Optical Coherence Tomography Image Classification

DIP/Computer Vision

Computer and Information Sciences

05/19/2018

Caleb Sarikey & Tim Densmore

Supervised by Dr. Gang Hu

Summary

Using C++ and OpenCV, we aim to analyze Optical Coherence Tomography (OCT) scan images to classify them as either normal or having age-related macular degeneration (AMD), a prevalent ocular disease. The OCT image dataset contains 283 scans of the central retina. Our programs will focus on segmenting and analyzing the inner limiting membrane, which is the layer affected by AMD and is represented in the image by the strong red pixels in the middle. If the inner limiting membrane line has a "hump" in it, this indicates a drusen and the potential presence of AMD. We use a Support Vector Machine to train and test data for classification. A comprehensive description of the dataset, our methodology, and our results follows. We conclude by discussing potential improvements moving forward.

Dataset

The dataset of OCT images was acquired from an Optovue iVue 3.2.1.8 machine owned and operated by Dr. Philip Sarikey. The dataset contains 283 OS and OD Retina Map images in total, with 103 of them showing AMD and 180 of them being normal. All of the images have a width of 980 pixels, a height of 275 pixels, and are in .png file format. The files are named {"A1.png", "A2.png", …, "A103.png"} for AMD images and {"N1.png", "N2.png", …, "N180.png"} for normal images. No other patient data has been obtained throughout any phase of this project.

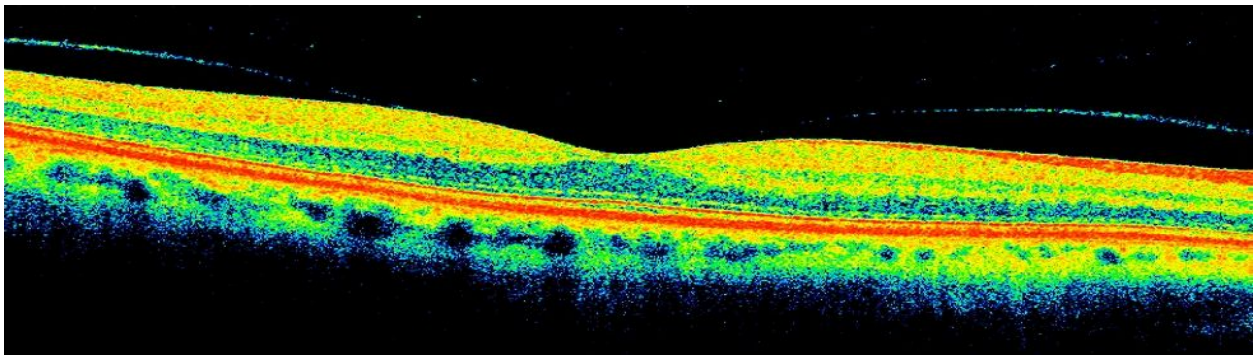

Fig 1.1. *OCT image showing a normal, healthy retina*

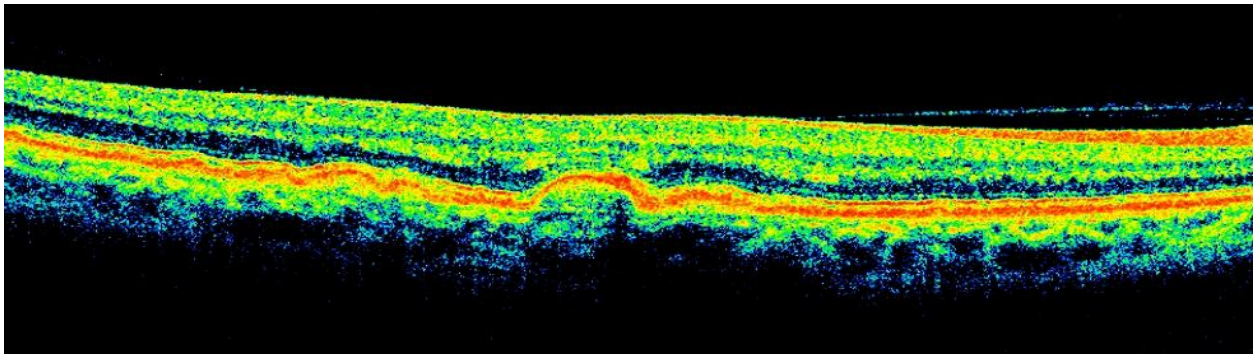

Fig 1.2. *OCT image showing Age-related Macular Degeneration*

Methodology

1. **Read OCT image files one-by-one in color**
    a. We chose to use 75% of the data for training and 25% of the data for testing. This resulted in training 77 AMD images and 135 normal images, and testing 26 AMD images and 45 normal images.
    b. To spread out and 'randomize' the selection of images, every 4th image was chosen for testing while the rest were used for training. For example, the set of testing data for AMD = {A1.png, A5.png, A9.png…, A101.png}.
2. **Extract red pixels from the image**
    a. By applying a binary threshold, we extract the red pixels from the image while making all other pixels black. This removes majority of the irrelevant pixels from our images and leaves us almost with only the inner limiting membrane. There are still red pixels outside of the inner limiting membrane layer, and this is extra noise that we must work around.
3. **Apply Median Blur**
    a. This step is not performed on all images. Rather, it is only necessary when there is a substantial density of red pixels. Some images show a very thin inner limiting membrane, and thus applying a median blur removes too many red pixels, leaving us with almost nothing to analyze. For this reason, we only apply a median blur filter if the number of red pixels exceeds a threshold of 3,500 total red pixels. The

median blur smoothes the image and further reduces noise, making the inner limiting membrane line clearer and more continuous.

