Pomona College
Department of Computer Science

Towards More Inclusive Facial Detection

Chinasa T. Okolo

April 29, 2018

Submitted as part of the senior project for the degree of
Bachelor of Arts in Computer Science

Professor Alexandra Papoutsaki, advisor

Abstract

WebGazer uses computer webcams to track eye-gaze locations of users in real time. This eye-tracking library is solely implemented in JavaScript and can be easily integrated within a web browser. WebGazer uses a model implemented by another JavaScript library, `clmtrackr` to fit facial models on faces tracked through video. This library allows WebGazer to train models that map eye features and their relative position on the screen.

Currently, WebGazer fails to accurately track people with beards, people wearing glasses, or people with dark skin. With the training image set for one of the face detection modules integrated within WebGazer being one of the most diverse available, it is crucial to understand why it still continues to fail. In order for WebGazer to be as accurate as possible, it is crucial for this face detection module, `clmtrackr` to be as inclusive as possible and work for a wide variety of faces.

Learning extensively how the external face detection tracker module `clmtrackr`, included by default in WebGazer, works is the best way to improve how facial features are tracked while the script runs and help determine external factors that inhibit WebGazer from achieving its full potential.

Acknowledgments

The author is grateful to Professor Alexandra Papoutsaki for her guidance and support throughout this project, as well as Alex Clemens and the Pomona College Human-Computer Interaction Lab.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Eye-tracking involves the use of a camera to measure the gaze of a subject and determine their gaze position on a screen. Eye-tracking has a multitude of applications in neuroscience, psychology, and human-computer interaction research. Data gathered from this technique enable researchers to understand how visual information is processed [PB06]. WebGazer is a JavaScript library that detects where a user is looking by mapping the appearance of the eyes to screen coordinates. Eyes are identified using external facial tracking modules to support its software framework and reduce the complexity of its code structure [PSL+16]. WebGazer's use of webcams to infer eye-gaze locations can provide useful data to help companies understand how to improve the interfaces they implement within their websites and computer applications.

The motivation behind this work came from the author's experiences with WebGazer and the overall bias that has persisted in applications of artificial intelligence. The focus of this research has been moved from eye-tracking to face detection due to this being the primary factor determining whether eyes can be tracked or not. The big question surrounding this project was whether the underlying face detection library `clmtrackr` fails because it cannot find individual facial landmarks or because it cannot detect a face at all in certain images.

The focus of the author's research has been on investigating whether the skin color was significant factor in the face detection failures experienced in WebGazer. If skin color does play a significant role, there must be a reason why. The author would like to additionally research what could be done to reduce these types of failures and if diverse datasets are a good step in this direction. The author's work in this project spans a skin color classification

and a quantitative description of state of the art datasets.

Along with subjects with darker skin, WebGazer fails to accurately track subjects who wear glasses and those who have significant amounts of facial hair. WebGazer's ability to discern faces is aided by `clmtrackr`, an external facial detection library implemented in JavaScript [PSL+16]. Because of the way this library is structured within WebGazer's code, any failure WebGazer experiences with face detection is due to `clmtrackr`. While the dataset used to train the `clmtrackr` algorithm contains images of people with these characteristics, it is unknown if the quality and quantity those images negatively affected `clmtrackr`. Improving `clmtrackr`'s ability to detect a diverse array of faces will enhance WebGazer's functionality and set the standard for inclusion and diversity for other eye-tracking libraries.

Given the use of `clmtrackr`, the module used as the underlying facial detection library in WebGazer, it is essential to understand how it works. It is also important to understand how it contributes to WebGazer's lack of functionality in face detection. If faces are not able to be properly tracked, then eye-gaze prediction cannot be accurate no matter how good the module claims to be. Methods used to fit faces such as the non-rigid/deformable face tracking/alignment using constrained local models [Mar12] do not work on the author of this paper. Due to this failure, it is hypothesized that WebGazer will not work for others who have features similar to hers. To begin deconstructing the `clmtrackr` module, a significant amount of time is needed to understand how the computer vision algorithms implemented within its source code track faces and why this causes `clmtrackr` to fail. Two diverse datasets, the Pilots Parliaments Benchmark and the Milborrow/University of Cape Town (MUCT) Landmarked Face Database will be used to test `clmtrackr` through work done by Alex Clemens. Once this is done, a color analysis will be completed to extract skin tone colors from the images in each dataset in hopes of finding significant results.

1.1 A novel approach to eye-tracking methods

WebGazer is currently the only eye-tracking library using webcams with the capability to be added on any web page. TurkerGaze, a tool for crowdsourcing saliency datasets, has provided a foundation for WebGazer. The system was designed using the relationships between gaze patterns and interaction cues incorporating user preferences and habits. TurkerGaze collects saliency data through large-scale methods such as Amazon Mechanical Turk [XEZ+15]. The importance of saliency in eye-tracking lies in its use-

ful in determining how on-screen images, shapes, or text stand out relative to its neighbors [SMS10]. The auto-calibration method created by Alnajar et al. estimates the gaze points (locations) of a user based on the gaze patterns of previous users [AGVG13]. Using observations of similar gaze patterns in humans, this method provides efficient gaze-tracking without the need for repeated calibration. Similar to WebGazer, PACE is an application that identifies and collects hours of user-interaction data to predict the gaze without the need of specialized tracking hardware [HKN$^+$16]. The methods presented in these papers demonstrate the power of open-source contributions in creating novel technology, but also display the downsides of inherently biased software.

