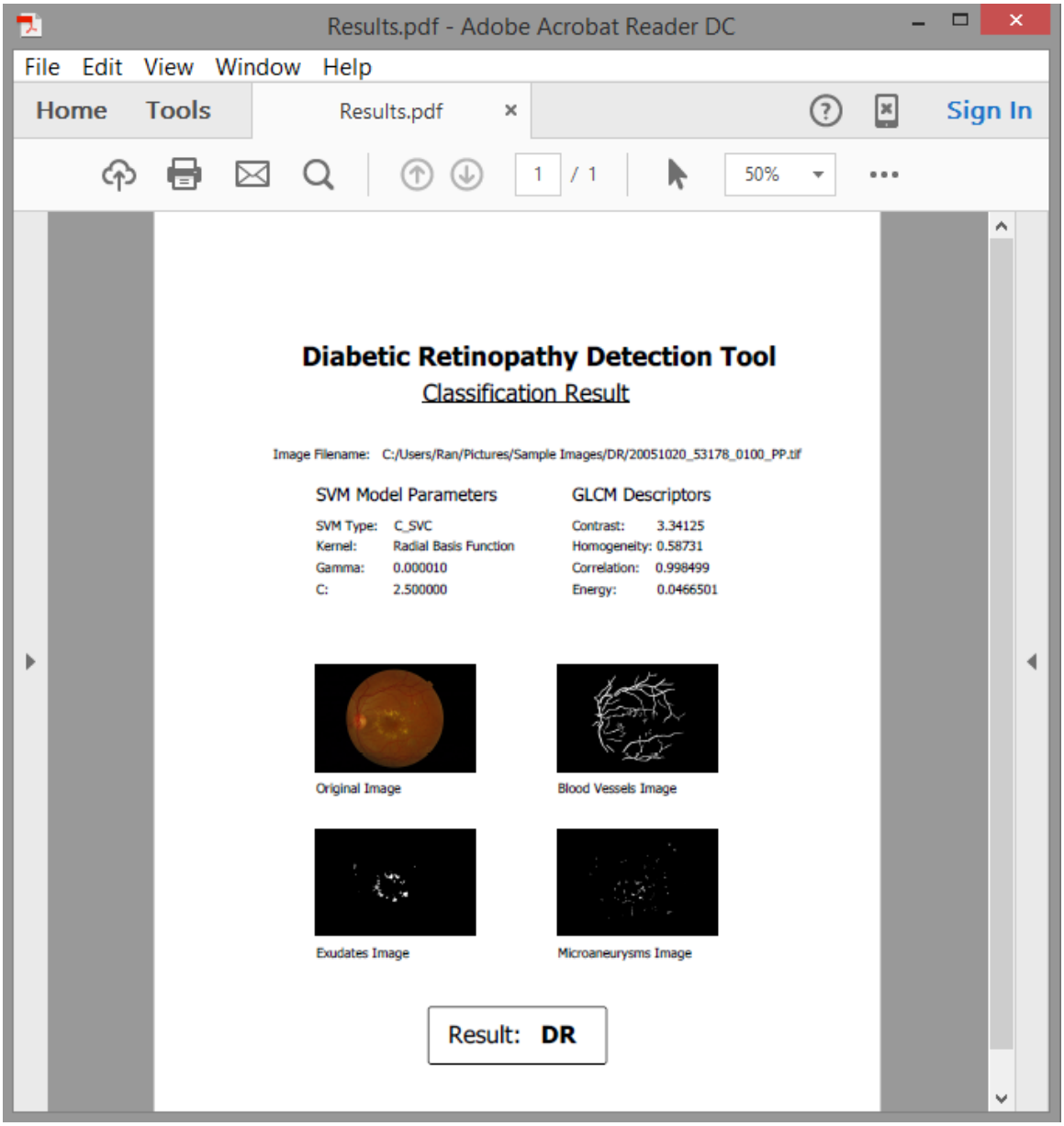

Figure 43: Exported PDF sample, DRDT(release)

The *Tutorials* tab provides learning materials for the user of DRDT. It is accessible by either clicking *Watch Tutorials* button in the *Home* tab or clicking the *Tutorials* tab itself. The user can select from the videos he/she wants to view from the left side of the tab. The videos include an animation illustrating diabetic retinopathy by NIH, a more comprehensive overview of diabetic retinopathy by Khan Academy and an easy lecture on how SVM algorithm works by Korting. There are also additional

materials the user can read by clicking the link on the bottom side of the tab. Figure 44 shows the *Tutorials* tab.

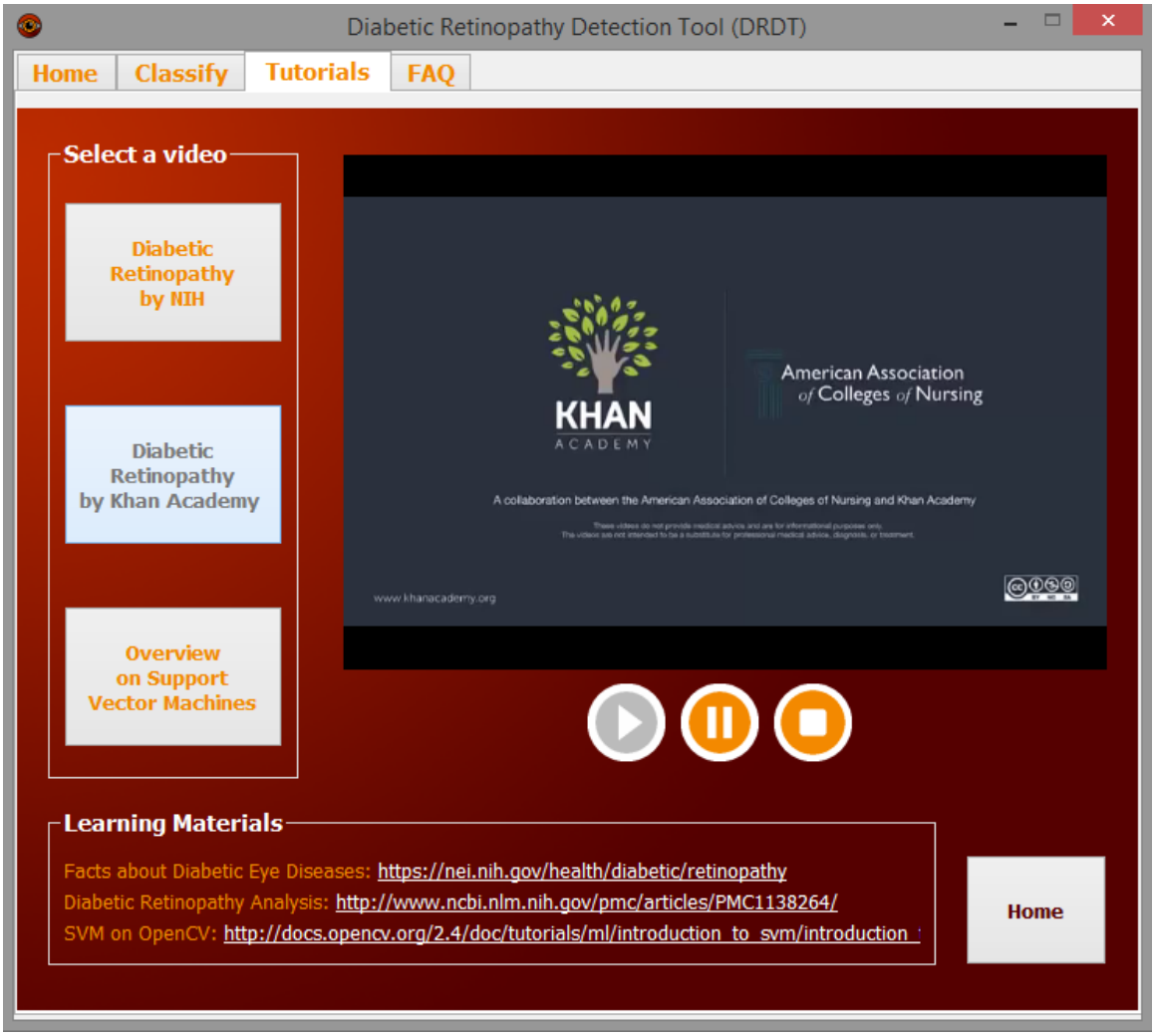

Figure 44: *Tutorials* Tab, DRDT(release)

DRDT also provides a guide for users of the system: the *FAQ* tab. New users would find this tab useful specially if they don't have a clue on how to use DRDT. It can be accessed by clicking the *FAQ* button on the Home tab or clicking the *FAQ* tab itself located on the top of the interface of the tool.

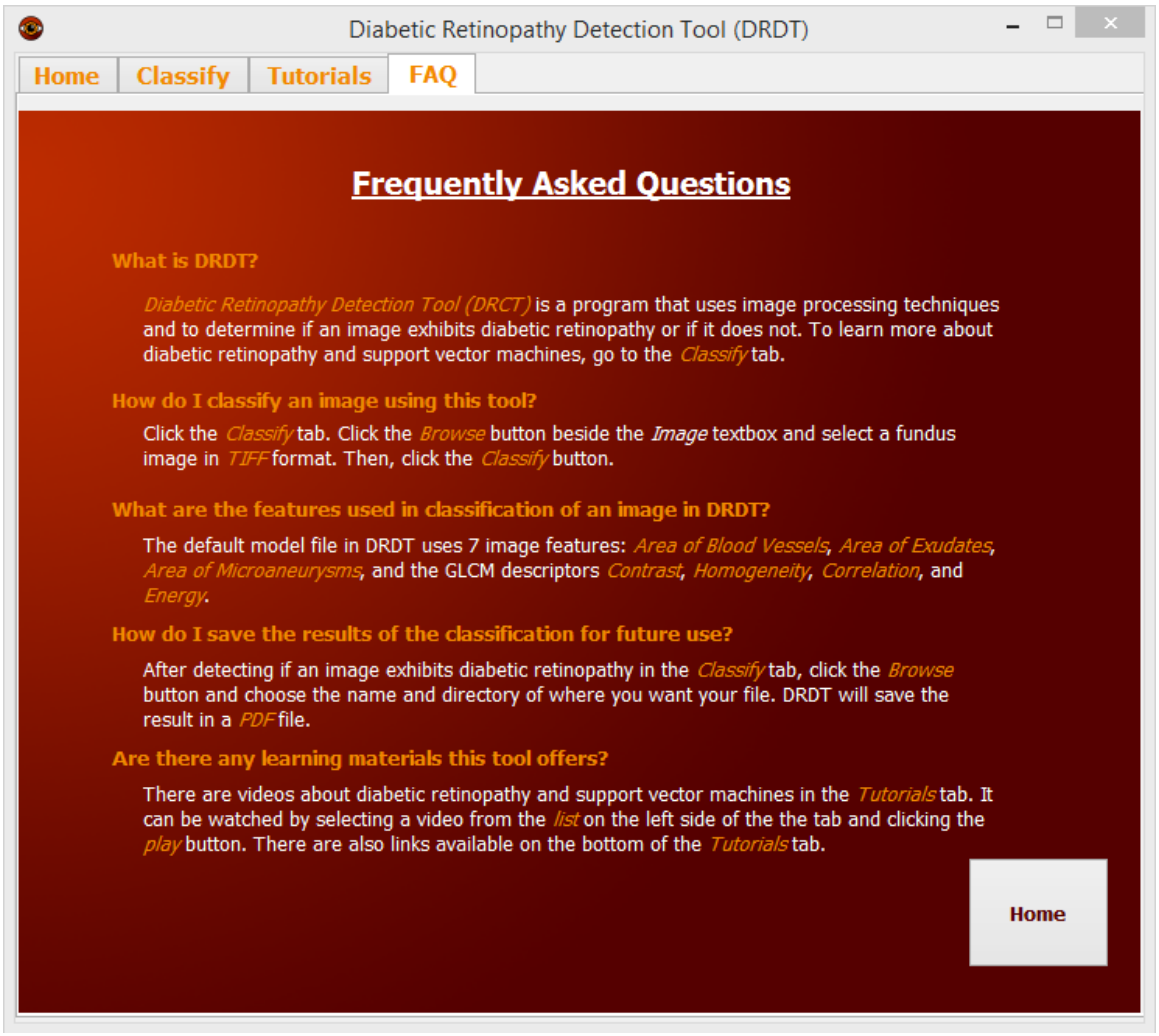

Figure 45: *FAQ* Tab, DRDT(release)

Finally, when the user is done using the tool, there is a *Quit* button on the *Home* tab the user can click to exit the application.

VI. Discussions

DRDT is an application used for automatically detecting the presence of diabetic retinopathy in fundus images. The machine learning model used by the tool for classification is **SVM**. There are only a few images used in the study and SVM works well even with few training examples. SVM also allows the use of kernel functions. This lets the user to apply a classifier to data that have no obvious fixed-dimensional vector space representation [44]. Additionally, the performances of SVM in a wide range of problems are consistently very good when compared to a variety of methods including decision tree based algorithms, statistical algorithms, and instance based learning methods [45].

The SVM type used in the training is **C_SVC** while the kernel used is **Radial Basis Function (RBF)**. The training also involved the use of **train_auto**, a function in OpenCV, which trains the SVM model automatically by choosing the optimal parameters **C**, **gamma**, **p**, **nu**, **coef0**, and **degree** [46].

The study aims to provide a tool for medical students/trainees to refine and train their ability to detect DR and a second opinion to physicians who experience cases where the presence of DR is hard to detect. It is non-proprietary, so using it for research, training, or other medical purposes is free and would not require anything other than the tool itself and the images that would be used.

However, the most important factor to consider in using decision support tools is the correctness in its decisions. In DRDT's case, it is the accuracy in classifying the fundus images as DR-positive or DR-negative (normal). The current version of DRDT has an average accuracy of **66.6667%** during its pre-release phase. This may be caused by some factors in the design of processing of images used by the developer. In order for this tool to be able to achieve a higher accuracy in its detection,

improvements must be made.

Since the paper (Selvathi's work) this was originally based on [15] used proprietary tools to carry out some of the algorithms needed, the researcher used alternatives and based some parts of DRDT on other parts of researches such as Sopharak's [14], Leandro's [17] and Mudassar's [18] which can be done using open-source tools. One possible cause of the low accuracy could be the parameters used in the image processing methods. Some parameters of image processing techniques based on the papers above are not specified. Another possible cause of the accuracy may also be attributed in the images used in the study. The images in MESSIDOR varies widely in terms of illumination and several images have artifacts and optical aberrations, and that might have prevented the tool to achieve a high accuracy.

VII. Conclusions

Diabetic Retinopathy Detection Tool is an application developed to automatically detect diabetic retinopathy in fundus images using image processing techniques and SVM algorithm. The features used for the detection are **areas of blood vessels**, **exudates**, and **microaneurysms**, and **GLCM descriptors** (**contrast**, **homogeneity**, **correlation** and **energy**), which are all extracted from the input image.

DRDT can serve as a learning tool for its users since it provides them with learning materials. These materials cover introduction to both diabetic retinopathy and support vector machine. Aside from these materials, DRDT can be used to familiarize medical trainees/students in identifying DR with just a few clicks. Using the DRDT for research purposes is also free of charge.

Since the accuracy of the DRDT is not that high yet, it still needs to be improved before it can be classified as a reliable decision support tool and as a tool that is able to provide ophthalmologists a second opinion on DR cases.

However, this study could be used as a basis for a better decision support tool for diabetic retinopathy. The design and algorithms used to develop DRDT could be further refined to provide a more reliable and accurate tool, enabling the DR diagnosis to be closer to having a fast and automated process.

