A deep learning image registration method to classify photos used in classifying Thyroid eye disease

Xilai Zhang
Electrical and Computer Engineering Department
University of California, Los Angeles
Los Angeles, US
xilaizhang@g.ucla.edu

Zecheng Fan
Department of Mathematics
University of California, Los Angeles
Los Angeles, US
fzc20070415@ucla.edu

I. Introduction

Thyroid eye disease, also known as Graves' disease, is a multi-system autoimmune condition associated with thyroid hormone abnormalities (usually hyperthyroidism), and a chronic inflammatory disorder affecting the eye surface, orbit and periocular tissues. [1] The ophthalmic component of this disease ranges from subclinical enlargement of the extraocular muscles to disfiguring and vision-threatening proptosis and optic neuropathy. [2]

To classify thyroid eye disease, various rubrics have been developed. In 1969, Werner proposed the 'NO SPECS' mnemonic, which was later used broadly as a description of the characteristics and severity of the disease. [3] This mnemonic is helpful in outlining the clinical symptoms and signs, but has been shown to be less useful in identifying the factors that describe the clinical activity. [4] In 1989, Mouritis and colleagues introduced the Clinical Activity Score (CAS), which, on a scale of 0 to 10, classifies the disease as inactive if the score is less than three and active if the score is three or more. [5] In 2006, Dolman and Rootman proposed the VISA classification with and outlined a gestalt for clinical evaluation. [6] This clinical evaluation is based on vision, optic nerve function, inflammation, strabismus and facial appearance, in descending order of importance. In 2008, The European Group on Graves' Orbitopathy (EUGOGO) first published a consensus statement. [7] A universal method of categorizing the activity and severity of thyroid eye disease is lacking in the literature. It would be beneficial to patients if there was a general classification method that can be applied widely. For this reason, we are motivated to develop a deep learning method that classify thyroid eye disease.

Machine learning has been widely applied to analysis of digital photographs and as such has become highly refined and robust. Given that thyroid eye disease is a condition which leads to striking and stereotypical facial and periocular features, it is logical to apply machine learning to thyroid eye disease.

The key clinical advantage of employing machine learning to classify thyroid eye disease is that such algorithms may detect thyroid eye disease in an early stage. An early intervention is crucial because the natural history of Thyroid eye disease follows "Rundle's curve." The entire time course is divided into active phase and stable phase. A higher severity of the disease at the active phase will lead to higher severity of the disease at the stable phase. And thus, initiating therapy at the active phase will greatly diminish the overall severity of the disease. Thus, a machine learning software that can detect Thyroid eye disease at its early stage could greatly help patients to reduce the overall severity of the disease.

To implement machine learning algorithms to classify Thyroid eye disease, we also need a comprehensive dataset. The dataset should include a large number of standardized external photos of the eyes and periocular region. It should also be as unbiased as possible, incorporating photos of eyes of people with and without Thyroid eye disease, of old people and young people, and of males and females, of people of all ethnicities, facial configurations and skin colors. However, as the number of photos we wish to analyze increases, the quality and angles of the photo taken become more diverse. And a common problem is that photos taken from different angles are fed to the same machine learning algorithm, and thus leads to errors. Thus we want to use image registration methods combined with deep learning to classify the angle and the quality of the photo taken: whether it is from the side, high or low angle, and whether it is high resolution or bad quality.

The classification of photos taken from different angles will help with better classifying Thyroid disease, because it makes different clinical representations much more obvious and thus easier for the algorithm to learn. One of the clinical representation of Thyroid eye disease, for example, is the bulging eyes. While the bulging of the eyes might be difficult to observe from level angle, it is obvious if the photo is taken from the low angle. On the other hand, if we are trying to observe the eyelid retraction, which is another common clinical representation of Thyroid eye disease, a photo taken from the level angle will reveal much more information than a photo of eyes taken from the low angle. Thus we want to use image registration methods to classify the different angles from which the photos of eyes are taken.

Besides classifying the angles of the photos taken, we also know that classifying the quality of the photos will help with the detection of Thyroid eye disease. If we can discard the low quality images, as in blurred or defocused images, we will be able to improve the range and standard distribution of the photos, and thus lead to better machine learning results.