1.2 Significance

The significance of this study lies in the ability to expose problems within WebGazer that focus on the issues of diversity and inclusion in computer science. Ensuring that software is inclusive of the population at-large and not just small subsets within the computing community is a democratic process that should be employed far more often. This paper reviews `clmtrackr`, the most popular JavaScript library able to detect faces using a web browser and computer webcam. An in-depth, investigation of `clmtrackr`'s capabilities will be performed and the results from an analysis of datasets run through `clmtrackr` will be discussed. With significant results, suggestions can be made to improve the accuracy and expand the functionality of WebGazer. Producing quantifiable modifications to `clmtrackr` would improve WebGazer and provide an opportunity for future scientists to integrate this tool into their research.

Chapter 2

Eye-tracking

Eye-tracking incorporates methods that measure eye movements and the sequence of changes in eye location on a screen [JK03]. The importance of understanding how people distribute their attention when viewing web pages cannot be understated [HWB12]. The benefits of knowing this information can aid in user interface design and how long pages and chunks of text are visually examined. Advances in eye-tracking were aided by the invention of devices that could collect information faster and more accurately. Significant innovation in eye-tracking led to non-intrusive eye-trackers that allowed researchers throughout the '50s and '60s to incorporate cameras into their work. This improved the accuracy of such devices and began the integration of cognitive factors into eye-tracking research [eye14].

Eye-tracking throughout the late 19th and mid 20th centuries was conducted with the main focus of improving educational techniques and medical treatments [JK03]. With the advent of the World Wide Web in the 90s, it became increasingly important to understand the differences between how users consume information on print and how they navigate similar or different information on screens. Data generated by a variety of eye-tracking studies has allowed researchers to describe general location-based characteristics of visual attention and how humans allocate their awareness towards certain objects. Using eye-tracking in human-computer interaction research has solved a multitude of problems and created new theories linking learning and memory to their respective cognitive processes. Concepts such as fixation impact [BCM09] that maps gaze data to visual scenes and gaze point-interaction which defines user interactions (clicks, cursor movements, etc.) in respect to current gaze locations have been useful in defining new metrics to provide valuable insights on eye movements. With these new methodologies, increased understanding of users' gaze location on webpages

will be useful in the creation of new digital interfaces.

2.1 Traditional eye-tracking methods

Traditional eye-tracking methods have relied on expensive and bulky hardware that tediously collects gaze data. Eye-trackers with built-in head and chin rests inhibit natural movement that is seen in regular computer use. While this may be necessary in psychology and neuroscience experiments, researchers in human-computer interaction cannot gain useful information from these methods [PB06]. Video-based eye-trackers encompassing web cameras have reduced the cost associated with traditional equipment while still meeting image resolution and frame speed requirements needed to collect usable data. These types of eye-tracking systems reduce the need for complicated calibration procedures and can be trained to self-calibrate, improving accuracy greatly. Current work in webcam eye-tracking either uses image saliency or gaze patterns to estimate user gaze [SMS10]. Computational models of visual saliency have been primarily used to analyze visual attention on images. The difference between saliency and gaze estimation is that visual saliency provides information about which particular regions attract more gaze rather than which regions people look at [SMS10].

Visual saliency models have not been able to expand as rapidly as gaze prediction models due to the small number of datasets in comparison to other vision recognition tasks. To create more datasets, faster methods are needed to collect gaze and saliency data. The less calibration eye-tracking software and programs need to run, the faster data can be collected. Sugano, Matsushita, and Sato proposed a calibration-free method by identifying gaze points from saliency maps, images generated to illustrate unique characteristics of its pixels [SMS10]. While eye-tracking has come a long way, future work is still needed to improve the computer vision algorithms used to track faces and pupils. Software that is not inclusive of skin tone, facial structure, eye shape, and other features cannot be expected to provide fair representation and may fail to help people most deserving of its benefits.