VIII. Recommendations

The Diabetic Retinopathy Detection Tool (DRDT), as it is now, could still be improved significantly. For instance, a feature that enables users to choose the training parameters could be added into the tool to allow users to configure and train the classifier themselves when the need arises.

Another feature that could be added into DRDT is the multiple file selection. DRDT currently only allows the detection of DR on a single image. Because some of the users may want to do it in bulk, there should be an option to allow the detection of DR in several images. Additionally, the tool should still be able to export the results of all the images in a PDF file/s.

Since the DRDT is only about *67%* accurate, the classifier's accuracy needs to be increased. The following can be done in order to improve the DRDT classifier's accuracy:

1. Polish the algorithms used to process the images to improve detection of retinal structures.

2. Apply additional pre-processing techniques into the images to correct image variability such as differences in illumination.

3. Explore other features which may be extracted from the fundus images.

4. Experiment with the features to be used to train the classifier.

5. Try other datasets such as the Kaggle dataset. *Kaggle* (https://www.kaggle.com) is a community hosts competitions involving machine learning. It also allows access to datasets of different medical images which includes diabetic retinopathy dataset.

6. Explore other machine learning algorithms and incorporate them together into a hybrid system.

IX. Bibliography

[1] "Diabetes programme." http://www.who.int/diabetes/en/. Accessed: March 18, 2015.

[2] S. Mendis, "Global status report on noncommunicable diseases 2014," tech. rep., World Health Organization, Geneva, Switzerland, 2014.

[3] C. D. Mathers and D. Loncar, "Projections of global mortality and burden of disease from 2002 to 2030," *PLoS Medicine*, vol. 3, November 2006.

[4] "Diabetes programme: About diabetes." http://www.who.int/diabetes/action_online/basics/en/index3.html. Accessed: March 18, 2015.

[5] World Health Organization, *Prevention of blindness from diabetes mellitus*, (Geneva, Switzerland), WHO Press, November 2005.

[6] R. Priya and P. Aruna, "Diagnosis of diabetic retinopathy using machine learning techniques," *Journal on Soft computing*, vol. 3, pp. 563–575, July 2013.

[7] M. García, C. I. Sánchez, M. I. Lopez, D. Abásolo, and R. Hornero, "Neural network based detection of hard exudates in retinal images," *Computer Methods and Programs in Biomedicine*, vol. 93, pp. 9–19, January 2009.

[8] W. S. Noble, "What is a support vector machine?," *Nature Biotechnology*, vol. 24, pp. 1565–1567, December 2006.

[9] S. Sivakumar and C. Chandrasekar, "Lung nodule detection using fuzzy clustering and support vector machines," *International Journal of Engineering and Technology*, vol. 5, pp. 179–185, February 2013.

[10] C. W. Hsu, C. C. Chang, and C. J. Lin, *A Practical Guide to Support Vector Classification*. Taipei, Taiwan, April 2010. Accessed: December 19, 2014.

[11] E. Decenciére, X. Zhang, G. Cazaguel, B. Lay, B. Cochener, C. Trone, P. Gain, R. Ordonez, P. Massin, A. Erginay, B. Charton, and J. C. Klein, "Feedback on a publicly distributed image database: The messidor database," *Image Analysis & Stereology*, vol. 33, pp. 231–234, August 2014.

[12] M. U. Akram, S. Khalid, and S. A. Khan, "Identification and classification of microaneurysms for early detection of diabetic retinopathy," *Pattern Recognition*, vol. 46, pp. 107–116, January 2013.

[13] D. Usher, M. Dumskyj, M. Himaga, T. H. Williamson, S. Nussey, and J. Boyce, "Automated detection of diabetic retinopathy in digital retinal images: a tool for diabetic retinopathy screening," *Diabetic Medicine*, vol. 21, pp. 84–90, January 2004.

[14] A. Sopharak, B. Uyyanovara, S. Barman, and T. H. Williamson, "Automatic detection of diabetic retinopathy exudates from non-dilated retinal images using mathematical morphology methods," *Computerized Medical Images and Graphics*, vol. 32, pp. 720–727, December 2008.

[15] D. Selvathi, N. B. Prakash, and N. Balagopal, "Automated detection of diabetic retinopathy for early diagnosis using feature extraction and support vector machine," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, pp. 762–767, November 2012.

[16] M. García, M. I. López, D. Álvarez, and R. Hornero, "Assessment of four neural network based classifiers to automatically detect red lesion in retinal images," *Medical Engineering and Physics*, vol. 32, pp. 1085–1093, December 2010.

[17] J. J. G. Leandro, R. J. M. Cesar, and H. F. Jelinek, "Blood vessels segmentation in retina: Preliminary assessment of the mathematical morphology and of the wavelet transform techniques," *Computer Graphics and Image Processing, 2001 Proceedings of XIV Brazilian Symposium on*, pp. 84–90, October 2001.

[18] A. A. Mudassar and S. Butt, "Extraction of blood vessels in retinal images using four different techniques," *Journal of Medical Engineering*, pp. 1–21, November 2013.

[19] K. Noronha, U. R. Acharya, K. P. Nayak, S. Kamath, and S. V. Bhandary, "Decision support system for diabetic retinopathy using discrete wavelet transform," *Journal of Engineering in Medicine*, vol. 227, pp. 251–261, December 2012.

[20] Y. Sahin and E. Duman, "Detecting credit card fraud by decision trees and support vector machines," *International Multi-Conference of Engineers and Computer Scientists*, vol. 1, March 2011.

[21] S. Munir, V. Gupta, S. Nemade, and M. Z. Alam, "Face recognition using support vector machines," *International Journal on Emerging Technologies*, vol. 2, pp. 67–70, February 2011.

[22] Y. Pan, P. Shen, and L. Shen, "Speech emotion recognition using support vector machine," *International Journal of Smart Home*, vol. 5, pp. 101–108, April 2012.

[23] H. L. Chen, B. Yang, J. Liu, and D. Y. Liu, "A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis," *Expert Systems with Applications*, vol. 38, pp. 9014–9022, July 2011.

[24] J. Virmani, V. Kumar, N. Kalra, and N. Khandelwal, "Svm-based characterization of liver ultrasound images using wavelet packet texture descriptors," *Journal of Digital Imaging*, vol. 26, pp. 530–543, June 2013.

[25] A. Kharrat, K. Gasmi, M. B. Messaoud, and N. B. ad Mohamed Abid, "A hybrid approach for automatic classification of brain mri using genetic algorithm and support vector machine," *Leonardo Journal of Sciences*, vol. 17, pp. 71–82, July 2010.

[26] "Facts about diabetic eye disease." https://nei.nih.gov/health/diabetic/retinopathy. Accessed: May 9, 2015.

[27] V. Beal, "Decision support system - dss." http://www.webopedia.com/TERM/D/decision_support_system.html. Accessed: May 9, 2015.

[28] "Digital image processing." http://nptel.ac.in/courses/117104069/chapter_8/8_16.html. Accessed: May 9, 2015.

[29] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Median filter." http://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm, 2003. Accessed: May 9, 2015.

[30] S. Philip, "Adaptive histogram equalization." http://www.cs.utah.edu/~sujin/courses/reports/cs6640/project2/ahe. Accessed: May 13, 2015.

[31] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. T. H. Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355–368, 1987.

[32] S. Philip, "Contrast limited adaptive histogram equalization." http://www.cs.utah.edu/~sujin/courses/reports/cs6640/project2/clahe.html. Accessed: June 10, 2015.

[33] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Morphology." http://

homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm, 2003. Accessed: May 13, 2015.

[34] H. Rhody, "Lecture 3: Basic morphological image processing." https://www.cis.rit.edu/class/simg782/lectures/lecture_03/lec782_05_03.pdf, September 2005. Accessed: May 13, 2015.

[35] "Morphological operators." http://www.coe.utah.edu/~cs4640/slides/Lecture11.pdf, February 2012. Accessed: May 13, 2015.

[36] G. T. Schuster, "Watershed transform (top-hat and bottom-hat filters)." http://utam.gg.utah.edu/tomo03/03_mid/HTML/node120.html. Accessed: September 23, 2015.

[37] D. A. Greensted, "Otsu thresholding." http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html. Accessed: January 7, 2016.

[38] N. Otsu, "A threshold selection method from gray-level," *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-9, pp. 62–66, January 1979.

[39] "Documentation of gray-level co-occurence matrix (glcm)." http://www.mathworks.com/help/images/gray-level-co-occurrence-matrixglcm.html. Accessed: May 17, 2015.

[40] D. Gadkari, *Image Quality Analysis Using GLCM*. Doctoral dissertation, University of Central Florida Orlando, Florida, 2004.

[41] T. Fletcher, "Support vector machines explained." www.cs.ucl.ac.uk/staff/T.Fletcher/, 2009. Accessed: May 18, 2015.

[42] K. S. Durgesh and B. Lekha, "Data classification using support vector machine," *Journal of Theoretical and Applied Information Technology*, vol. 12, no. 1, pp. 1–7,

2010.

[43] "Adcis download third party: Messidor database." http://www.adcis.net/en/Download-Third-Party/Messidor.html. Accessed: May 19, 2015.

[44] A. Ben-Hur and J. Weston, *A User's Guid to Support Vector Machines*. USA. Accessed: May 25, 2016.

[45] T. V. Gestel, J. A. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. D. Moor, and J. Vandewalle, "Benchmarking least squares support vector machine classifiers," *Machine Learning*, vol. 54, pp. 5–32, January 2004.

[46] OpenCV Development Team, *Support Vector Machines*, 2011-2014. Accessed: May 25, 2016.

X. Appendix

A. Glossary

(a) *blood vessels*	a tubular structure which carries blood through the tissues and organs
(b) *diabetes*	a group of metabolic disease in which there are high blood sugar levels over a prolonged period
(c) *exudate*	a mass of cells and fluid that has seeped out of blood vessels or an organ, especially in inflammation
(d) *image processing*	the analysis and manipulation of a digitized image to improve the images' quality
(e) *machine learning*	a field in computer science which deals with algorithms learning from and making predictions on data
(f) *microaneurysm*	a tiny aneurysm or swelling in the side of the blood vessel
(g) *ophthalmologist*	a medical doctor who specializes in eye and vision care
(h) *optic disc*	a circular area in the back of the eye where the nerves in the retina connect to form the optic nerve
(i) *retina*	a light-sensitive tissue lining the back of the eye where an image of the visual world is created