Machine learning has proved previously successful in medical imaging and classifying, for instance in functional connectivity parcellation of the human brain [8] and kernel machine regression in neuroimaging genetics [9]. Thus we believe the use of deep learning image registration methods may be effective in classification of Thyroid eye disease. We aim to optimize image co-registration in order to maximize the chances that a classification algorithm will be effective in identifying the early stages of thyroid eye disease from clinical photos.

II. METHODS

Our methods consist of three major parts: find the facial landmarks of a face by employing functions in dlib library in python, use the facial landmarks to apply OpenCV functions to detect the face angle in the photo, and use OpenCV functions to obtain the affine matrix to achieve image registration. The code we wrote for the first two parts referenced the template provided by Adrian Rosebrock and Satya Mallick. [10][11]

Here is an overall high level understanding of the major steps in our methods: the code we wrote first extracts points on the face that can be considered as facial landmarks, such as nose tip, chin and eyes etc. These points are then stored into an array and passed to OpenCV for processing. The OpenCV module has its own default positions of the facial landmarks in the 3D space. Once the OpenCV sees the points passed to it by dlib module, it compares these points with the default positions of the face, and solves for the face angle in the photo. The face angle is then used as the criteria to filter out low quality photos and keep the good ones. After that we will apply the affine matrix transformation method taught by professor Scalzo in class. We will use the OpenCV function to help us retrieve the affine transform matrix. And then we apply this affine transform to each of the points to obtain a new image. This newly generated image is aligned, and we thus achieve the goal of image registration.

We also added other auxiliary functionalities to our code, such as detecting blurriness. In the remaining of this section, we will go into details of each of the modules of our code. Let us begin by first looking at the major modules.

The first major module we will examine is using the dlib library functions to extract the facial landmarks of the face in the photo. Here the primary functions we used are dlib.get_frontal_face_detector() and dlib.shape_predictor(). Before delving deep into what these two functions are and what they do, we first introduce what exactly dlib library is.

Dlib is a general purpose cross-platform software library. [12] It is most commonly used in C++, but we can also use it in python. Dlib contains a wide variety of tools in many areas of computer science, including image processing.

With this in mind, we can explore the functions in dlib that are especially useful in this project. The first function, get_frontal_face_detector(), returns the default face detector in dlib. The second function, dlib.shape_predictor(), takes in one parameter of dataset name and outputs a predictor trained by the specified dataset. The data package we used is the most commonly used one: iBUG 300-W, we also attached this data package together with our code. Dlib uses this data package to train a model to recognize 68 facial landmarks on the human face. Note that one can also use other dataset such as HELEN dataset to train the model. The idea is that we will use these function calls to retrieve a pre-trained model that recognizes the 68 facial land marks on the human face. The code will then annotate these facial landmarks on the photo.

Now that we have a predictor from dlib, we can proceed to explore the points we want. Note that we will only need a few of these facial land mark points. To understand why, we need to think about our assumption of predicting face angle.

What we are assuming here is that the face angle is predicted by the transformation of facial landmark points. Namely, we need two sets of facial landmark points, and infer the face angle by exploring the transformation between these two sets of points. We can already get the coordinates of the facial landmark points in the photo by applying dlib function calls. But this is only one set of the points. Where do we obtain the other set of points so that we have two sets of points to compare with each other?

The other set of points we need is based on pure estimation. We choose the locations for a few facial landmarks by where we think they would be when the face is not turned at all. The set of estimation points we used in this project are nose tip, chin, left eye corner, right eye corner and right mouth corner. And their assumed positions when the face is not turning are (0.0, 0.0, 0.0), (0.0, -330.0, -65.0), (-225.0, 170.0, -135.0), (225.0, 170.0, -135.0), (-150.0, -150.0, -125.0) and (150.0, -150.0, -125.0), in 3D coordinates respectively. Note that these numerical values are generated by our assumptions, so they could be a potential error or bias affecting the value of face angle detected.

We now have the set of points, namely, the locations of facial landmarks when the face is not turning. We will refer to this set of points as the "standard points" in the rest of this paper. And we will refer to the points annotated by dlib model as "marked points" in the rest of this paper.