4. **Partition the image into vertical windows of size 49 x 275.**

   Within each window:

   a. **Apply a customized Mean-Shift algorithm** to find the center of mass of red pixels. A thin rectangular Mean-Shift window of size 29 x 15 is used. By starting from the bottom of the image and shifting upward to find pixel density, we first reach the red pixels of the inner limiting membrane and thus ignore the noisey, irrelevant red pixels that lie above the inner limiting membrane. As a result, we avoid mapping a best-fit line to the noise rather than the inner limiting membrane in the following step.

   b. **Calculate the line of best-fit.** To calculate a line of best-fit for the red pixels within each window, we first store all of the pixel (x,y) coordinates in a vector. We then take a 'random' sample of 40% of the red pixel points to map lines onto. For each line that is mapped, the line formula (y=mx+b) is used to find the vertical distance of all red pixels from this line. These distances are stored in a vector. After all lines have been mapped, we ultimately choose the line with the lowest distance of red pixels from it and store its slope value.

5. **Calculate the change in slope for each window**

   a. In addition to the slope within each window, we take into account the rate of change of the slope from window to window. The reason is so that the slope values are not affected by a general, gradual curve in the inner limiting

membrane, and rather focus on small, sharp changes in the slope values from window to window. If we have large spikes in the changes of slope values, it is likely that this is the region where AMD occurs, as the slopes of the best-fit lines will change drastically with the curvature of AMD anomalies.

6. **Write training and testing data to text files**
    a. For each image, we produce a vector of 78 floating point values. Each image is divided into vertical windows with a width of 49 pixels. These windows are shifted 24 pixels at a time, resulting in 38 total windows. Within each of the 38 windows, we calculate two values - the slope and the change in slope. This gives us 38*2=76 values. Lastly, we find two additional values - the total sum of the slopes and changes in slopes. This brings us to our final vector of 78 values for each image.
    b. As the data is calculated, the floating point values are written to text files. The process is computationally expensive, so storing the data in text files avoids having to compute the data more than once. The following text files are written:
        i. AMD-Training.txt
        ii. AMD-Testing.txt
        iii. Normal-Training.txt
        iv. Normal-Testing.txt
7. **Read data from files into arrays**
    a. In order to train and test our classifier, we read the training and testing data from the text files and store it into arrays. Each row in the array represents an image.

The 78 columns represent the values calculated for each image. Thus, the arrays are as follows:

i. TrainingData (**212** images) = AMDTraining[**212**][78]

ii. TestingData (**71** images) = AMDTesting[**71**][78]

i.e. Feature vector for "A1.png" corresponds to TestingData[0][k] for k=0; k<78; k++

8. **Train the Support Vector Machine (SVM)**
    a. The arrays of training data are converted to Mat objects that are used to train OpenCV's SVM. Additionally, we setup an array of labels for the training data. AMD is given a label of -1.0 and Normal is given a label of 1.0. We then use OpenCV's functions to train the SVM with our data and their corresponding class labels.
9. **Test the Support Vector Machine (SVM)**
    a. Now that the SVM has been trained, we convert the arrays of testing data into Mat objects to test our SVM. For each image, the SVM will classify the data as either a 1.0 (Normal) or -1.0 (AMD). The classification results are written to a text file, "SVM-Results.txt". These results are used to analyze classification performance.

Results

Of the 26 AMD images tested, 23 of them were correctly classified and 3 of them were misclassified as 1.0. Of the 45 Normal images tested, 40 of them were correctly classified and 5 of them were misclassified.

Precision, recall, and accuracy is computed to assess the performance of our classifier:

True Positive: 23

True Negative: 40

False Positive: 5

False Negative: 3

Precision = 0.82 = 82%

Recall = 0.88 = 88%

Accuracy = 0.89 = 89%

All misclassified images have a distance of less than 1.2 to the hyperplane, with majority of them having a distance less than 0.2, indicating that it was near the border and in an area of "crossover" between the two classes. Improvements suggested in the next section aim to increase precision, recall, and accuracy and decrease misclassification.

Improvements

1. **Mean-Shift** is dependent on window size and thus, this window size should be optimized. We found better results as we reduced the window size, but it may need to be even smaller yet.
2. **Line of best-fit** within each window could be more accurate by taking a better/larger sample of red pixels on which to map lines. The sample size may be increased from the current size of 40%.

3. **Vertical window size needs to be optimized.** Upon analyzing our results, we believe that an even smaller vertical window size would result in better best-fit lines which would more accurately catch small but drastic changes in slope. The size may be decreased from the current size of 49 pixels.
4. **Improve method of removing noise.** It would be ideal to completely remove or ignore all red pixels that are not part of the inner limiting membrane in order to reduce misclassification.
5. **Take into account features other than slope.** Although slope is a good indicator, our classifier may be more accurate with additional features such as thickness or variance.
6. **Obtain a larger dataset.** The dataset we used was adequate for a starting project, but having a large, publicly available dataset would allow us to better train the SVM, better learn the images, and more accurately classify the images.
7. **Compare to a popular, standard method.** To assess the performance of our method, it must be compared to a currently widely used method such as Scale Invariant Feature Transform (SIFT), Histogram of Oriented Gradients (HOG), or Bag Of Visual Words (BOVW). We would expect our specialized method to outperform these standard methods.
8. **Classify other diseases.** In addition to AMD, a variety of ocular diseases can be detected through visual analysis of OCT scans, including but not limited to glaucoma, macular hole, macular edema, diabetic retinopathy, and Central serous chorioretinopathy. Eventually, it would be ideal to provide classification for all of these diseases.