B. Source Code

```
/********************************************************************************
** Form generated from reading UI file 'drdt.ui'
**
** Created by: Qt User Interface Compiler version 5.5.0
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
********************************************************************************/

#ifndef UI_DRDT_H
#define UI_DRDT_H

#include <QtCore/QVariant>
#include <QtMultimediaWidgets\qvideowidget.h>
#include <QtWidgets/QAction>
#include <QtWidgets/QApplication>
```

```
#include <QtWidgets/QButtonGroup>
#include <QtWidgets/QCommandLinkButton>
#include <QtWidgets/QGroupBox>
#include <QtWidgets/QHeaderView>
#include <QtWidgets/QLabel>
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QTabWidget>
#include <QtWidgets/QWidget>

QT_BEGIN_NAMESPACE

class Ui_DRDTClass
{
public:
    QWidget *centralWidget;
    QTabWidget *tabWidget;
    QWidget *homeTab;
    QPushButton *startClassificationButton;
    QPushButton *quitButton;
    QLabel *backgroundLabel;
    QPushButton *watchTutorialsButton;
    QPushButton *helpButton;
    QWidget *classifyTab;
    QGroupBox *fileSelectGroup;
    QLabel *imageLabel;
    QLineEdit *imagePath;
    QPushButton *imageButton;
    QPushButton *classifyButton;
    QGroupBox *fundusImageGroup;
    QLabel *fundusImageLabel;
    QGroupBox *bvImageGroup;
    QLabel *bvImageLabel;
    QGroupBox *exuImageGroup;
    QLabel *exuImageLabel;
    QGroupBox *maImageGroup;
    QLabel *maImageLabel;
    QLabel *resultLabel;
    QLineEdit *resultText;
    QPushButton *exportButton;
    QPushButton *homeButton1;
    QLabel *movieLabel;
    QGroupBox *glcmGroup;
    QLabel *contrastLabel;
    QLineEdit *contrastValue;
    QLabel *homogeneityLabel;
    QLineEdit *homogeneityValue;
    QLabel *correlationLabel;
    QLineEdit *correlationValue;
    QLabel *energyLabel;
    QLineEdit *energyValue;
    QLabel *backgroundLabel2;
    QPushButton *clearButton;
    QWidget *tutorialsTab;
    QPushButton *homeButton2;
    QLabel *backgroundLabel3;
    QGroupBox *learningMaterialsGroup;
    QLabel *learningMaterial3;
    QLabel *learningMaterial1_2;
    QLabel *learningMaterial2;
    QVideoWidget *videoWidget;
    QCommandLinkButton *playButton;
    QGroupBox *groupBox;
    QPushButton *drPlayButton;
    QPushButton *dr2PlayButton;
    QPushButton *svmPlayButton;
    QCommandLinkButton *pauseButton;
    QCommandLinkButton *stopButton;
    QWidget *helpTab;
    QPushButton *homeButton3;
    QLabel *backgroundLabel4;
    QLabel *faqLabel;
    QLabel *question1Label;
    QLabel *question2Label;
    QLabel *question3Label;
    QLabel *question4Label;
    QLabel *answer1Label;
    QLabel *answer2Label;
    QLabel *answer3Label;
    QLabel *answer4Label;
    QLabel *question5Label;
    QLabel *answer5Label;

    void setupUi(QMainWindow *DRDTClass)
    {
        if (DRDTClass->objectName().isEmpty())
            DRDTClass->setObjectName(QStringLiteral("DRDTClass"));
        DRDTClass->resize(725, 623);
        QFont font;
        font.setPointSize(10);
        DRDTClass->setFont(font);
```

```
QIcon icon;
icon.addFile(QStringLiteral(":/images/logo.png"), QSize(), QIcon::Normal, QIcon::Off);
DRDTClass->setWindowIcon(icon);
DRDTClass->setWindowOpacity(1);
DRDTClass->setLayoutDirection(Qt::LeftToRight);
DRDTClass->setAutoFillBackground(true);
DRDTClass->setIconSize(QSize(50, 50));
centralWidget = new QWidget(DRDTClass);
centralWidget->setObjectName(QStringLiteral("centralWidget"));
centralWidget->setAutoFillBackground(true);
tabWidget = new QTabWidget(centralWidget);
tabWidget->setObjectName(QStringLiteral("tabWidget"));
tabWidget->setGeometry(QRect(0, 0, 731, 621));
QPalette palette;
QBrush brush(QColor(243, 137, 0, 255));
brush.setStyle(Qt::SolidPattern);
palette.setBrush(QPalette::Active, QPalette::WindowText, brush);
QBrush brush1(QColor(85, 0, 0, 255));
brush1.setStyle(Qt::SolidPattern);
palette.setBrush(QPalette::Active, QPalette::Text, brush1);
palette.setBrush(QPalette::Inactive, QPalette::WindowText, brush);
palette.setBrush(QPalette::Inactive, QPalette::Text, brush1);
QBrush brush2(QColor(120, 120, 120, 255));
brush2.setStyle(Qt::SolidPattern);
palette.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
palette.setBrush(QPalette::Disabled, QPalette::Text, brush2);
tabWidget->setPalette(palette);
QFont font1;
font1.setPointSize(11);
font1.setBold(true);
font1.setWeight(75);
tabWidget->setFont(font1);
tabWidget->setAutoFillBackground(false);
tabWidget->setTabShape(QTabWidget::Rounded);
tabWidget->setIconSize(QSize(16, 26));
tabWidget->setElideMode(Qt::ElideNone);
homeTab = new QWidget();
homeTab->setObjectName(QStringLiteral("homeTab"));
homeTab->setAutoFillBackground(true);
startClassificationButton = new QPushButton(homeTab);
startClassificationButton->setObjectName(QStringLiteral("startClassificationButton"));
startClassificationButton->setGeometry(QRect(240, 260, 241, 71));
QPalette palette1;
QBrush brush3(QColor(0, 0, 0, 255));
brush3.setStyle(Qt::SolidPattern);
palette1.setBrush(QPalette::Active, QPalette::WindowText, brush3);
QBrush brush4(QColor(240, 240, 240, 255));
brush4.setStyle(Qt::SolidPattern);
palette1.setBrush(QPalette::Active, QPalette::Button, brush4);
palette1.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette1.setBrush(QPalette::Inactive, QPalette::WindowText, brush3);
palette1.setBrush(QPalette::Inactive, QPalette::Button, brush4);
palette1.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette1.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
palette1.setBrush(QPalette::Disabled, QPalette::Button, brush4);
palette1.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
startClassificationButton->setPalette(palette1);
QFont font2;
font2.setPointSize(15);
font2.setBold(true);
font2.setWeight(75);
startClassificationButton->setFont(font2);
startClassificationButton->setCursor(QCursor(Qt::PointingHandCursor));
startClassificationButton->setAutoFillBackground(true);
startClassificationButton->setFlat(false);
quitButton = new QPushButton(homeTab);
quitButton->setObjectName(QStringLiteral("quitButton"));
quitButton->setGeometry(QRect(240, 440, 241, 41));
QPalette palette2;
palette2.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette2.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette2.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
quitButton->setPalette(palette2);
quitButton->setFont(font2);
quitButton->setCursor(QCursor(Qt::PointingHandCursor));
quitButton->setAutoFillBackground(true);
quitButton->setFlat(false);
backgroundLabel = new QLabel(homeTab);
backgroundLabel->setObjectName(QStringLiteral("backgroundLabel"));
backgroundLabel->setGeometry(QRect(0, 10, 721, 581));
QSizePolicy sizePolicy(QSizePolicy::Maximum, QSizePolicy::Maximum);
sizePolicy.setHorizontalStretch(0);
sizePolicy.setVerticalStretch(0);
sizePolicy.setHeightForWidth(backgroundLabel->sizePolicy().hasHeightForWidth());
backgroundLabel->setSizePolicy(sizePolicy);
backgroundLabel->setMaximumSize(QSize(721, 581));
backgroundLabel->setPixmap(QPixmap(QString::fromUtf8(":/images/homeBackground.png")));
backgroundLabel->setScaledContents(true);
watchTutorialsButton = new QPushButton(homeTab);
watchTutorialsButton->setObjectName(QStringLiteral("watchTutorialsButton"));
watchTutorialsButton->setGeometry(QRect(240, 340, 241, 41));
QPalette palette3;
```

```
palette3.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette3.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette3.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
watchTutorialsButton->setPalette(palette3);
watchTutorialsButton->setFont(font2);
watchTutorialsButton->setCursor(QCursor(Qt::PointingHandCursor));
watchTutorialsButton->setAutoFillBackground(true);
watchTutorialsButton->setFlat(false);
helpButton = new QPushButton(homeTab);
helpButton->setObjectName(QStringLiteral("helpButton"));
helpButton->setGeometry(QRect(240, 390, 241, 41));
QPalette palette4;
palette4.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette4.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette4.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
helpButton->setPalette(palette4);
helpButton->setFont(font2);
helpButton->setCursor(QCursor(Qt::PointingHandCursor));
helpButton->setAutoFillBackground(true);
helpButton->setFlat(false);
tabWidget->addTab(homeTab, QString());
backgroundLabel->raise();
startClassificationButton->raise();
quitButton->raise();
watchTutorialsButton->raise();
helpButton->raise();
classifyTab = new QWidget();
classifyTab->setObjectName(QStringLiteral("classifyTab"));
classifyTab->setFont(font);
classifyTab->setAutoFillBackground(true);
fileSelectGroup = new QGroupBox(classifyTab);
fileSelectGroup->setObjectName(QStringLiteral("fileSelectGroup"));
fileSelectGroup->setGeometry(QRect(10, 10, 691, 101));
QPalette palette5;
QBrush brush5(QColor(255, 255, 255, 255));
brush5.setStyle(Qt::SolidPattern);
palette5.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette5.setBrush(QPalette::Active, QPalette::Text, brush3);
palette5.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette5.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette5.setBrush(QPalette::Inactive, QPalette::Text, brush3);
palette5.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette5.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
palette5.setBrush(QPalette::Disabled, QPalette::Text, brush2);
palette5.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
fileSelectGroup->setPalette(palette5);
fileSelectGroup->setAutoFillBackground(false);
imageLabel = new QLabel(fileSelectGroup);
imageLabel->setObjectName(QStringLiteral("imageLabel"));
imageLabel->setGeometry(QRect(210, 20, 71, 21));
QFont font3;
font3.setPointSize(10);
font3.setBold(false);
font3.setWeight(50);
imageLabel->setFont(font3);
imageLabel->setAlignment(Qt::AlignCenter);
imagePath = new QLineEdit(fileSelectGroup);
imagePath->setObjectName(QStringLiteral("imagePath"));
imagePath->setGeometry(QRect(20, 50, 451, 21));
QFont font4;
font4.setBold(false);
font4.setWeight(50);
imagePath->setFont(font4);
imagePath->setReadOnly(true);
imageButton = new QPushButton(fileSelectGroup);
imageButton->setObjectName(QStringLiteral("imageButton"));
imageButton->setGeometry(QRect(490, 20, 81, 71));
QPalette palette6;
palette6.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette6.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette6.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
imageButton->setPalette(palette6);
QFont font5;
font5.setPointSize(10);
font5.setBold(true);
font5.setWeight(75);
imageButton->setFont(font5);
imageButton->setCursor(QCursor(Qt::PointingHandCursor));
imageButton->setToolTipDuration(-1);
classifyButton = new QPushButton(fileSelectGroup);
classifyButton->setObjectName(QStringLiteral("classifyButton"));
classifyButton->setGeometry(QRect(580, 20, 91, 71));
QPalette palette7;
palette7.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette7.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette7.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
classifyButton->setPalette(palette7);
classifyButton->setFont(font5);
classifyButton->setCursor(QCursor(Qt::PointingHandCursor));
fundusImageGroup = new QGroupBox(classifyTab);
fundusImageGroup->setObjectName(QStringLiteral("fundusImageGroup"));
fundusImageGroup->setGeometry(QRect(10, 120, 231, 181));
```

```
sizePolicy.setHeightForWidth(fundusImageGroup->sizePolicy().hasHeightForWidth());
fundusImageGroup->setSizePolicy(sizePolicy);
QPalette palette8;
palette8.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette8.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette8.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
fundusImageGroup->setPalette(palette8);
fundusImageGroup->setAutoFillBackground(false);
fundusImageGroup->setFlat(false);
fundusImageGroup->setCheckable(false);
fundusImageLabel = new QLabel(fundusImageGroup);
fundusImageLabel->setObjectName(QStringLiteral("fundusImageLabel"));
fundusImageLabel->setGeometry(QRect(10, 20, 211, 151));
sizePolicy.setHeightForWidth(fundusImageLabel->sizePolicy().hasHeightForWidth());
fundusImageLabel->setSizePolicy(sizePolicy);
fundusImageLabel->setAutoFillBackground(false);
fundusImageLabel->setScaledContents(true);
fundusImageLabel->setAlignment(Qt::AlignCenter);
bvImageGroup = new QGroupBox(classifyTab);
bvImageGroup->setObjectName(QStringLiteral("bvImageGroup"));
bvImageGroup->setGeometry(QRect(260, 120, 231, 181));
sizePolicy.setHeightForWidth(bvImageGroup->sizePolicy().hasHeightForWidth());
bvImageGroup->setSizePolicy(sizePolicy);
QPalette palette9;
palette9.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette9.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette9.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
bvImageGroup->setPalette(palette9);
bvImageGroup->setAutoFillBackground(false);
bvImageGroup->setFlat(false);
bvImageGroup->setCheckable(false);
bvImageLabel = new QLabel(bvImageGroup);
bvImageLabel->setObjectName(QStringLiteral("bvImageLabel"));
bvImageLabel->setGeometry(QRect(10, 20, 211, 151));
sizePolicy.setHeightForWidth(bvImageLabel->sizePolicy().hasHeightForWidth());
bvImageLabel->setSizePolicy(sizePolicy);
bvImageLabel->setAutoFillBackground(false);
bvImageLabel->setScaledContents(true);
bvImageLabel->setAlignment(Qt::AlignCenter);
exuImageGroup = new QGroupBox(classifyTab);
exuImageGroup->setObjectName(QStringLiteral("exuImageGroup"));
exuImageGroup->setGeometry(QRect(10, 320, 231, 181));
sizePolicy.setHeightForWidth(exuImageGroup->sizePolicy().hasHeightForWidth());
exuImageGroup->setSizePolicy(sizePolicy);
QPalette palette10;
palette10.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette10.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette10.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
exuImageGroup->setPalette(palette10);
exuImageGroup->setAutoFillBackground(false);
exuImageGroup->setFlat(false);
exuImageGroup->setCheckable(false);
exuImageLabel = new QLabel(exuImageGroup);
exuImageLabel->setObjectName(QStringLiteral("exuImageLabel"));
exuImageLabel->setGeometry(QRect(10, 20, 211, 151));
sizePolicy.setHeightForWidth(exuImageLabel->sizePolicy().hasHeightForWidth());
exuImageLabel->setSizePolicy(sizePolicy);
exuImageLabel->setAutoFillBackground(false);
exuImageLabel->setScaledContents(true);
exuImageLabel->setAlignment(Qt::AlignCenter);
maImageGroup = new QGroupBox(classifyTab);
maImageGroup->setObjectName(QStringLiteral("maImageGroup"));
maImageGroup->setGeometry(QRect(260, 320, 231, 181));
sizePolicy.setHeightForWidth(maImageGroup->sizePolicy().hasHeightForWidth());
maImageGroup->setSizePolicy(sizePolicy);
QPalette palette11;
palette11.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette11.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette11.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
maImageGroup->setPalette(palette11);
maImageGroup->setAutoFillBackground(false);
maImageGroup->setFlat(false);
maImageGroup->setCheckable(false);
maImageLabel = new QLabel(maImageGroup);
maImageLabel->setObjectName(QStringLiteral("maImageLabel"));
maImageLabel->setGeometry(QRect(10, 20, 211, 151));
sizePolicy.setHeightForWidth(maImageLabel->sizePolicy().hasHeightForWidth());
maImageLabel->setSizePolicy(sizePolicy);
maImageLabel->setAutoFillBackground(false);
maImageLabel->setScaledContents(true);
maImageLabel->setAlignment(Qt::AlignCenter);
resultLabel = new QLabel(classifyTab);
resultLabel->setObjectName(QStringLiteral("resultLabel"));
resultLabel->setGeometry(QRect(150, 520, 81, 41));
QPalette palette12;
palette12.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette12.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette12.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
resultLabel->setPalette(palette12);
QFont font6;
font6.setPointSize(15);
font6.setBold(false);
```

```
font6.setWeight(50);
resultLabel->setFont(font6);
resultLabel->setAlignment(Qt::AlignCenter);
resultText = new QLineEdit(classifyTab);
resultText->setObjectName(QStringLiteral("resultText"));
resultText->setGeometry(QRect(230, 520, 131, 41));
QFont font7;
font7.setPointSize(15);
font7.setBold(true);
font7.setItalic(false);
font7.setWeight(75);
resultText->setFont(font7);
resultText->setCursor(QCursor(Qt::ArrowCursor));
resultText->setToolTipDuration(-1);
resultText->setLayoutDirection(Qt::LeftToRight);
resultText->setAutoFillBackground(false);
resultText->setAlignment(Qt::AlignCenter);
resultText->setReadOnly(true);
exportButton = new QPushButton(classifyTab);
exportButton->setObjectName(QStringLiteral("exportButton"));
exportButton->setGeometry(QRect(510, 490, 91, 71));
QPalette palette13;
palette13.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette13.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette13.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
exportButton->setPalette(palette13);
exportButton->setFont(font5);
exportButton->setCursor(QCursor(Qt::PointingHandCursor));
homeButton1 = new QPushButton(classifyTab);
homeButton1->setObjectName(QStringLiteral("homeButton1"));
homeButton1->setGeometry(QRect(610, 490, 91, 71));
QPalette palette14;
palette14.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette14.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette14.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
homeButton1->setPalette(palette14);
homeButton1->setFont(font5);
homeButton1->setCursor(QCursor(Qt::PointingHandCursor));
movieLabel = new QLabel(classifyTab);
movieLabel->setObjectName(QStringLiteral("movieLabel"));
movieLabel->setGeometry(QRect(250, 250, 201, 181));
QPalette palette15;
palette15.setBrush(QPalette::Active, QPalette::Base, brush1);
palette15.setBrush(QPalette::Inactive, QPalette::Base, brush1);
palette15.setBrush(QPalette::Disabled, QPalette::Base, brush4);
movieLabel->setPalette(palette15);
movieLabel->setAutoFillBackground(false);
movieLabel->setAlignment(Qt::AlignCenter);
glcmGroup = new QGroupBox(classifyTab);
glcmGroup->setObjectName(QStringLiteral("glcmGroup"));
glcmGroup->setGeometry(QRect(510, 120, 191, 231));
QPalette palette16;
palette16.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette16.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette16.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
glcmGroup->setPalette(palette16);
contrastLabel = new QLabel(glcmGroup);
contrastLabel->setObjectName(QStringLiteral("contrastLabel"));
contrastLabel->setGeometry(QRect(60, 20, 71, 21));
QPalette palette17;
palette17.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette17.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette17.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
contrastLabel->setPalette(palette17);
contrastLabel->setFont(font4);
contrastLabel->setAlignment(Qt::AlignCenter);
contrastValue = new QLineEdit(glcmGroup);
contrastValue->setObjectName(QStringLiteral("contrastValue"));
contrastValue->setGeometry(QRect(32, 40, 131, 20));
contrastValue->setCursor(QCursor(Qt::ArrowCursor));
contrastValue->setReadOnly(true);
homogeneityLabel = new QLabel(glcmGroup);
homogeneityLabel->setObjectName(QStringLiteral("homogeneityLabel"));
homogeneityLabel->setGeometry(QRect(50, 70, 91, 21));
QPalette palette18;
palette18.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette18.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette18.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
homogeneityLabel->setPalette(palette18);
homogeneityLabel->setFont(font4);
homogeneityLabel->setAlignment(Qt::AlignCenter);
homogeneityValue = new QLineEdit(glcmGroup);
homogeneityValue->setObjectName(QStringLiteral("homogeneityValue"));
homogeneityValue->setGeometry(QRect(32, 90, 131, 20));