Now we will turn our discussion back to obtaining the "marked points". We now know exactly which points we want: the points corresponding to nose tip, chin, left eye corner, right eye corner and right mouth corner. But there are 68 points in total annotated by our dlib model. So we have to find out which points in these 68 points correspond to the six facial landmarks we want. After a lot of print statements and guessing, we find the correspondences between the six facial land marks and their sequence numbers in the 68 points: nose tip is point 30, chin is point 8, left eye corner is point 36, right eye corner is point 45, left mouth corner is point 48 and right mouth corner is point 54.

We can thus iterate over the image to extract these points. Here, we also convert the default rectangle generated by dlib into a bounding box in OpenCV. After we extract the locations of these six facial landmark points, we put them into an array and pass it to the next module. This concludes all the steps of the first major module.

The next major module we will be looking at is the face angle detection module. This module is important because it determines whether a photo can be used in further machine learning study. If the face angle is too big, it means that the eyes are turned away too much from the camera, and will thus not be a good component of the dataset. This module acts as a filter on the photos.

As mentioned before, the face angle detection module will compare the "standard points" and the "marked points" to obtain the face angle, but besides that, this module also explores translation from world coordinates to camera coordinates and 3D-to-2D projection estimation. We will explain these extra steps in details.

First of all we need to figure out the relations between different coordinates. The facial land mark points we are interested in exist in the world coordinates. That is, the x,y and z values are defined with respect to the 3D space of the image world, and will not change if we move our heads around and view them from different angles. However, the camera coordinates are defined with respect to the viewer. The 3D coordinates of the facial landmark points in the camera coordinates are related to the position and focal length of viewing camera. Thus if we change our viewing port or the focal length of the viewing camera, the coordinates of facial landmark points in camera coordinates will change as well.

Now that we understand the difference between world coordinates and camera coordinates, we will examine the transformation we are interested in: the transformation between the "standard points" and "marked points". What we are trying to do is translating one 3D coordinate in the world coordinate to another 3D coordinate in the world coordinate, but at the same time we are always viewing the point from the camera coordinate, at any given point of the translation. Thus we can say what we are trying to find is the rotation and translation of the world coordinates with respect to the camera coordinate.

Up to this point sharp reader might have already realized one problem: all the translations we are talking about are 3D points, but the coordinates we obtained for the "marked points" are all 2D. So how are we going to find the translation between 2D "marked points" and 3D "standard points"? (Note here that our "standard points" are generated in 3D)

To solve this issue, we will need to use approximations. The function we used in our code is solvePnP in openCV. By default solvePnP uses a combination of DLT solution and Levenberg-Marquardt optimization. [13][14]

DLT stands for direct linear transform. And it helps us relating 2D coordinates to 3D coordinates by the following equation.

$$\begin{bmatrix} 2D-x \\ 2D-y \\ 1 \end{bmatrix} = scalar * \begin{bmatrix} focal-x & 0 & center-x \\ 0 & focal-y & center-y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3D-x \\ 3D-y \\ 3D-z \end{bmatrix} \quad (1)$$

Where focal-x/y is the focal length of the camera in the specified direction, center-x/y is the optical center of the camera in the x/y direction. 2D-x/y is the 2D coordinate and 3D-x/y/z is the 3D coordinate.

Another way of relating 2D coordinate to 3D coordinate we mentioned above is levenberg-Marquardt optimization. This method uses an algorithm to change the value of rotation vector and translation vector so that the re-projection error decreases in each iteration. Now a careful reader might be wondering what a re-projection error is and why we want to decrease it.

In the levenberg-Marquardt optimization, we project our 3D points onto the 2D image to obtain corresponding 2D coordinates. Our goal is that these projected 2D coordinates will match the 2D coordinates we set, in our case, the "marked points". However, at first we will have no guarantee that the projected 2D coordinate be at the exact locations of the "marked points". What we do know is that the more accurate our rotation vector and translation vector are, the closer our 2D projections will be to the "marked points". We thus measure the closeness by summing the squared distances between projected 3D points and the 2D "marked points". And this sum is called re-projection error. What levenberg-Marquardt optimization does is that it gives us a way to change rotation and translation vector such that the re-projection error is minimized.

