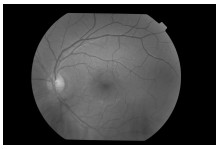
COMPX 301 - Retina Matching Assignment

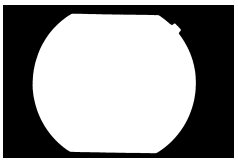
Alexander Stokes - 1578409
Rowan Thorley - 1650315

Retina Matching Pipeline

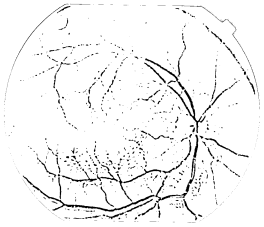
>Convert Image to Grayscale
Grayscale for single channel comparisons



>Generate threshold mask for later
Create the mask around the lens outline so it can be masked out later



>Contrasting
Makes the veins stand out more for easier edge detection on the background
>Blur the grayscale image - Gaussian
Smooth out the artifacts
Gaussian blur returned what looked to be a better result than median blur.
>Find the laplacian of gaussian edges
Experimented with canny, but laplacian returned the best result
>Blur the edges result - Gaussian
Blur the result of the laplacian to smooth irrelevant lines.
Gaussian blur returned a smoother image than that of median blur.
>Threshold based on delta of laplacian
Turn the edge matrix into a white/black image
>Apply mask to image
Remove the black outline - done after thresholding to remove the black outline.
Done before the inversion of the image so the whole background goes to white. (important for template matching so that the thick black outline doesn't impact different versions of the same image where the images are misaligned with where the retina is.
>Invert image
Switch to white background, black foreground to follow colouring conventions in lecture 11 - black foreground, white background.
>Median blur thresholded mat
Removes noise from the thresholded image, and blurs the thresholded image for dilation.
Median blur was used to eliminate some 1 pixel wide noise, as the 1 pixel is not median it was not picked - it was important not to take the average like Gaussian, otherwise the noise would have been preserved.



>Open the image's features
(On windows dilate does what erode should, vice versa for some reason)
Remove more noise from the thresholded image.
Shrinks the noise first, then regrows the remaining structures for a clearer image.

–Template Matching-

>Crop the 2nd image around the largest contour in the mask (Mask generated earlier)

Removes the majority of the white space around the original image.

Uses the boundingRect of the largest contour to become smaller.

>Use the cropped 2nd image as a template to match template over the 1st image

We do not crop the 1st image, as the white space makes for suitable padding for the template matching to work with.

We do not have both images cropped, because it is important the wiggle room is present for the matching.

> Use CCoefficient Normalized template matching method

Max value of the result matrix will be the best match - because it is a normalized method, so that which is closest to 1 is the best match.

In our testing, CCoeff_normed provided the best results.

> Determine the max/min value of the template match result

Using minMaxLoc to find the max value.

Compare the max value with the boundary value (currently 0.123) - if so, trigger a positive match.

Number was determined just from trial and error as to what returned the best number of positives, while still returning 0 false positives (partially correct)

>Repeat template matching process with reversed roles (image 1 is template, image 2 is image)

Because we have a record of 0 false positives, we can assume that if either of the 2 template matches return true, there is a definite match.

We compare the first permutation(img 1 vs img2), if false, then we compare the reversed permutation (img 2 vs img 1) - if neither returns true, it is a non-match.

This step improved our overall accuracy from 99.2% to 99.8%, because there were some permutations where image 1 wouldn't template match with image 2, but 2 would match with 1.

Alternative ways experimented with:

Canny edge detection

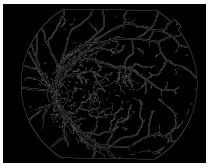
Tried using Canny for edge detection, but found that the edges determined were too noisy and hard to de-noise when preparing the image for thresholding. When testing against laplacian edge detection, we found that Laplacian returned an easily thresholdable matrix & was filled in upon thresholding, rather than the squiggly lines in Canny - we could easily threshold the laplacian's result by matching the threshold to the delta of the Laplacian function.

Blurring

Tried various orders of either gaussian blurs, median filter, before every step in the process. Checking before and after the amount of noise generated, to determine the best smoothing filter to use.

The location of blurring is important, e.g. by removing gaussian blur before Laplacian edge filter, our success rate dropped to 20.1% success.

Contrasting

Tested using the 'equalizeHist' function to match all the images' brightnesses, however found that it produced too much noise in the edge filters, even after various stages of blurring. 

We found we had the best results when taking the grayscale image + a little contrasting + blurring, and applying the Laplacian edge detection algorithm over it.
The barely contrasted grayscale image preserved the intensity of the edges well enough for the Laplacian edge detection to return an accurate result.
We found that just a little contrasting allowed the edges to be just a bit more intense for the edge detection, but for a minor reduction in performance having no contrasting would also be fine.

Thresholding
Used the adaptive thresholding on the image with an equalized histogram - however it had a significant amount of hard to remove noise even after both smoothing and median filters.
We experimented with various modes of thresholding, Otsu, Binary, Trunc, and To Zero, however the default Binary mode (combined with laplacian) returned the clearest result, with minimal extra steps/artifact removal required.

Comparisons
We originally thought a histogram match would be the best, however, we found that using a thresholded image (where the only values are black or white). Unless we did spatial pyramid histogram matching, it wouldn't work. Since we had taken the path of extracting a thresholded image of the veins, we instead looked to template matching as a more suitable match for our method.

Performance
We had no issue with the speed of the system when comparing against large datasets (tested around 500-1000 at a time), so found that we did not need to downsize and/or reduce the resolution of the images for this reason.

Accuracy
Overall
The accuracy of the whole system is around (+/- 1) 2/1000
(~99.8% success rate)
This statistic was determined using a randomly generated distribution of images, and did not compare repeat combinations, 1000 unique pairings from the ~5000 combinations were chosen.
-Originally we compared only 1 permutation of the images, i.e. image 1 against 2 for the template match, which yielded a 99.2% success rate (5/1000 comparisons) - once we implemented checking both permutations, i.e. image 2 against 1, the accuracy improved to a 99.8% success rate (2/1000).

Accuracy for Matching Values (Comparing matching retinas only)
When comparing against all images where a positive match should occur - 12/500 (97.6% success rate).
Failures are from predominantly images where from one to the other, the photo's retina has changed position a significant amount, such that a significant amount of new veins are visible and others are now hidden.
-Originally we compared only 1 permutation of the images, i.e. image 1 against 2 for the template match, which yielded an accuracy of 37/500 (92.6% success) - once we implemented checking both permutations, i.e. image 2 against 1, the accuracy shot up to 12/500 (97.6% success)

Accuracy Conclusion
The error is consistently ~2-3% max
The accuracy skew of the system is such that I have had 0 false positives, only false negatives in all of my testing, at a threshold of a required >12.3% match of the templates.