homogeneityValue->setCursor(QCursor(Qt::ArrowCursor));
homogeneityValue->setReadOnly(true);
correlationLabel = new QLabel(glcmGroup);
correlationLabel->setObjectName(QStringLiteral("correlationLabel"));
correlationLabel->setGeometry(QRect(50, 120, 91, 21));
QPalette palette19;
palette19.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette19.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
```

```
palette19.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
correlationLabel->setPalette(palette19);
correlationLabel->setFont(font4);
correlationLabel->setAlignment(Qt::AlignCenter);
correlationValue = new QLineEdit(glcmGroup);
correlationValue->setObjectName(QStringLiteral("correlationValue"));
correlationValue->setGeometry(QRect(32, 140, 131, 20));
correlationValue->setCursor(QCursor(Qt::ArrowCursor));
correlationValue->setReadOnly(true);
energyLabel = new QLabel(glcmGroup);
energyLabel->setObjectName(QStringLiteral("energyLabel"));
energyLabel->setGeometry(QRect(50, 170, 91, 21));
QPalette palette20;
palette20.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette20.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette20.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
energyLabel->setPalette(palette20);
energyLabel->setFont(font4);
energyLabel->setAlignment(Qt::AlignCenter);
energyValue = new QLineEdit(glcmGroup);
energyValue->setObjectName(QStringLiteral("energyValue"));
energyValue->setGeometry(QRect(32, 190, 131, 20));
energyValue->setCursor(QCursor(Qt::ArrowCursor));
energyValue->setReadOnly(true);
backgroundLabel2 = new QLabel(classifyTab);
backgroundLabel2->setObjectName(QStringLiteral("backgroundLabel2"));
backgroundLabel2->setGeometry(QRect(0, 10, 721, 581));
sizePolicy.setHeightForWidth(backgroundLabel2->sizePolicy().hasHeightForWidth());
backgroundLabel2->setSizePolicy(sizePolicy);
backgroundLabel2->setMaximumSize(QSize(721, 581));
backgroundLabel2->setPixmap(QPixmap(QString::fromUtf8(":/images/tabBackground.png")));
backgroundLabel2->setScaledContents(true);
clearButton = new QPushButton(classifyTab);
clearButton->setObjectName(QStringLiteral("clearButton"));
clearButton->setGeometry(QRect(560, 390, 91, 71));
QPalette palette21;
palette21.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette21.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette21.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
clearButton->setPalette(palette21);
clearButton->setFont(font5);
clearButton->setCursor(QCursor(Qt::PointingHandCursor));
tabWidget->addTab(classifyTab, QString());
backgroundLabel2->raise();
glcmGroup->raise();
fileSelectGroup->raise();
fundusImageGroup->raise();
bvImageGroup->raise();
exuImageGroup->raise();
maImageGroup->raise();
resultLabel->raise();
resultText->raise();
exportButton->raise();
homeButton1->raise();
movieLabel->raise();
clearButton->raise();
tutorialsTab = new QWidget();
tutorialsTab->setObjectName(QStringLiteral("tutorialsTab"));
tutorialsTab->setAutoFillBackground(true);
homeButton2 = new QPushButton(tutorialsTab);
homeButton2->setObjectName(QStringLiteral("homeButton2"));
homeButton2->setGeometry(QRect(610, 490, 91, 71));
QPalette palette22;
palette22.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
palette22.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
palette22.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
homeButton2->setPalette(palette22);
homeButton2->setFont(font5);
homeButton2->setCursor(QCursor(Qt::PointingHandCursor));
backgroundLabel3 = new QLabel(tutorialsTab);
backgroundLabel3->setObjectName(QStringLiteral("backgroundLabel3"));
backgroundLabel3->setGeometry(QRect(0, 10, 721, 581));
sizePolicy.setHeightForWidth(backgroundLabel3->sizePolicy().hasHeightForWidth());
backgroundLabel3->setSizePolicy(sizePolicy);
backgroundLabel3->setMaximumSize(QSize(721, 581));
backgroundLabel3->setPixmap(QPixmap(QString::fromUtf8(":/images/tabBackground.png")));
backgroundLabel3->setScaledContents(true);
learningMaterialsGroup = new QGroupBox(tutorialsTab);
learningMaterialsGroup->setObjectName(QStringLiteral("learningMaterialsGroup"));
learningMaterialsGroup->setGeometry(QRect(20, 460, 571, 101));
QPalette palette23;
palette23.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette23.setBrush(QPalette::Active, QPalette::ButtonText, brush5);
palette23.setBrush(QPalette::Active, QPalette::ToolTipText, brush5);
palette23.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette23.setBrush(QPalette::Inactive, QPalette::ButtonText, brush5);
palette23.setBrush(QPalette::Inactive, QPalette::ToolTipText, brush5);
palette23.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
palette23.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
palette23.setBrush(QPalette::Disabled, QPalette::ToolTipText, brush5);
learningMaterialsGroup->setPalette(palette23);
learningMaterial3 = new QLabel(learningMaterialsGroup);
```

```
learningMaterial3->setObjectName(QStringLiteral("learningMaterial3"));
learningMaterial3->setGeometry(QRect(10, 70, 551, 20));
QPalette palette24;
palette24.setBrush(QPalette::Active, QPalette::WindowText, brush);
palette24.setBrush(QPalette::Inactive, QPalette::WindowText, brush);
palette24.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
learningMaterial3->setPalette(palette24);
learningMaterial3->setFont(font3);
learningMaterial3->setOpenExternalLinks(true);
learningMaterial3->setTextInteractionFlags(Qt::TextBrowserInteraction);
learningMaterial1_2 = new QLabel(learningMaterialsGroup);
learningMaterial1_2->setObjectName(QStringLiteral("learningMaterial1_2"));
learningMaterial1_2->setGeometry(QRect(10, 30, 551, 20));
QPalette palette25;
palette25.setBrush(QPalette::Active, QPalette::WindowText, brush);
palette25.setBrush(QPalette::Inactive, QPalette::WindowText, brush);
palette25.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
learningMaterial1_2->setPalette(palette25);
learningMaterial1_2->setFont(font3);
learningMaterial1_2->setOpenExternalLinks(true);
learningMaterial1_2->setTextInteractionFlags(Qt::TextBrowserInteraction);
learningMaterial2 = new QLabel(learningMaterialsGroup);
learningMaterial2->setObjectName(QStringLiteral("learningMaterial2"));
learningMaterial2->setGeometry(QRect(10, 50, 551, 20));
QPalette palette26;
palette26.setBrush(QPalette::Active, QPalette::WindowText, brush);
palette26.setBrush(QPalette::Inactive, QPalette::WindowText, brush);
palette26.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
learningMaterial2->setPalette(palette26);
learningMaterial2->setFont(font3);
learningMaterial2->setOpenExternalLinks(true);
learningMaterial2->setTextInteractionFlags(Qt::TextBrowserInteraction);
videoWidget = new QVideoWidget(tutorialsTab);
videoWidget->setObjectName(QStringLiteral("videoWidget"));
videoWidget->setGeometry(QRect(210, 40, 491, 331));
QPalette palette27;
palette27.setBrush(QPalette::Active, QPalette::Base, brush5);
palette27.setBrush(QPalette::Active, QPalette::Window, brush3);
palette27.setBrush(QPalette::Inactive, QPalette::Base, brush5);
palette27.setBrush(QPalette::Inactive, QPalette::Window, brush3);
palette27.setBrush(QPalette::Disabled, QPalette::Base, brush3);
palette27.setBrush(QPalette::Disabled, QPalette::Window, brush3);
videoWidget->setPalette(palette27);
videoWidget->setAutoFillBackground(true);
playButton = new QCommandLinkButton(tutorialsTab);
playButton->setObjectName(QStringLiteral("playButton"));
playButton->setGeometry(QRect(360, 370, 61, 71));
QIcon icon1;
icon1.addFile(QStringLiteral(":/images/playButton.png"), QSize(), QIcon::Normal, QIcon::
    Off);
playButton->setIcon(icon1);
playButton->setIconSize(QSize(50, 50));
groupBox = new QGroupBox(tutorialsTab);
groupBox->setObjectName(QStringLiteral("groupBox"));
groupBox->setGeometry(QRect(20, 30, 161, 411));
QPalette palette28;
palette28.setBrush(QPalette::Active, QPalette::WindowText, brush5);
palette28.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
palette28.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
groupBox->setPalette(palette28);
drPlayButton = new QPushButton(groupBox);
drPlayButton->setObjectName(QStringLiteral("drPlayButton"));
drPlayButton->setGeometry(QRect(10, 40, 141, 91));
QPalette palette29;
palette29.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette29.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette29.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
drPlayButton->setPalette(palette29);
drPlayButton->setFont(font);
dr2PlayButton = new QPushButton(groupBox);
dr2PlayButton->setObjectName(QStringLiteral("dr2PlayButton"));
dr2PlayButton->setGeometry(QRect(10, 170, 141, 91));
QPalette palette30;
palette30.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette30.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette30.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
dr2PlayButton->setPalette(palette30);
dr2PlayButton->setFont(font);
svmPlayButton = new QPushButton(groupBox);
svmPlayButton->setObjectName(QStringLiteral("svmPlayButton"));
svmPlayButton->setGeometry(QRect(10, 300, 141, 91));
QPalette palette31;
palette31.setBrush(QPalette::Active, QPalette::ButtonText, brush);
palette31.setBrush(QPalette::Inactive, QPalette::ButtonText, brush);
palette31.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
svmPlayButton->setPalette(palette31);
svmPlayButton->setFont(font);
pauseButton = new QCommandLinkButton(tutorialsTab);
pauseButton->setObjectName(QStringLiteral("pauseButton"));
pauseButton->setGeometry(QRect(420, 370, 61, 71));
QIcon icon2;
icon2.addFile(QStringLiteral(":/images/pauseButton.png"), QSize(), QIcon::Normal, QIcon::
```

```
        Off);
    pauseButton->setIcon(icon2);
    pauseButton->setIconSize(QSize(50, 50));
    stopButton = new QCommandLinkButton(tutorialsTab);
    stopButton->setObjectName(QStringLiteral("stopButton"));
    stopButton->setGeometry(QRect(480, 370, 61, 71));
    QIcon icon3;
    icon3.addFile(QStringLiteral(":/images/stopButton.png"), QSize(), QIcon::Normal, QIcon::
        Off);
    stopButton->setIcon(icon3);
    stopButton->setIconSize(QSize(50, 50));
    tabWidget->addTab(tutorialsTab, QString());
    backgroundLabel3->raise();
    homeButton2->raise();
    learningMaterialsGroup->raise();
    videoWidget->raise();
    playButton->raise();
    groupBox->raise();
    pauseButton->raise();
    stopButton->raise();
    helpTab = new QWidget();
    helpTab->setObjectName(QStringLiteral("helpTab"));
    helpTab->setAutoFillBackground(true);
    homeButton3 = new QPushButton(helpTab);
    homeButton3->setObjectName(QStringLiteral("homeButton3"));
    homeButton3->setGeometry(QRect(610, 490, 91, 71));
    QPalette palette32;
    palette32.setBrush(QPalette::Active, QPalette::ButtonText, brush1);
    palette32.setBrush(QPalette::Inactive, QPalette::ButtonText, brush1);
    palette32.setBrush(QPalette::Disabled, QPalette::ButtonText, brush2);
    homeButton3->setPalette(palette32);
    homeButton3->setFont(font5);
    homeButton3->setCursor(QCursor(Qt::PointingHandCursor));
    backgroundLabel4 = new QLabel(helpTab);
    backgroundLabel4->setObjectName(QStringLiteral("backgroundLabel4"));
    backgroundLabel4->setGeometry(QRect(0, 10, 721, 581));
    sizePolicy.setHeightForWidth(backgroundLabel4->sizePolicy().hasHeightForWidth());
    backgroundLabel4->setSizePolicy(sizePolicy);
    backgroundLabel4->setMaximumSize(QSize(721, 581));
    backgroundLabel4->setPixmap(QPixmap(QString::fromUtf8(":/images/tabBackground.png")));
    backgroundLabel4->setScaledContents(true);
    backgroundLabel4->setWordWrap(true);
    faqLabel = new QLabel(helpTab);
    faqLabel->setObjectName(QStringLiteral("faqLabel"));
    faqLabel->setGeometry(QRect(210, 30, 291, 51));
    QPalette palette33;
    palette33.setBrush(QPalette::Active, QPalette::WindowText, brush5);
    palette33.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
    palette33.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
    faqLabel->setPalette(palette33);
    QFont font8;
    font8.setPointSize(15);
    font8.setUnderline(true);
    faqLabel->setFont(font8);
    faqLabel->setAlignment(Qt::AlignCenter);
    question1Label = new QLabel(helpTab);
    question1Label->setObjectName(QStringLiteral("question1Label"));
    question1Label->setGeometry(QRect(60, 80, 121, 51));
    question1Label->setFont(font);
    question1Label->setAlignment(Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter);
    question2Label = new QLabel(helpTab);
    question2Label->setObjectName(QStringLiteral("question2Label"));
    question2Label->setGeometry(QRect(60, 170, 321, 51));
    question2Label->setFont(font);
    question2Label->setAlignment(Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter);
    question3Label = new QLabel(helpTab);
    question3Label->setObjectName(QStringLiteral("question3Label"));
    question3Label->setGeometry(QRect(60, 240, 501, 51));
    question3Label->setFont(font);
    question3Label->setAlignment(Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter);
    question4Label = new QLabel(helpTab);
    question4Label->setObjectName(QStringLiteral("question4Label"));
    question4Label->setGeometry(QRect(60, 320, 501, 51));
    question4Label->setFont(font);
    question4Label->setAlignment(Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter);
    answer1Label = new QLabel(helpTab);
    answer1Label->setObjectName(QStringLiteral("answer1Label"));
    answer1Label->setGeometry(QRect(80, 120, 551, 61));
    QPalette palette34;
    palette34.setBrush(QPalette::Active, QPalette::WindowText, brush5);
    palette34.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
    palette34.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
    answer1Label->setPalette(palette34);
    answer1Label->setFont(font3);
    answer1Label->setTextFormat(Qt::RichText);
    answer1Label->setWordWrap(true);
    answer2Label = new QLabel(helpTab);
    answer2Label->setObjectName(QStringLiteral("answer2Label"));
    answer2Label->setGeometry(QRect(80, 200, 551, 51));
    QPalette palette35;
    palette35.setBrush(QPalette::Active, QPalette::WindowText, brush5);
    palette35.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
```

```
        palette35.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
        answer2Label->setPalette(palette35);
        answer2Label->setFont(font3);
        answer2Label->setWordWrap(true);
        answer3Label = new QLabel(helpTab);
        answer3Label->setObjectName(QStringLiteral("answer3Label"));
        answer3Label->setGeometry(QRect(80, 280, 551, 51));
        QPalette palette36;
        palette36.setBrush(QPalette::Active, QPalette::WindowText, brush5);
        palette36.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
        palette36.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
        answer3Label->setPalette(palette36);
        answer3Label->setFont(font3);
        answer3Label->setWordWrap(true);
        answer4Label = new QLabel(helpTab);
        answer4Label->setObjectName(QStringLiteral("answer4Label"));
        answer4Label->setGeometry(QRect(80, 360, 551, 51));
        QPalette palette37;
        palette37.setBrush(QPalette::Active, QPalette::WindowText, brush5);
        palette37.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
        palette37.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
        answer4Label->setPalette(palette37);
        answer4Label->setFont(font3);
        answer4Label->setWordWrap(true);
        question5Label = new QLabel(helpTab);
        question5Label->setObjectName(QStringLiteral("question5Label"));
        question5Label->setGeometry(QRect(60, 400, 501, 51));
        question5Label->setFont(font);
        question5Label->setAlignment(Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter);
        answer5Label = new QLabel(helpTab);
        answer5Label->setObjectName(QStringLiteral("answer5Label"));
        answer5Label->setGeometry(QRect(80, 440, 551, 51));
        QPalette palette38;
        palette38.setBrush(QPalette::Active, QPalette::WindowText, brush5);
        palette38.setBrush(QPalette::Inactive, QPalette::WindowText, brush5);
        palette38.setBrush(QPalette::Disabled, QPalette::WindowText, brush2);
        answer5Label->setPalette(palette38);
        answer5Label->setFont(font3);
        answer5Label->setWordWrap(true);
        tabWidget->addTab(helpTab, QString());
        backgroundLabel4->raise();
        homeButton3->raise();
        faqLabel->raise();
        question1Label->raise();
        question2Label->raise();
        question3Label->raise();
        question4Label->raise();
        answer1Label->raise();
        answer2Label->raise();
        answer3Label->raise();
        answer4Label->raise();
        question5Label->raise();
        answer5Label->raise();
        DRDTClass->setCentralWidget(centralWidget);

        retranslateUi(DRDTClass);

        tabWidget->setCurrentIndex(0);
        startClassificationButton->setDefault(true);


        QMetaObject::connectSlotsByName(DRDTClass);
    } // setupUi

    void retranslateUi(QMainWindow *DRDTClass)
    {
        DRDTClass->setWindowTitle(QApplication::translate("DRDTClass", "Diabetic Retinopathy
            Detection Tool (DRDT)", 0));
        startClassificationButton->setText(QApplication::translate("DRDTClass", "Start
            Classification", 0));
        quitButton->setText(QApplication::translate("DRDTClass", "Quit", 0));
        backgroundLabel->setText(QString());
        watchTutorialsButton->setText(QApplication::translate("DRDTClass", "Watch Tutorials", 0));
        helpButton->setText(QApplication::translate("DRDTClass", "FAQ", 0));
        tabWidget->setTabText(tabWidget->indexOf(homeTab), QApplication::translate("DRDTClass", "
            Home", 0));
        fileSelectGroup->setTitle(QApplication::translate("DRDTClass", "File Selector", 0));
        imageLabel->setText(QApplication::translate("DRDTClass", "Image:", 0));
        imagePath->setText(QString());
        imagePath->setPlaceholderText(QString());
#ifndef QT_NO_TOOLTIP
        imageButton->setToolTip(QString());
#endif // QT_NO_TOOLTIP
        imageButton->setText(QApplication::translate("DRDTClass", "Browse", 0));
        classifyButton->setText(QApplication::translate("DRDTClass", "Classify", 0));
        fundusImageGroup->setTitle(QApplication::translate("DRDTClass", "Original Image", 0));
        fundusImageLabel->setText(QString());
        bvImageGroup->setTitle(QApplication::translate("DRDTClass", "Blood Vessels", 0));
        bvImageLabel->setText(QString());
        exuImageGroup->setTitle(QApplication::translate("DRDTClass", "Exudates", 0));
        exuImageLabel->setText(QString());
        maImageGroup->setTitle(QApplication::translate("DRDTClass", "Microaneurysms", 0));
```

```
        maImageLabel->setText(QString());
        resultLabel->setText(QApplication::translate("DRDTClass", "Result:", 0));
#ifndef QT_NO_TOOLTIP
        resultText->setToolTip(QString());
#endif // QT_NO_TOOLTIP
        resultText->setText(QString());
        exportButton->setText(QApplication::translate("DRDTClass", "Export to\n"
"PDF", 0));
        homeButton1->setText(QApplication::translate("DRDTClass", "Home", 0));
        movieLabel->setText(QString());
        glcmGroup->setTitle(QApplication::translate("DRDTClass", "GLCM", 0));
        contrastLabel->setText(QApplication::translate("DRDTClass", "Contrast", 0));
        homogeneityLabel->setText(QApplication::translate("DRDTClass", "Homogeneity", 0));
        correlationLabel->setText(QApplication::translate("DRDTClass", "Correlation", 0));
        energyLabel->setText(QApplication::translate("DRDTClass", "Energy", 0));
        backgroundLabel2->setText(QString());