Now that we have understood the two underlying estimation methods, we make our function call to solvePnP which returns the optimized rotation vector and translation vector. Note here that solvePnP also takes the camera_matrix as the input. camera_matrix holds the values of focal length and optical center of the camera, and the parameters are defined by ourselves. Again, since these parameters are defined by ourselves and based on approximation, they could be the potential cause of bias and error. We thus have tuned these parameters based on a lot of trials.

The rotation vector returned from solvePnP, however, is not in the format of the rotation matrix we learnt in class. To transform the rotation vector into the form we are familiar with, we use the cv2.Rodrigues() function.

Now that we have extracted the three by three rotation matrix that we are interested in, we can finally examine the angles of rotation along each axis. Here we will apply the arctan function, which is atan2 in python to retrieve the angles. Note here that we use math.degrees to convert the angle values from radians to degrees.

The retrieved angles of rotation are along x,y and z axis. We will later use these values to filter out low quality images. Let us now look at the third and final major module of this project: using OpenCV functions to obtain the 2D affine matrix to achieve image registration.

Up to this points, careful readers might be as confused as I used to be: Given that we already found out the rotation angles in 3D, what is the point of finding an extra 2D affine transformation matrix? Namely, why do we even need the third module?

This question is addressed by professor Scalzo during a scheduled meeting about this project. We know that the CNN architecture concludes different traits in each of its network layers. According to professor Scalzo, if the images we feed into the CNN architecture are not aligned in 2D, the CNN network will still have the ability to figure out that the photos are not aligned, and conclude traits based on the feature of the diseased eyes. However, this alignment will take up resources in the network layers that could be used to conclude other more important traits. This implies that if all of our images are aligned in 2D, the architecture will have higher efficiency. Furthermore,

if the photos are not aligned in 2D, they could cause bias and error if learnt by the machine. Thus, a pre-alignment of the photos in 2D will improve both the efficiency and accuracy of the architecture.

Thus we use OpenCV functions to help us achieve image registration. We will refer to this alignment process as image registration in the remaining of this paper, since image registration is defined as the process of transforming different sets of data into one coordinate system. [15]

To perform image registration, we first need to find the affine matrix. We know that a 2D three by three affine matrix can be calculated from translations of three 2D points. Quite similar to what we have done before to the facial landmark points, we will need two sets of points, the set of points before image registration takes place and the set of points where we want them to be after the image registration. And we only need three points in each set. Again, let us name the set of points before image registration the "tilted points", and the set of points after image registration the "aligned points".

We want the "aligned points" to be fixed 2D coordinates, so that all the photos can be aligned to the same coordinates. Again, we have to choose three points that we are especially interested in. In our project, we choose the nose tip, left eye left corner, and right eye right corner. We define the three coordinates of "aligned points" at (450.0, 450.0), (225.0, 620.0), and (675.0, 620.0), respectively.

It is kind of cliché but we would still like to state that the coordinates of "aligned points" are defined by ourselves and could thus lead to bias or error.

Now for each image, we can extract the coordinates of nose tip, left eye left corner, and right eye right corner to form the "tilted points", and try to align them to the "aligned points".

With the "tilted points" and "aligned points" set, we make a function call to the cv2.getAffineTransform. This function will return a three by three affine matrix that we are interested in.

After we obtain the three by three affine matrix, we call function cv2.warpAffine to apply the affine matrix to our original image. This will return a new image that is aligned to our "aligned points".

Up to this point, we have achieved image registration and finished the description of the three major modules in our project.

We will now discuss the auxiliary modules in our project. The filtering module based on face angle detection. Here, the user can specify the thresholds along x, y and z axis. In the iteration of all photos, if a photo has face angle bigger than the threshold, it is ignored. Only photos with face angle within the threshold will kept for further processing.

The crop module takes in parameters of the region of the image the viewer is interested in. The image is then converted into array form, and only the columns and rows corresponding to the region of interest are selected.

The control module make function calls to cv2.waitKey() and cv2.destroyAllWindows() etc. to allow control over the display of the image. The control module also takes as input parameters the input folder and output folder. It uses os.listdir() to open all the images in the specified directory that needed to be processed. When a file satisfies certain criteria, it makes function call to shutil.copyfile() to copy files into destination directory.

Even though the turning angles along three axis can be extracted, sometimes it could be difficult to visualize the turning angle of the output image by just looking at it. Thus there is also a visualization module in this project that draws a line from the nose tip of the face pointing outwards. This is achieved by using the function image_points.