        clearButton->setText(QApplication::translate("DRDTClass", "Clear All", 0));
        tabWidget->setTabText(tabWidget->indexOf(classifyTab), QApplication::translate("DRDTClass
            ", "Classify", 0));
        homeButton2->setText(QApplication::translate("DRDTClass", "Home", 0));
        backgroundLabel3->setText(QString());
        learningMaterialsGroup->setTitle(QApplication::translate("DRDTClass", "Learning Materials
            ", 0));
        learningMaterial3->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>SVM
             on OpenCV: <a href=\"http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/
            introduction_to_svm.html\"><span style=\" text-decoration: underline; color:#ffffff
            ;\">http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/
            introduction_to_svm.html</span></a></p></body></html>", 0));
        learningMaterial1_2->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>
            Facts about Diabetic Eye Diseases: <a href=\"https://nei.nih.gov/health/diabetic/
            retinopathy\"><span style=\" color:#ffffff;\">https://nei.nih.gov/health/diabetic/
            retinopathy</span></a></p></body></html>", 0));
        learningMaterial2->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>
            Diabetic Retinopathy Analysis: <a href=\"http://www.ncbi.nlm.nih.gov/pmc/articles/
            PMC1138264/\"><span style=\" color:#ffffff;\">http://www.ncbi.nlm.nih.gov/pmc/articles
            /PMC1138264/</span></a></p></body></html>", 0));
        playButton->setText(QString());
        groupBox->setTitle(QApplication::translate("DRDTClass", "Select a video", 0));
        drPlayButton->setText(QApplication::translate("DRDTClass", "Diabetic \n"
"Retinopathy \n"
"by NIH", 0));
        dr2PlayButton->setText(QApplication::translate("DRDTClass", "Diabetic \n"
"Retinopathy \n"
"by Khan Academy ", 0));
        svmPlayButton->setText(QApplication::translate("DRDTClass", "Overview \n"
"on Support \n"
"Vector Machines", 0));
        pauseButton->setText(QString());
        stopButton->setText(QString());
        tabWidget->setTabText(tabWidget->indexOf(tutorialsTab), QApplication::translate("DRDTClass
            ", "Tutorials", 0));
        homeButton3->setText(QApplication::translate("DRDTClass", "Home", 0));
        backgroundLabel4->setText(QString());
        faqLabel->setText(QApplication::translate("DRDTClass", "Frequently Asked Questions", 0));
        question1Label->setText(QApplication::translate("DRDTClass", "What is DRDT?", 0));
        question2Label->setText(QApplication::translate("DRDTClass", "How do I classify an image
            using this tool?", 0));
        question3Label->setText(QApplication::translate("DRDTClass", "What are the features used
            in classification of an image in DRDT?", 0));
        question4Label->setText(QApplication::translate("DRDTClass", "How do I save the results of
             the classification for future use?", 0));
        answer1Label->setText(QApplication::translate("DRDTClass", "<html><head/><body><p><span
            style=\" font-style:italic; color:#f38900;\">Diabetic Retinopathy Detection Tool (DRCT
            )</span> is a program that uses image processing techniques and to determine if an
            image exhibits diabetic retinopathy or if it does not. To learn more about diabetic
            retinopathy and support vector machines, go to the <span style=\" font-style:italic;
            color:#f38900;\">Classify</span> tab.</p></body></html>", 0));
        answer2Label->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>Click
            the <span style=\" font-style:italic; color:#f38900;\">Classify</span> tab. Click the
            <span style=\" font-style:italic; color:#f38900;\">Browse</span> button beside the <
            span style=\" font-style:italic;\">Image</span> textbox and select a fundus image in <
            span style=\" font-style:italic; color:#f38900;\">TIFF</span> format. Then, click the
            <span style=\" font-style:italic; color:#f38900;\">Classify</span> button. </p></body
            ></html>", 0));
        answer3Label->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>The
            default model file in DRDT uses 7 image features: <span style=\" font-style:italic;
            color:#f38900;\">Area of Blood Vessels</span>, <span style=\" font-style:italic; color
            :#f38900;\">Area of Exudates</span>, <span style=\" font-style:italic; color:#f38900
            ;\">Area of Microaneurysms</span>, and the GLCM descriptors <span style=\" font-style:
            italic; color:#f38900;\">Contrast</span>, <span style=\" font-style:italic; color:#
            f38900;\">Homogeneity</span>, <span style=\" font-style:italic; color:#f38900;\">
            Correlation</span>, and <span style=\" font-style:italic; color:#f38900;\">Energy</
            span>.</p></body></html>", 0));
        answer4Label->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>After
            detecting if an image exhibits diabetic retinopathy in the <span style=\" font-style:
            italic; color:#f38900;\">Classify</span> tab, click the <span style=\" font-style:
            italic; color:#f38900;\">Browse</span> button and choose the name and directory of
            where you want your file. DRDT will save the result in a <span style=\" font-style:
            italic; color:#f38900;\">PDF</span> file.</p></body></html>", 0));
        question5Label->setText(QApplication::translate("DRDTClass", "Are there any learning
            materials this tool offers?", 0));
        answer5Label->setText(QApplication::translate("DRDTClass", "<html><head/><body><p>There
```

```
            are videos about diabetic retinopathy and support vector machines in the <span style
            =\" font-style:italic; color:#f38900;\">Tutorials</span> tab. It can be watched by
            selecting a video from the <span style=\" font-style:italic; color:#f38900;\">list</
            span> on the left side of the the tab and clicking the <span style=\" font-style:
            italic; color:#f38900;\">play</span> button. There are also links available on the
            bottom of the <span style=\" font-style:italic; color:#f38900;\">Tutorials</span> tab
            .</p></body></html>", 0));
        tabWidget->setTabText(tabWidget->indexOf(helpTab), QApplication::translate("DRDTClass", "
            FAQ", 0));
    } // retranslateUi

};

namespace Ui {
    class DRDTClass: public Ui_DRDTClass {};
} // namespace Ui

QT_END_NAMESPACE

#endif // UI_DRDT_H


#ifndef DRDT_H
#define DRDT_H

#include "ui_drdt.h"

#include <opencv2\core\core.hpp>
#include <opencv2\ml\ml.hpp>

#include <QtCore\qfuturewatcher.h>
#include <QtMultimedia\qmediaplayer.h>
#include <QtMultimediaWidgets\qvideowidget.h>

using namespace cv;
using namespace std;

//ImageProcessing
Mat binarizeImage(Mat src, int thresh);
Mat medianFilter(Mat src, int maxKernelLength);
Mat contrastLimitedAHE(Mat src);
Mat closingOperation(Mat src, int morphSize);
Mat removeSmallerDetails(Mat src, int sizeOfDetail, int thresh);
vector<Mat> processImage(string imageLoc);
QPixmap convertMatToQpixmap(Mat src, QImage::Format format);

//ExudatesDetection
Mat exudatesDetection(Mat src, Mat mask);

//BloodVesselsDetection
Mat bloodVesselsPreDetection(Mat src, Mat maskImage, Mat exudatesImage);
Mat averageBloodVessels(Mat firstBvImage, Mat secondBvImage);

//MicroaneurysmsDetection
Mat microaneurysmsPreDetection(Mat src, Mat maskImage);

//Glcm
vector<float> getGLCM(string imageLoc);

class DRDT : public QMainWindow, private Ui::DRDTClass {
        Q_OBJECT

public:
        DRDT(QWidget *parent = 0);
        ~DRDT();

private slots:
        void switchToClassifyTab();
        void switchToTutorialsTab();
        void switchToFAQTab();
        void quitApp();

        void switchToHomeTab();

        void getImagePath();
        void classifyImage();
        void finishedImageProcessing();
        void clearContents();
        void exportResults();

        void drVideoSelected();
        void dr2VideoSelected();
        void svmVideoSelected();
        void playVideo();
        void pauseVideo();
        void stopVideo();
        void finishedVideo();

private:
        CvSVM svm;
        string imageLoc;
```

```
        QMovie *movie;
        QFuture<vector<Mat>> future;
        QFutureWatcher<void> watcher;

        QMediaPlayer *mediaPlayer;
};


#endif // DRDT_H


/**
*BloodVesselsDetection.cpp
*This file contains all functions
*exclusive for segmenting the
*blood vessels from the fundus
*image
*/

#include "drdt.h"

#include <opencv2\legacy\legacy.hpp>
#include <opencv2\highgui\highgui.hpp>

using namespace cv;
using namespace std;

/**
*Gets the green channel image
*bysplitting image to channels
*parameters
*src - image to be processed
*/

Mat getGreenChannel(Mat src) {
        vector<Mat> channel;

        split(src, channel);

        return channel[1];
}

/**
*Finds the supremum of openings
*parameters:
*src - image to be processed
*morphSize - size of opening
*/

Mat openingOperation(Mat src, int morphSize) {
        Mat flippedElement, openedImage, openedImage1, openedImage2, openedImage3, openedImage4;
        int morphOperation = 2; //opening
        int morphElement = 0;   //rectangle element

        Mat element = getStructuringElement(morphElement, Size(morphSize, 1));
        morphologyEx(src, openedImage1, morphOperation, element);
        element = getStructuringElement(morphElement, Size(1, morphSize));
        morphologyEx(src, openedImage2, morphOperation, element);
        element = Mat::eye(morphSize, morphSize, element.type());
        morphologyEx(src, openedImage3, morphOperation, element);
        flip(element, flippedElement, 1);
        morphologyEx(src, openedImage4, morphOperation, flippedElement);

        openedImage = max(openedImage1, openedImage2);
        openedImage = max(openedImage, openedImage3);
        openedImage = max(openedImage, openedImage4);

        return openedImage;
}

/**
*Finds the supremum of tophats
*parameters:
*src - image to be processed
*morphSize - size of tophat
*/

Mat topHatOperation(Mat src, Mat bg, int morphSize) {
        Mat flippedElement, topHatImage, topHatImage1, topHatImage2, topHatImage3, topHatImage4;
        int morphOperation = 5; //tophat
        int morphElement = 0;   //rectangle

        Mat element = getStructuringElement(morphElement, Size(morphSize, 1));
        morphologyEx(src, topHatImage1, morphOperation, element);
        element = getStructuringElement(morphElement, Size(1, morphSize));
        morphologyEx(src, topHatImage2, morphOperation, element);
        element = Mat::eye(morphSize, morphSize, element.type());
        morphologyEx(src, topHatImage3, morphOperation, element);
        flip(element, flippedElement, 1);
        morphologyEx(src, topHatImage4, morphOperation, flippedElement);

        topHatImage = max(topHatImage1, topHatImage2);
```

```
        topHatImage = max(topHatImage, topHatImage3);
        topHatImage = max(topHatImage, topHatImage4);

        bitwise_and(topHatImage, bg, topHatImage);

        return topHatImage;
}

/**
*Performs alternating sequential
*filter
*parameters:
*src - image to be processed
*/

Mat alternatingSequentialFilter(Mat src) {
        Mat structuringElement;
        Mat asfImage = src.clone();

        int morphOperation = 3; //closing
        for (int i = 0; i < 5; i++) {
                structuringElement = Mat::ones(i, i, CV_8U);
                morphologyEx(asfImage, asfImage, morphOperation, structuringElement);
                morphOperation = 2;     //opening
                morphologyEx(asfImage, asfImage, morphOperation, structuringElement);
        }

        return asfImage;
}

/**
*Performs the continuation algorithm
*on images
*parameters:
*image1 - image to perform ca on
*image2 - seecond image used as
*reference
*/

void continuationAlgorithm(Mat image1, Mat image2) {
        int sum, counter;

        while (true) {
                counter = 0;
                for (int i = 0; i < image1.rows - 3; i++) {
                        for (int j = 0; j < image1.cols - 3; j++) {
                                if ((image1.at<char>(i, j) == 0) && (image2.at<char>(i, j) == 1))
                                    {
                                        sum = image1.at<char>(i, j + 1) +
                                                image1.at<char>(i + 1, j) +
                                                image1.at<char>(i + 1, j + 1) +
                                                image1.at<char>(i + 1, j + 2) +
                                                image1.at<char>(i + 1, j + 3) +
                                                image1.at<char>(i + 2, j + 1) +
                                                image1.at<char>(i + 2, j + 3) +
                                                image1.at<char>(i + 3, j + 1) +
                                                image1.at<char>(i + 3, j + 2);
                                        counter++;
                                        if (sum >= 100)
                                                image1.at<char>(i, j) = 100;
                                        else
                                                counter--;
                                }
                        }
                }

                if (counter == 0)
                        break;
        }

        return;
}

/**
*Averages two blood vessels
*images through the use of
*continuation algorithm
*parameters:
*bvImage1, bvImage2 - images
*to be averaged
*/

Mat averageBloodVessels(Mat bvImage1, Mat bvImage2) {
        Mat normBvImage1, normBvImage2, temp, bloodVesselsImage;

        normalize(bvImage1, normBvImage1, 0, 1, NORM_MINMAX);
        normalize(bvImage2, normBvImage2, 0, 1, NORM_MINMAX);
        normBvImage1 = normBvImage1 * 100;

        continuationAlgorithm(normBvImage1, normBvImage2); //average vessels - original image

        //flip vertically
```

```
        flip(normBvImage1, temp, 1);
        normBvImage1 = temp.clone();
        flip(normBvImage2, temp, 1);
        normBvImage2 = temp.clone();

        continuationAlgorithm(normBvImage1, normBvImage2); //average vessels - vertically flipped
            image

        //flip horizontally
        flip(normBvImage1, temp, 0);
        normBvImage1 = temp.clone();
        flip(normBvImage2, temp, 0);
        normBvImage2 = temp.clone();

        continuationAlgorithm(normBvImage1, normBvImage2); //average vessels - horizontally
            flipped image

        //flip vertically
        flip(normBvImage1, temp, 1);
        normBvImage1 = temp.clone();
        flip(normBvImage2, temp, 1);
        normBvImage2 = temp.clone();

        continuationAlgorithm(normBvImage1, normBvImage2); //average vessels - vertically flipped
            image

        //flip horizontally
        flip(normBvImage1, temp, 0);
        normBvImage1 = temp.clone();
        flip(normBvImage2, temp, 0);
        normBvImage2 = temp.clone();

        //convert the image back to 0-255
        normalize(normBvImage1, bloodVesselsImage, 0, 255, NORM_MINMAX);

        return bloodVesselsImage;
}

/**
*Function which segments the
*basic structure of the blood
*vessels
*parameters:
*src - image to be segmented
*maskImage - mask image
*exudatesImage - exudates
*segmented from image
*/

Mat bloodVesselsPreDetection(Mat src, Mat mask, Mat exudatesImage) {
        Mat greenChannelImage = getGreenChannel(src);
        Mat medianFilterImage = medianFilter(greenChannelImage, 13);
        Mat invertedImage;
        bitwise_not(medianFilterImage, invertedImage);
        Mat openedBvImage = openingOperation(invertedImage, 10);
        Mat topHatBvImage = topHatOperation(openedBvImage, mask, 10);
        Mat noisyBvImage = alternatingSequentialFilter(topHatBvImage);
        Mat cleanBvImage = removeSmallerDetails(noisyBvImage, 550, 2);
        Mat bvBasicStructureImage = cleanBvImage - exudatesImage;

        return bvBasicStructureImage;
}


/**
*drdt.cpp
*This file handles the GUI
*and displaying the results
*of DR detection
*/

#include "drdt.h"

#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

#include <QtWidgets\qfiledialog.h>
#include <QtWidgets\qmessagebox.h>
#include <QtGui\qpainter.h>
#include <QtGui\qmovie.h>
#include <QtConcurrent\qtconcurrentrun.h>
#include <QtPrintSupport\qprinter.h>

using namespace cv;
using namespace std;

/**
*Sets up the main GUI
*/

DRDT::DRDT(QWidget *parent)
        : QMainWindow(parent) {
```

```
        setupUi(this);

        //Disables the maximize button in the GUI
        Qt::WindowFlags flags;
        flags = Qt::Window | Qt::WindowMinimizeButtonHint | Qt::WindowCloseButtonHint;
        setWindowFlags(flags);

        connect(startClassificationButton, SIGNAL(clicked()), this, SLOT(switchToClassifyTab()));
        connect(watchTutorialsButton, SIGNAL(clicked()), this, SLOT(switchToTutorialsTab()));
        connect(helpButton, SIGNAL(clicked()), this, SLOT(switchToFAQTab()));
        connect(quitButton, SIGNAL(clicked()), this, SLOT(quitApp()));

        /**
        *Sets up classify tab's starting
        *state
        */

        connect(imageButton, SIGNAL(clicked()), this, SLOT(getImagePath()));
        connect(classifyButton, SIGNAL(clicked()), this, SLOT(classifyImage()));
        connect(clearButton, SIGNAL(clicked()), this, SLOT(clearContents()));
        connect(exportButton, SIGNAL(clicked()), this, SLOT(exportResults()));
        connect(homeButton1, SIGNAL(clicked()), this, SLOT(switchToHomeTab()));

        classifyButton->setEnabled(false);
        clearButton->setEnabled(false);
        exportButton->setEnabled(false);

        imagePath->clear();

        connect(&this->watcher, SIGNAL(finished()), this, SLOT(finishedImageProcessing()));

        movie = new QMovie(":/images/loading.gif");
        movieLabel->setMovie(movie);

        /**
        *Sets up tutorials tab's starting
        *state
        */

        connect(drPlayButton, SIGNAL(clicked()), this, SLOT(drVideoSelected()));
        connect(dr2PlayButton, SIGNAL(clicked()), this, SLOT(dr2VideoSelected()));
        connect(svmPlayButton, SIGNAL(clicked()), this, SLOT(svmVideoSelected()));
        connect(playButton, SIGNAL(clicked()), this, SLOT(playVideo()));
        connect(pauseButton, SIGNAL(clicked()), this, SLOT(pauseVideo()));
        connect(stopButton, SIGNAL(clicked()), this, SLOT(stopVideo()));
        connect(homeButton2, SIGNAL(clicked()), this, SLOT(switchToHomeTab()));

        mediaPlayer = new QMediaPlayer;
        mediaPlayer->setVideoOutput(videoWidget);
        mediaPlayer->stop();

        playButton->setEnabled(false);
        pauseButton->setEnabled(false);
        stopButton->setEnabled(false);

        /**
        *Sets up FAQ tab's starting
        *state
        */

        connect(homeButton3, SIGNAL(clicked()), this, SLOT(switchToHomeTab()));
}

/**
*Switches the current tab to
*classify tab
*/

void DRDT::switchToClassifyTab() {
        tabWidget->setCurrentIndex(1);

        return;
}

/**
*Switches the current tab to
*tutorials tab
*/

void DRDT::switchToTutorialsTab() {
        tabWidget->setCurrentIndex(2);

        return;
}

/**
*Switches the current tab to
*faq tab
*/

void DRDT::switchToFAQTab() {
```

```
        tabWidget->setCurrentIndex(3);

        return;
}

/**
*Quits the tool
*/

void DRDT::quitApp() {
        QApplication::quit();

        return;
}

/**
*Switches the current tab to
*home tab
*/

void DRDT::switchToHomeTab() {
        tabWidget->setCurrentIndex(0);
        mediaPlayer->stop();

        return;
}

/**
*Gets the location of the image
*and displays it
*/

void DRDT::getImagePath() {
        QString path = QFileDialog::getOpenFileName(
                this,
                "Choose image file to open",
                QDir::rootPath(),
                tr("TIFF (*.tif)"));

        if (path != "") {
                imagePath->setText(path);
                fundusImageLabel->clear();
                bvImageLabel->clear();
                exuImageLabel->clear();
                maImageLabel->clear();
                contrastValue->clear();
                homogeneityValue->clear();
                correlationValue->clear();
                energyValue->clear();
                resultText->clear();

                classifyButton->setEnabled(true);
                clearButton->setEnabled(true);
                exportButton->setEnabled(false);
        }

        return;
}

/**
*Sets up the model and passes
*the image to be processed
*/

void DRDT::classifyImage() {
        imageButton->setEnabled(false);
        classifyButton->setEnabled(false);
        clearButton->setEnabled(false);
        exportButton->setEnabled(false);

        if (imagePath->text() != "") {
                movie->start();
                movieLabel->show();

                QString qtModel = QCoreApplication::applicationDirPath().append("/TrainingResults.
                    xml");
                QFile::copy(":/model/TrainingResults.xml", qtModel);
                svm.load(qtModel.toStdString().c_str());
                QFile::remove(qtModel);

                imageLoc = imagePath->text().toStdString();

                future = QtConcurrent::run(processImage, imageLoc);
                this->watcher.setFuture(future);
        }

        return;
}

/**
*Uses the features extracted
*from the image and predicts
```

```
*the result using svm and
*displays the result
*/

void DRDT::finishedImageProcessing() {
        movie->stop();
        movieLabel->hide();

        imageButton->setEnabled(true);
        clearButton->setEnabled(true);

        if (future.result().empty()) {
                QMessageBox::critical(this, tr("No image selected"),
                        tr("Please select an image file to classify."));

        } else {
                exportButton->setEnabled(true);

                vector<float> features;
                vector<Mat> processedImage = future.result();
                vector<float> glcmFeatures = getGLCM(imageLoc);

                /**
                *Feature extraction
                */

                features.push_back(countNonZero(processedImage.at(1)));
                features.push_back(countNonZero(processedImage.at(2)));
                features.push_back(countNonZero(processedImage.at(3)));
                features.push_back(glcmFeatures.at(0));
                features.push_back(glcmFeatures.at(1));
                features.push_back(glcmFeatures.at(2));
                features.push_back(glcmFeatures.at(3));

                /**
                *Predicts the classification
                *using svm
                */

                Mat featuresMat(1, features.size(), CV_32FC1, &features[0]);
                float response = svm.predict(featuresMat);

                string classification;
                if (response == 0)
                        classification = "Normal";
                else
                        classification = "DR";

                /**
                *Display images and result
                */

                QPixmap fundusPixmap = convertMatToQpixmap(processedImage.at(0), QImage::
                    Format_RGB888);
                fundusImageLabel->setPixmap(fundusPixmap);
                QPixmap bvPixmap = convertMatToQpixmap(processedImage.at(1), QImage::
                    Format_Indexed8);
                bvImageLabel->setPixmap(bvPixmap);
                QPixmap exuPixmap = convertMatToQpixmap(processedImage.at(2), QImage::
                    Format_Indexed8);
                exuImageLabel->setPixmap(exuPixmap);
                QPixmap maPixmap = convertMatToQpixmap(processedImage.at(3), QImage::
                    Format_Indexed8);
                maImageLabel->setPixmap(maPixmap);

                contrastValue->setText(QString::number(glcmFeatures.at(0)));
                homogeneityValue->setText(QString::number(glcmFeatures.at(1)));
                correlationValue->setText(QString::number(glcmFeatures.at(2)));
                energyValue->setText(QString::number(glcmFeatures.at(3)));

                resultText->setText(classification.c_str());
        }

        return;
}

/**
*Clears the contents of the
*classification tab
*/

void DRDT::clearContents() {
        imagePath->clear();
        fundusImageLabel->clear();
        bvImageLabel->clear();
        exuImageLabel->clear();
        maImageLabel->clear();
        contrastValue->clear();
        homogeneityValue->clear();
        correlationValue->clear();
        energyValue->clear();
        resultText->clear();
```

```
        clearButton->setEnabled(false);
        classifyButton->setEnabled(false);
        exportButton->setEnabled(false);

        return;
}

/**
*Exports the results into a
*PDF
*/

void DRDT::exportResults() {
        QString saveDir = QFileDialog::getSaveFileName(this, tr("Choose where to save file"),
                QDir::rootPath(), tr("PDF (*.pdf)"));

        if (saveDir != "") {
                QPrinter printer(QPrinter::HighResolution);

                printer.setPageSize(QPrinter::Letter);
                printer.setOutputFileName(saveDir);
                printer.setOutputFormat(QPrinter::PdfFormat);

                QPainter painter;
                if (painter.begin(&printer)) {
                        painter.setFont(QFont("Tahoma", 22, QFont::Bold));
                        painter.drawText(1900, 2000, "Diabetic Retinopathy Detection Tool");
                        QFont a;

                        a.setUnderline(true);
                        a.setFamily("Tahoma");
                        a.setPointSize(20);
                        painter.setFont(a);
                        painter.drawText(3600, 2500, "Classification Result");

                        painter.setFont(QFont("Tahoma", 11));
                        painter.drawText(1500, 3300, "Image Filename:    " + imagePath->text());

                        painter.setFont(QFont("Tahoma", 15, QFont::StyleItalic));
                        painter.drawText(2100, 3900, "SVM Model Parameters");

                        painter.setFont(QFont("Tahoma", 11));
                        int svmType = svm.get_params().svm_type;
                        switch (svmType) {
                        case 100:
                                painter.drawText(2100, 4300, "SVM Type:     C_SVC");
                                break;
                        case 101:
                                painter.drawText(2100, 4300, "SVM Type:     NU_SVC");
                                break;
                        case 102:
                                painter.drawText(2100, 4300, "SVM Type:     ONE_CLASS");
                                break;
                        case 103:
                                painter.drawText(2100, 4300, "SVM Type:     EPS_SVR");
                                break;
                        case 104:
                                painter.drawText(2100, 4300, "SVM Type:     NU_SVR");
                                break;
                        }

                        int kernel = svm.get_params().kernel_type;
                        switch (kernel) {
                        case 0:
                                painter.drawText(2100, 4600, "Kernel:          Linear");
                                break;
                        case 1:
                                painter.drawText(2100, 4600, "Kernel:          Polynomial");
                                break;
                        case 2:
                                painter.drawText(2100, 4600, "Kernel:          Radial Basis
                                    Function");
                                break;
                        case 3:
                                painter.drawText(2100, 4600, "Kernel:          Sigmoid");
                                break;
                        case 4:
                                painter.drawText(2100, 4600, "Kernel:          Chi-squared");
                                break;
                        case 5:
                                painter.drawText(2100, 4600, "Kernel:          Intersection");
                                break;
                        default:
                                painter.drawText(2100, 4600, "Kernel:          Custom");
                                break;
                        }

                        string gamma = to_string(svm.get_params().gamma);
                        painter.drawText(2100, 4900, "Gamma:         " + QString::fromStdString(
                            gamma));
```

```
                        string cValue = to_string(svm.get_params().C);
                        painter.drawText(2100, 5200, "C:                  " + QString::fromStdString
                            (cValue));

                        painter.setFont(QFont("Tahoma", 15));
                        painter.drawText(5700, 3900, "GLCM Descriptors");

                        painter.setFont(QFont("Tahoma", 11));
                        painter.drawText(5700, 4300, "Contrast:          " + contrastValue->text());
                        painter.drawText(5700, 4600, "Homogeneity: " + homogeneityValue->text());
                        painter.drawText(5700, 4900, "Correlation:     " + correlationValue->text()
                            );
                        painter.drawText(5700, 5200, "Energy:            " + energyValue->text());

                        painter.drawPixmap(2100, 6200, *fundusImageLabel->pixmap());
                        painter.drawText(2100, 8000, "Original Image");

                        painter.drawPixmap(5500, 6200, *bvImageLabel->pixmap());
                        painter.drawText(5500, 8000, "Blood Vessels Image");

                        painter.drawPixmap(2100, 8500, *exuImageLabel->pixmap());
                        painter.drawText(2100, 10300, "Exudates Image");

                        painter.drawPixmap(5500, 8500, *maImageLabel->pixmap());
                        painter.drawText(5500, 10300, "Microaneurysms Image");

                        painter.setFont(QFont("Tahoma", 20));
                        if (resultText->text() == "DR") {
                                painter.drawText(3960, 11500, "Result:");
                                painter.setFont(QFont("Tahoma", 20, QFont::Bold));
                                painter.drawText(5260, 11500, "DR");
                                painter.drawRoundedRect(3680, 11000, 2500, 800, 25, 25);
                        } else {
                                painter.drawText(3780, 11500, "Result:");
                                painter.setFont(QFont("Tahoma", 20, QFont::Bold));
                                painter.drawText(4980, 11500, "Normal");
                                painter.drawRoundedRect(3450, 11000, 3010, 800, 25, 25);
                        }
                        painter.end();
                }
        }

        return;
}

/**
*Sets up the first dr video
*/

void DRDT::drVideoSelected() {
        drPlayButton->setEnabled(false);
        dr2PlayButton->setEnabled(true);
        svmPlayButton->setEnabled(true);
        mediaPlayer->stop();
        mediaPlayer->setMedia(QUrl::fromLocalFile("Videos/Animation- Diabetic Retinopathy.mp4"));
        playButton->setEnabled(true);
        pauseButton->setEnabled(false);
        stopButton->setEnabled(false);

        return;
}

/**
*Sets up the second dr video
*/

void DRDT::dr2VideoSelected() {
        drPlayButton->setEnabled(true);
        dr2PlayButton->setEnabled(false);
        svmPlayButton->setEnabled(true);
        mediaPlayer->stop();
        mediaPlayer->setMedia(QUrl::fromLocalFile("Videos/Diabetic retinopathy.mp4"));
        playButton->setEnabled(true);
        pauseButton->setEnabled(false);
        stopButton->setEnabled(false);

        return;
}

/**
*Sets up the svm video
*/

void DRDT::svmVideoSelected() {
        drPlayButton->setEnabled(true);
        dr2PlayButton->setEnabled(true);
        svmPlayButton->setEnabled(false);
        mediaPlayer->stop();
        mediaPlayer->setMedia(QUrl::fromLocalFile("Videos/How SVM (Support Vector Machine)
            algorithm works.mp4"));
        playButton->setEnabled(true);
        pauseButton->setEnabled(false);
```

```
        stopButton->setEnabled(false);

        return;
}

/**
*Plays the video currently
*selected
*/

void DRDT::playVideo() {
        mediaPlayer->play();
        playButton->setEnabled(false);
        pauseButton->setEnabled(true);
        stopButton->setEnabled(true);

        return;
}

/**
*Pauses the video currently
*selected */

void DRDT::pauseVideo() {
        mediaPlayer->pause();
        playButton->setEnabled(true);
        pauseButton->setEnabled(false);
        stopButton->setEnabled(true);

        return;
}

/**
*Stops the video currently
*selected
*/

void DRDT::stopVideo() {
        mediaPlayer->stop();
        playButton->setEnabled(true);
        pauseButton->setEnabled(false);
        stopButton->setEnabled(false);
}

/**
*Signals the video is already
*done
*/

void DRDT::finishedVideo() {
        if (mediaPlayer->state() == 0) {
                playButton->setEnabled(true);
        }
}

/**
*Default DRDT destructor
*/

DRDT::~DRDT() {

}


/**
*ExudatesDetection.cpp
*This file contains all functions
*exclusive for segmenting the
*exudates from the fundus image
*/

#include "drdt.h"

#include <opencv2\legacy\legacy.hpp>

using namespace cv;

/**
*Converts the RGB image into
*HSI (hue-saturation-intensity)
*parameters:
*image - image to be converted
*/

Mat convertRGBToHSI(Mat image) {
        float red, green, blue, hue, saturation, intensity;
        Mat HSIImage(image.rows, image.cols, image.type());

        for (int i = 0; i < image.rows; i++) {
                for (int j = 0; j < image.cols; j++) {
                        blue = image.at<Vec3b>(i, j)[0];
                        green = image.at<Vec3b>(i, j)[1];
```

```
                        red = image.at<Vec3b>(i, j)[2];

                        intensity = (blue + green + red) / 3;

                        int minVal = min(red, min(blue, green));
                        saturation = 1 - 3 * (minVal / (blue + green + red));

                        if (saturation < 0.00001)
                                saturation = 0;
                        else if (saturation > 0.99999)
                                saturation = 1;

                        if (saturation != 0) {
                                hue = 0.5 * ((red - green) + (red - blue))
                                        / sqrt(((red - green)*(red - green))
                                        + ((red - blue)*(green - blue)));
                                hue = acos(hue);

                                if (blue > green)
                                        hue = ((360 * 3.14159265) / 180.0) - hue;
                        }

                        HSIImage.at<Vec3b>(i, j)[0] = (hue * 180) / 3.14159265;
                        HSIImage.at<Vec3b>(i, j)[1] = saturation * 100;
                        HSIImage.at<Vec3b>(i, j)[2] = intensity;
                }
        }

        return HSIImage;
}

/**
*Splits the image and gets
*the intensity channel
*parameters:
*image - image where the
*intensity channel is extracted
*/

Mat getIntensityChannel(Mat src) {
        vector<Mat> channel;

        Mat HSIImage = convertRGBToHSI(src);
        split(src, channel);

        return channel[1];
}

/**
*Performs dilation on an image
*parameters:
*src - image to be dilated
*morphSize - size of dilation
*/

Mat dilatingOperation(Mat src, int morphSize) {
        Mat dilatedImage;
        int morphElement = 2;   //disc or ellipse

        Mat element = getStructuringElement(morphElement,
                Size(2 * morphSize + 1, 2 * morphSize + 1),
                Point(morphSize, morphSize));

        dilate(src, dilatedImage, element);

        return dilatedImage;
}

/**
*Performs geodilation on an
*image
*parameters:
*src - image where geodilation
*will be performed
*mask - image used as mask
*dilationSize - size of dilation
*/

Mat geoDilateImage(Mat src, Mat mask, int dilationSize) {
        Mat dilatedImage = dilatingOperation(src, dilationSize);
        Mat geoDilatedImage = min(dilatedImage, mask);

        return geoDilatedImage;
}

/**
*Morphologically reconstructs
*image using dilation
*parameters:
*src - image to be reconstructed
*mask - image used as mask
dilationSize - size of dilation
```

```
*/

Mat reconstructionByDilation(Mat src, Mat mask, int dilationSize) {
        Mat geoDilatedImage, diff;
        Mat temp = src.clone();

        do {
                geoDilatedImage = temp;
                temp = geoDilateImage(temp, mask, dilationSize);
                diff = temp != geoDilatedImage;
        } while (countNonZero(diff) != 0);

        Mat reconstructedImage = temp;

        return reconstructedImage;
}


/**
*Performs otsu thresholding
*on an image
*parameters:
*src - image to be thresholded
*Original Author: Bharath
*Prabhuswamy
*/

Mat otsuThresholding(Mat src) {
        int hist[256] = { 0 }; //array to store histogram values
        float normHist[256] = { 0 }; //array to store normalized histogram values
        int pixelValue;
        float firstCum = 0; //first order cumulative
        float secondCum = 0; //second order cumulative
        float globalMean = 0; //global mean level
        int optThreshVal = 0; //optimum threshold value
        float param1, param2, param3 = 0; //parameters for OTSU algorithm

        /**
        *creates the histogram
        */

        for (int i = 0; i < src.rows; i++) {
                for (int j = 0; j < src.cols; j++) {
                        pixelValue = src.at<uchar>(i, j);
                        hist[pixelValue]++;
                }
        }

        /**
        *normalizes histogram and
        *calculates global mean level
        */

        for (int i = 0; i < 256; i++) {
                normHist[i] = hist[i] / (float)(src.cols *
                        src.rows);
                globalMean += (i * normHist[i]);
        }

        /**
        *OTSU algorithm
        */

        for (int i = 0; i < 255; i++) {
                firstCum += normHist[i];
                secondCum += (i * normHist[i]);

                param1 = (globalMean * firstCum) - secondCum;
                param2 = (param1 * param1) / (firstCum * (1 - firstCum));

                if (param2 > param3) {
                        param3 = param2;
                        optThreshVal = i;
                }
        }

        Mat otsuImage = binarizeImage(src, optThreshVal);

        return otsuImage;
}

/**
*Computes the standard devition
*of an image
*parameters:
*src - image to be processed
*winSize - size of window
*/

Mat computeStdDevImage(Mat src, int winSize) {
        Mat image32f, mu, mu2, stdDevImage32f, stdDevImage;
```

```
        src.convertTo(image32f, CV_32F);

        blur(image32f, mu, Size(winSize, winSize));
        blur(image32f.mul(image32f), mu2, Size(winSize, winSize));
        sqrt(mu2 - mu.mul(mu), stdDevImage32f);