We have thus finished description of all the modules in our project. We welcome readers to play with the code to gain a better understanding of each module.

III. RESULTS

The face angle detection part of the project gives professor Karlin the option to filter out low quality images he does not want. The remaining images after the filter are faces with eyes looking into the camera with small turned angles. This helps professor Karlin to better examine the condition of the eyes of the patients.

The image registration part has provided to the machine learning code of thyroid eye disease dataset of better quality. Since the previous machine learning code was running on unaligned images, there could be unseen bias underlying in the network architecture. The image registration of this project will eliminate the error caused by unaligned images.

We tested our project by running it over 10000 faces provided by professor Karlin. The faces include a wide range of diversity: faces of people with and without thyroid eye disease, faces of young, old, and middle age people, faces of male and females etc. Our project successfully filtered out faces with large turned angles and was able to apply image registration.

This project is also done by undergraduate students. We worked with professor Scalzo, professor Karlin and a group of graduate students. We also attended the meeting held in the Bel Air Room and talked to professor Eskin. Professor Karlin is satisfied with the work we have done, and professor Eskin offered Zecheng to continue this research in summer as paid research internship. We hope this project will ease the life of doctors in the future.

IV. DISCUSSION

We have mentioned throughout this paper that all of our self-defined parameters could be potential causes of bias and errors. But we need them to be defined to apply all the crucial function calls. Thus self-defined parameters are inevitable. But what we can do is to tune them so that they produce the best results.

After this project is done, we build two CNN modules. One detects if the face in the photo has glasses, and the other detects if the face is blurred. We did not go into details about them in the above description, but we have attached the code of these two modules in our submission, and introduced them in our powerpoint presentation. We highly encourage readers to look into them.

This project is written to help with the early detection and diagnosis of thyroid eye disease, but its functionality such as image registration and face angle detection and filtering can be applied to other medical issues as well. We hope this project helps to make the world a better place.

[1] FanYang, Hongwei Ma, and Xi-Qin Ding, A Thyroid Hormone Signaling in Retinal Development, Survival, and Disease, Vitamins and Hormones, vol 106, 2018, pp. 333-349.

[2] Henry B. Burch, and Rebecca S.Bahn, Endocrinology: Adult and Pediatric, 2016, pp. 1465-1477.

[3] Werner SC. Classification of the eye changes of Graves' disease. Am J Ophthalmol 1969; 68: 646–648.

[4] Weiler DL, “Thyroid eye disease: a review”, Clinical and Experimental Optometry, January 2017, pp.20-25.

[5] Mourits MP, Koornneef L, Wiersinga WM et al. Clinical criteria for the assessment of disease activity in Graves' ophthalmopathy: a novel approach. Br J Ophthalmol 1989; 73: 639–644.

[6] Dolman PJ, Rootman J. VISA Classification for Graves orbitopathy. Ophthal Plast Reconstruct Surg 2006; 22: 319–324.

[7] Bartalena L, Baldeschi L, Dickinson A et al. Consensus statement of the European Group on Graves' orbitopathy (EUGOGO) on management of GO. Eur J Endocrinol 2008; 158: 273–285.

[8] A. Schaefer, R. Kong and B.T.Thomas Yeo, Machine Learning and Medical Imaging, 2016, pp. 3-29.

[9] T. Ge, J.W. Smoller and M.R. Sabuncu, Machine Learning and Medical Imaging, 2016, pp. 31-68.

[10] [Online]Availablehttps://www.pyimagesearch.com/2018/04/02/faster-facial-landmark-detector-with-dlib/ [Accessed: 8- Jun- 2019].

[11] [Online]Available: https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/ [Accessed: 8- Jun- 2019].

[12] [Online]Available:https://en.wikipedia.org/wiki/Dlib://en.wikipedia.org/wiki/Dlib [Accessed: 3- Jun- 2019].

[13] [Online]Available:https://en.wikipedia.org/wiki/Direct_linear_transformation [Accessed: 3- Jun- 2019].

[14] [Online]https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm [Accessed: 3- Jun- 2019].

[15] [Online] https://en.wikipedia.org/wiki/Image_registration [Accessed: 3- Jun- 2019].