        stdDevImage32f.convertTo(stdDevImage, CV_8U);

        return stdDevImage;
}

/**
*Performs erosion on an image
*parameters:
*src - image to be eroded
*morphSize - size of erosion
*/

Mat erodingOperation(Mat src, int morphSize) {
        Mat erodedImage;
        int morphElement = 2;   //disc or ellipse

        Mat element = getStructuringElement(morphElement,
                Size(2 * morphSize + 1, 2 * morphSize + 1),
                Point(morphSize, morphSize));

        erode(src, erodedImage, element);

        return erodedImage;
}

/**
*Fills the holes of an image
*using floodfill
*parameters:
*src - image to be filled
*mask - mask used for the
*floodfill operation
*Original author: HD_Mouse
*/

Mat fillImageHoles(Mat src, Mat mask) {

        //remove the white boundary on the image
        Mat holesFilledImage = src.clone();
        Mat erodedMask = mask.clone();
        erodedMask = erodingOperation(erodedMask, 30); //30
        bitwise_and(holesFilledImage, erodedMask, holesFilledImage);

        Mat newMask;
        holesFilledImage.copyTo(newMask);

        for (int i = 0; i < newMask.cols; i++) {
                if (newMask.at<char>(0, i) == 0) {
                        floodFill(newMask, Point(i, 0), 255, 0, 10, 10);
                }

                if (newMask.at<char>(newMask.rows - 1, i) == 0) {
                        floodFill(newMask, Point(i, newMask.rows - 1), 255, 0, 10, 10);
                }
        }

        for (int i = 0; i < newMask.rows; i++) {
                if (newMask.at<char>(i, 0) == 0) {
                        floodFill(newMask, Point(0, i), 255, 0, 10, 10);
                }

                if (newMask.at<char>(i, newMask.cols - 1) == 0) {
                        floodFill(newMask, Point(newMask.cols - 1, i), 255, 0, 10, 10);
                }
        }

        for (int row = 0; row < newMask.rows; ++row) {
                for (int col = 0; col < newMask.cols; ++col) {
                        if (newMask.at<char>(row, col) == 0) {
                                holesFilledImage.at<char>(row, col) = 255;
                        }
                }
        }

        return holesFilledImage;
}

/**
*Main function which handles
*the segmentation of exudates
*parameters:
*src - image to be segmented
*mask - mask image
*/

Mat exudatesDetection(Mat src, Mat mask) {
```

```
        Mat intensityChannelImage = getIntensityChannel(src);
        Mat medianFilterImage = medianFilter(intensityChannelImage, 15);
        Mat contrastLimitedAHEImage = contrastLimitedAHE(medianFilterImage);
        Mat closedImage = closingOperation(contrastLimitedAHEImage, 7);
        Mat binarizedImageMask = binarizeImage(closedImage, 150);
        Mat markerImage = contrastLimitedAHEImage - binarizedImageMask;
        Mat reconstructedByDilationImage = reconstructionByDilation(markerImage,
             contrastLimitedAHEImage, 2);
        Mat differenceImage = contrastLimitedAHEImage - reconstructedByDilationImage;
        Mat otsuImage = otsuThresholding(differenceImage);
        Mat opticDiscImage = removeSmallerDetails(otsuImage, 3000, 1);
        Mat dilatedOpticDiscImage = dilatingOperation(opticDiscImage, 5);

        Mat closedImage2 = closingOperation(contrastLimitedAHEImage, 7);
        Mat stdDevImage = computeStdDevImage(closedImage2, 40);
        Mat otsuStdDevImage = otsuThresholding(stdDevImage);
        Mat dilatedStdDevImage = dilatingOperation(otsuStdDevImage, 5);
        Mat holesFilledImage = fillImageHoles(dilatedStdDevImage, mask);
        Mat dilatedOpticDiscImage2 = dilatingOperation(dilatedOpticDiscImage, 30);
        Mat woOpticDiscImage = holesFilledImage - dilatedOpticDiscImage2;
        Mat markerImage2 = contrastLimitedAHEImage - woOpticDiscImage;
        Mat reconstructedByDilationImage2 = reconstructionByDilation(markerImage2,
             contrastLimitedAHEImage, 2);
        Mat differenceImage2 = contrastLimitedAHEImage - reconstructedByDilationImage2;
        Mat exudatesImage = otsuThresholding(differenceImage2);

        return exudatesImage;
}

/**
*Glcm.cpp
*This file handles the extraction
*of the textural features in an
*image
*/

#include "drdt.h"

#include <opencv2\highgui\highgui.hpp>
#include <opencv2\legacy\legacy.hpp>

using namespace cv;
using namespace std;

/**
*Computes the GLCM contrast,
*homogeneity, correlation
*and energy of an image
*parameters:
*imageLoc - location of image
*/

vector<float> getGLCM(string imageLoc) {
        Mat gray;
        int stepsize = 0;

        Mat src = imread(imageLoc);

        cvtColor(src, gray, CV_BGR2GRAY);
        IplImage grayIm = gray;

        CvGLCM *glcm = cvCreateGLCM(&grayIm, 1, NULL, 4, CV_GLCM_OPTIMIZATION_LUT);                //
             create the GLCM
        cvCreateGLCMDescriptors(glcm, CV_GLCMDESC_OPTIMIZATION_ALLOWDOUBLENEST);                   //
             create GLCM descriptors

        double contrast = cvGetGLCMDescriptor(glcm, stepsize, CV_GLCMDESC_CONTRAST);
        double homogeneity = cvGetGLCMDescriptor(glcm, stepsize, CV_GLCMDESC_HOMOGENITY);
        double correlation = cvGetGLCMDescriptor(glcm, stepsize, CV_GLCMDESC_CORRELATION);
        double energy = cvGetGLCMDescriptor(glcm, stepsize, CV_GLCMDESC_ENERGY);

        vector<float> features(4);
        features[0] = contrast;
        features[1] = homogeneity;
        features[2] = correlation;
        features[3] = energy;

        return features;
}

/**
*ImageProcessing.cpp
*This file has the common image
*processing functions used in
*other files and connects the
*segmentation of the retinal
*structures
*/

#include "drdt.h"
```

```
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

/**
*Segments the background
*of a fundus image
*parameters:
*src - image to be processed
*/

Mat imageMask(Mat src) {
        Mat maskImage;
        int thresh = 15;        //15

        cvtColor(src, maskImage, CV_RGB2GRAY);
        maskImage = binarizeImage(maskImage, thresh);

        return maskImage;
}

/**
*Binarizes an image according
*to a thresh specified
*parameters:
*src - image to be binarized
*thresh - threshold for the image
*/

Mat binarizeImage(Mat src, int thresh) {
        Mat binaryImage;
        int maxValue = 255;
        int threshType = 0;     //thresh binary

        threshold(src, binaryImage, thresh, maxValue, threshType);

        return binaryImage;
}

/**
*Performs median filter on an
*image
*parameters:
*src - image to be processed
*maxKernelLength - max possible
length of kernel
*/

Mat medianFilter(Mat src, int maxKernelLength) {
        Mat medianFilterImage;

        for (int i = 1; i < maxKernelLength; i = i + 2) {
                medianBlur(src, medianFilterImage, i);
        }

        return medianFilterImage;
}

/**
*Performs the CLAHE algorithm
*on an image
*parameters:
*src - image to be processed
*/

Mat contrastLimitedAHE(Mat src) {
        Mat contrastLimitedAHEImage;

        Ptr<CLAHE> clahe = createCLAHE();
        clahe->setClipLimit(2);
        clahe->apply(src, contrastLimitedAHEImage);

        return contrastLimitedAHEImage;
}

/**
*Performs closing operation on
*an image
*parameters:
*src - image to be closed
*morphSize - size of closing operator
*/

Mat closingOperation(Mat src, int morphSize) {
        Mat closedImage;
        int morphOperation = 3; //closing
        int morphElement = 0;   //rectangle
```

```
        Mat element = getStructuringElement(morphElement,
                Size(2 * morphSize + 1, 2 * morphSize + 1),
                Point(morphSize, morphSize));
        morphologyEx(src, closedImage, morphOperation, element);

        return closedImage;
}

/**
*Removes unwanted small details
*on an image using contours
*parameters:
*src - image to be processed
*sizeOfDetail - minimum size
*of contour that will be retained
*thresh - threshold
*/

Mat removeSmallerDetails(Mat src, int sizeOfDetail, int thresh) {
        vector<vector<Point>> contours;
        double area;

        Mat threshImage = binarizeImage(src, thresh);
        Mat dst = threshImage.clone();

        // Find all contours
        findContours(dst.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

        for (int i = 0; i < contours.size(); i++) {
                area = contourArea(contours[i]);

                // Remove small objects by drawing the contour with black color
                if (area > 0 && area <= sizeOfDetail)
                        drawContours(dst, contours, i, CV_RGB(0, 0, 0), -1);
        }

        return dst;
}

/**
*Function which handles the
*segmentation of the retinal
*structures of an image loaded
*in the tool
*parameters:
*imageLoc - location of the image
*/

vector<Mat> processImage(string imageLoc) {
        vector<Mat> processedImage;

        Mat image = imread(imageLoc);
        if (!image.data) {
                cout << "Could not open or find the image" << endl;
                return processedImage;
        }

        Mat maskImage = imageMask(image);

        /**
        *Segmentation of retinal structures
        */

        Mat exudatesImage = exudatesDetection(image, maskImage);
        Mat bvBasicStructureImage = bloodVesselsPreDetection(image, maskImage, exudatesImage);
        Mat bvAndMaImage = microaneurysmsPreDetection(image, maskImage);
        Mat bloodVesselsImage = averageBloodVessels(bvBasicStructureImage, bvAndMaImage) -
             exudatesImage;
        Mat microaneurysmsImage = bvAndMaImage - bloodVesselsImage - exudatesImage;

        processedImage.push_back(image);
        processedImage.push_back(bloodVesselsImage);
        processedImage.push_back(exudatesImage);
        processedImage.push_back(microaneurysmsImage);

        return processedImage;
}

/**
*Converts a Mat image into
*pixmap to be displayed in
*the GUI
*parameters:
*src - image to be converted
*format - format of the image
*/

QPixmap convertMatToQpixmap(Mat src, QImage::Format format) {
        if (format == QImage::Format_RGB888) {
                QImage qImage(src.data, src.cols, src.rows, src.step, QImage::Format_RGB888);
```

```
                return QPixmap::fromImage(qImage.rgbSwapped());
        } else if (format == QImage::Format_Indexed8) {
                static QVector<QRgb>  colorTable;

                if (colorTable.isEmpty()) {
                        for (int i = 0; i < 256; ++i)
                                colorTable.push_back(qRgb(i, i, i));
                }

                QImage qImage(src.data, src.cols, src.rows, src.step, QImage::Format_Indexed8);
                qImage.setColorTable(colorTable);

                return QPixmap::fromImage(qImage);
        } else {
                QImage qImage(src.data, src.cols, src.rows, src.step, QImage::Format_Indexed8);

                return QPixmap::fromImage(qImage);
        }
}


/**
*Launcher.cpp
*This file contains the main
*function which creates the
*QApplication
*/

#include "windows.h"
#include "drdt.h"

#include <QtCore/qcoreapplication.h>
#include <QtCore/qplugin.h>
#include <QtCore/qresource.h>

/**
*Import the plugins that will
*be used
*/

Q_IMPORT_PLUGIN(QWindowsIntegrationPlugin)
Q_IMPORT_PLUGIN(DSServicePlugin)


int main(int argc, char* argv[]) {
        ShowWindow(GetConsoleWindow(), SW_HIDE);

        QApplication app(argc, argv);

        //Initialize the resources
        Q_INIT_RESOURCE(resources);

        DRDT drdt;
        drdt.show();

        return app.exec();
}


/**
*MicroaneurysmsDetection.cpp
*This file contains all functions
*exclusive for segmenting the
*microaneurysms and smaller
*blood vessels from the fundus
*image
*/

#include "drdt.h"

#include <opencv2\highgui\highgui.hpp>
#include <opencv2\legacy\legacy.hpp>

using namespace cv;

/**
*Returns the bright regions in an image
*parameters:
*src - image to be processed
*morphSize - size of closing operator
*/

Mat getBrightRegions(Mat src, int morphSize, int thresh) {

        Mat brightRegions = closingOperation(src, morphSize);
        brightRegions = binarizeImage(brightRegions, thresh);

        return brightRegions;
}

/**
*Perform filling operation on
*the image
```

```
*parameters:
*src - image to be filled
*bg - image background
*/

Mat fillingOperation(Mat src, Mat bg) {
        Mat filledImage;
        int thresh = 50;
        int fillValue = 50;

        src.copyTo(filledImage);

        // create a mask where to fill
        Mat threshImage = binarizeImage(src, thresh);
        bitwise_not(threshImage, threshImage);
        bitwise_and(threshImage, bg, threshImage);

        bitwise_and(src, Scalar(0), filledImage, threshImage);
        bitwise_or(filledImage, Scalar(fillValue), filledImage, threshImage);

        return filledImage;
}

/**
*Functions which detects the
*microaneurysms along with the
*small blood vessels in the retina
*parameters:
*src - image to be processed
*maskImage - mask image
*/

Mat microaneurysmsPreDetection(Mat src, Mat maskImage) {
        Mat grayscaleImage;
        cvtColor(src, grayscaleImage, CV_BGR2GRAY);
        Mat contrastLimitedAHEImage = contrastLimitedAHE(grayscaleImage);
        Mat filteredCLAHEImage;
        bilateralFilter(contrastLimitedAHEImage, filteredCLAHEImage, 9, 75, 75); //diameter = 9
             sigmaColor = 75 sigmaSpace = 75
        bitwise_and(filteredCLAHEImage, maskImage, filteredCLAHEImage); //put the black bg
        Mat closedMaImage = closingOperation(filteredCLAHEImage, 20); //morphsize = 20
        Mat brightRegionsImage = getBrightRegions(filteredCLAHEImage, 45, 120); // morphsize = 45
             thresh = 120
        Mat nonBrightClosedMaImage = closedMaImage - brightRegionsImage;
        Mat filledMaImage = fillingOperation(filteredCLAHEImage, maskImage);
        Mat noisyMaImage = nonBrightClosedMaImage - filledMaImage;
        Mat bvAndMaImage = removeSmallerDetails(noisyMaImage, 50, 20); //sizeOfDetail = 50 thresh
             = 20

        return bvAndMaImage;
}


/****************************************************************************
** Meta object code from reading C++ file 'drdt.h'
**
** Created by: The Qt Meta Object Compiler version 67 (Qt 5.5.0)
**
** WARNING! All changes made in this file will be lost!
*****************************************************************************/

#include "../../drdt.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
#if !defined(Q_MOC_OUTPUT_REVISION)
#error "The header file 'drdt.h' doesn't include <QObject>."
#elif Q_MOC_OUTPUT_REVISION != 67
#error "This file was generated using the moc from 5.5.0. It"
#error "cannot be used with the include files from this version of Qt."
#error "(The moc has changed too much.)"
#endif

QT_BEGIN_MOC_NAMESPACE
struct qt_meta_stringdata_DRDT_t {
    QByteArrayData data[19];
    char stringdata0[260];
};
#define QT_MOC_LITERAL(idx, ofs, len) \
    Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_DRDT_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
static const qt_meta_stringdata_DRDT_t qt_meta_stringdata_DRDT = {
    {
QT_MOC_LITERAL(0, 0, 4), // "DRDT"
QT_MOC_LITERAL(1, 5, 19), // "switchToClassifyTab"
QT_MOC_LITERAL(2, 25, 0), // ""
QT_MOC_LITERAL(3, 26, 20), // "switchToTutorialsTab"
QT_MOC_LITERAL(4, 47, 14), // "switchToFAQTab"
QT_MOC_LITERAL(5, 62, 7), // "quitApp"
QT_MOC_LITERAL(6, 70, 15), // "switchToHomeTab"
QT_MOC_LITERAL(7, 86, 12), // "getImagePath"
```

```
QT_MOC_LITERAL(8, 99, 13), // "classifyImage"
QT_MOC_LITERAL(9, 113, 23), // "finishedImageProcessing"
QT_MOC_LITERAL(10, 137, 13), // "clearContents"
QT_MOC_LITERAL(11, 151, 13), // "exportResults"
QT_MOC_LITERAL(12, 165, 15), // "drVideoSelected"
QT_MOC_LITERAL(13, 181, 16), // "dr2VideoSelected"
QT_MOC_LITERAL(14, 198, 16), // "svmVideoSelected"
QT_MOC_LITERAL(15, 215, 9), // "playVideo"
QT_MOC_LITERAL(16, 225, 10), // "pauseVideo"
QT_MOC_LITERAL(17, 236, 9), // "stopVideo"
QT_MOC_LITERAL(18, 246, 13) // "finishedVideo"

    },
    "DRDT\0switchToClassifyTab\0\0"
    "switchToTutorialsTab\0switchToFAQTab\0"
    "quitApp\0switchToHomeTab\0getImagePath\0"
    "classifyImage\0finishedImageProcessing\0"
    "clearContents\0exportResults\0drVideoSelected\0"
    "dr2VideoSelected\0svmVideoSelected\0"
    "playVideo\0pauseVideo\0stopVideo\0"
    "finishedVideo"
};
#undef QT_MOC_LITERAL

static const uint qt_meta_data_DRDT[] = {

 // content:
       7,       // revision
       0,       // classname
       0,    0, // classinfo
      17,   14, // methods
       0,    0, // properties
       0,    0, // enums/sets
       0,    0, // constructors
       0,       // flags
       0,       // signalCount

 // slots: name, argc, parameters, tag, flags
       1,    0,   99,    2, 0x08 /* Private */,
       3,    0,  100,    2, 0x08 /* Private */,
       4,    0,  101,    2, 0x08 /* Private */,
       5,    0,  102,    2, 0x08 /* Private */,
       6,    0,  103,    2, 0x08 /* Private */,
       7,    0,  104,    2, 0x08 /* Private */,
       8,    0,  105,    2, 0x08 /* Private */,
       9,    0,  106,    2, 0x08 /* Private */,
      10,    0,  107,    2, 0x08 /* Private */,
      11,    0,  108,    2, 0x08 /* Private */,
      12,    0,  109,    2, 0x08 /* Private */,
      13,    0,  110,    2, 0x08 /* Private */,
      14,    0,  111,    2, 0x08 /* Private */,
      15,    0,  112,    2, 0x08 /* Private */,
      16,    0,  113,    2, 0x08 /* Private */,
      17,    0,  114,    2, 0x08 /* Private */,
      18,    0,  115,    2, 0x08 /* Private */,

 // slots: parameters
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,
    QMetaType::Void,

       0        // eod
};

void DRDT::qt_static_metacall(QObject *_o, QMetaObject::Call _c, int _id, void **_a)
{
    if (_c == QMetaObject::InvokeMetaMethod) {
        DRDT *_t = static_cast<DRDT *>(_o);
        Q_UNUSED(_t)
        switch (_id) {
        case 0: _t->switchToClassifyTab(); break;
        case 1: _t->switchToTutorialsTab(); break;
        case 2: _t->switchToFAQTab(); break;
        case 3: _t->quitApp(); break;
        case 4: _t->switchToHomeTab(); break;
        case 5: _t->getImagePath(); break;
        case 6: _t->classifyImage(); break;
        case 7: _t->finishedImageProcessing(); break;
```

```
        case 8: _t->clearContents(); break;
        case 9: _t->exportResults(); break;
        case 10: _t->drVideoSelected(); break;
        case 11: _t->dr2VideoSelected(); break;
        case 12: _t->svmVideoSelected(); break;
        case 13: _t->playVideo(); break;
        case 14: _t->pauseVideo(); break;
        case 15: _t->stopVideo(); break;
        case 16: _t->finishedVideo(); break;
        default: ;
        }
    }
    Q_UNUSED(_a);
}

const QMetaObject DRDT::staticMetaObject = {
    { &QMainWindow::staticMetaObject, qt_meta_stringdata_DRDT.data,
      qt_meta_data_DRDT,  qt_static_metacall, Q_NULLPTR, Q_NULLPTR}
};


const QMetaObject *DRDT::metaObject() const
{
    return QObject::d_ptr->metaObject ? QObject::d_ptr->dynamicMetaObject() : &staticMetaObject;
}

void *DRDT::qt_metacast(const char *_clname)
{
    if (!_clname) return Q_NULLPTR;
    if (!strcmp(_clname, qt_meta_stringdata_DRDT.stringdata0))
        return static_cast<void*>(const_cast< DRDT*>(this));
    return QMainWindow::qt_metacast(_clname);
}

int DRDT::qt_metacall(QMetaObject::Call _c, int _id, void **_a)
{
    _id = QMainWindow::qt_metacall(_c, _id, _a);
    if (_id < 0)
        return _id;
    if (_c == QMetaObject::InvokeMetaMethod) {
        if (_id < 17)
            qt_static_metacall(this, _c, _id, _a);
        _id -= 17;
    } else if (_c == QMetaObject::RegisterMethodArgumentMetaType) {
        if (_id < 17)
            *reinterpret_cast<int*>(_a[0]) = -1;
        _id -= 17;
    }
    return _id;
}
QT_END_MOC_NAMESPACE
```

XI. Acknowledgement

I would like to express my deepest gratitude to the higher being that looks after me. It might have been a bumpy ride, but You made sure that I reached my destination. I thought I should give up and stop asking You, but I'm glad I didn't. Thank you for answering my prayers and giving me strength to finish my SP.

I want to give thanks to my supportive parents who witnessed me shed tears and blood for this final requirement. Mama, thank you for encouraging me to do my best and for always worrying about me. Papa, thank you for telling me to take it easy sometimes and for assuring me that not graduating on time does not make me less of a beautiful person. Pa and Ma, thank you for waking me up during the times that I badly need to. Remember that I will forever be indebted to you.

I also want to acknowledge my adviser Ma'am Perl, who played a huge part in this work. She guided me throughout the duration of my SP. Ma'am, thank you for extending your patience in replying to my numerous texts and emails. I'm very grateful you didn't get sick of answering my inquiries. Thank you for helping me make it through my defense. To Sir Marvin, for giving me an idea for a topic, and Sir Gonzaga, for helping me understand a topic I'm not familiar to, thank you very much. I couldn't have done it without these people.

To netizens who extended their help concerning my SP, thank you. I also want to say thanks to Mr. Korting who helped me understand my SP better when I was just starting out.

I want to convey my gratitude to my best friends who are always there to support me and make thesis-making less stressful. Kriselle, Sheila and Lei, thank you for taking time to answer SP related questions to help me even though you three are very

busy. You girls are my happy pill during stressful days.

Thank you Larisse, Jaimee and ate Jaira for spending time making SP with me. Those 13 days were really productive in a lot of ways! I also want to thank tita Happy for making this possible.

To my family, friends, and Block 12, you were my inspiration in doing this SP.

Lastly, I would like to congratulate you for seeing this to the end and finishing it. Your efforts are all worth it. I told you, you can do it Christine!