2.2 Webcam eye-tracking

2.2.1 TurkerGaze

TurkerGaze is a webcam based eye tracker deployed on Amazon Mechanical Turk that collects high-quality gaze data to predict saliency on images

[XEZ+15]. To improve the performance of current eye-tracking methods, the power of machine learning can be harnessed, but large datasets are needed to train machine learning algorithms. Traditional research experiments would involve the use of commercial eye-tracking equipment and participation of a plethora of subjects. However, this process is both expensive and time consuming. To get the large amount of data needed, the researchers involved in the TurkerGaze project built a browser-based tracking system to efficiently acquire responses from hundreds of participants. TurkerGaze was the first gaze-tracking system of its kind to both support crowdsourced eye tracking and use webcams while doing this on the browser. Users are guided through games on Amazon's crowdsourcing platform with a goal of being able to collect information for large-scale datasets. To account for poor lighting conditions, unfeasible browser conditions, and lack of memory that may be present, TurkerGaze required subjects to limit head movements and recalibrate frequently throughout their tasks. Compared to WebGazer, TurkerGaze requires extensive calibration and training of its sophisticated machine learning algorithms. The computational pipeline implemented within TurkerGaze uses the ridge regression model with the same input and parameters that WebGazer uses and becomes enhanced through the interactions of its users.

2.2.2 PACE

PACE, a Personalized, Auto-Calibrating Eye-tracking system developed by researchers at Hong Kong Polytechnic University is a desktop application that performs auto-calibrated eye tracking through user interactions [HKN+16]. Gaze information can provide valuable data that can be used in a variety of applications ranging from market research to education. To attain this information, gaze-estimation methods are needed that can accurately and efficiently track eye gaze in real time. PACE gathers data based on eye and facial information received from webcams. Based on this daily interaction data, PACE can identify good training samples resulting in better tracking of gaze patterns. PACE is auto-calibrated and uses a variety of user interactions to deviate from previous work that assumed users were always looking at the location they interacted at. The approach the researchers took to PACE involved applying "behavior-informed and data-driven approaches" [HKN+16], allowing to produce suitable training data. Eye-tracking software that can auto-calibrate can adapt to less than ideal lighting environments and permits data collection to operate much smoother. With this in mind, PACE uses implicit gaze-correspondence correlation shown in prior research

to aim for integration within real-world computing environments. Compared to WebGazer, PACE needs several hundred user interactions to provide gaze predictions. WebGazer can provide gaze predictions instantaneously after a single interaction takes place, but PACE incorporates intricate training algorithms that cannot do this taking it longer to reach the accuracy of WebGazer. Thus, WebGazer proves to be more suitable for continuous gaze prediction on any website.

2.2.3 ITU Gaze Tracker

As discussed previously, the cost and bulk of traditional eye-tracking equipment has limited access of these systems to people who cannot afford them. Open-source software such as the IT University (ITU) of Copenhagen's Gaze Tracker provides an accessible gaze-tracking system incorporating a webcam through a head-mounted setup [SASM+10]. This gaze tracker was evaluated through eye-typing tasks which allowed users, some with limited motor capabilities to successfully complete the task assigned. The design approach and considerations taken in the development of ITU's Gaze Tracker contrast sharply with the other applications listed above. The researchers involved in this gaze-tracking system emphasized the use of low-cost components, customization in where users could place different components (camera, display, lights, etc.) of the system, and robustness and accuracy of its software [SASM+10]. The ITU Gaze Tracker uses a three-component software pipeline that includes a gaze-tracking library, a camera class that processes images using the library, and the user interface that serves as an intermediary between the user and the gaze-tracking library.

2.2.4 WebGazer

WebGazer provides a novel approach to webcam-based eye-tracking by being the "first browser-based model to self-calibrate on real time via gaze-interaction relationships" [PSL+16]. WebGazer is an in-browser webcam eye tracking library that can be incorporated in any website. The eye-tracking model contained within WebGazer is self-calibrating and can infer eye-gaze locations in real time. Similar to the methods implemented in "Calibration-Free Gaze Estimation Using Human Gaze Patterns", WebGazer can create mappings between eye features and their respective locations on a screen. In their method, two components that detect pupils and estimate eye gaze are combined to form the basis of WebGazer's technology. To obtain more useful data, WebGazer combines user interactions such as clicks and cur-

sor movements with coordination delay. This forms a model that is based on the understanding of behavioral aspects of human-computer interaction [PSL+16]. Removing the need for explicit calibration phases allows WebGazer to be feasible in an assortment of environments and can provide marketers, advertisers, and researchers with more data that is fully comprehensive of a user's interactions. To evaluate the effectiveness of WebGazer's technology, an online study was conducted with 76 participants and demonstrated significant performance in two of its regression models.

WebGazer is completely implemented within JavaScript and can be integrated within any website with a few lines of JavaScript. Once this implementation occurs, eye tracking can be performed and data collection will begin. WebGazer relies on three facial feature detection libraries to supplement its lack of face and eye detection algorithms. The libraries, `js-objectdetect` enclose rectangles around the face and eyes after detecting them [Mar12] while `clmtrackr` fits models to the eyes and face [Mat14]. Current limitations of WebGazer inhibit functional face tracking for people with darker skin tones, those who wear glasses, and individuals with beards and other significant amounts of facial hair. With this in mind, it is plausible that WebGazer will be unable to track the faces of individuals who wear head or face coverings for religious reasons. These few problems may make WebGazer inaccessible to those who may benefit from it the most and highlight the common neglect of underrepresented groups in technological applications.

Chapter 3

Face Detection

Face detection is a computer technology used in a variety of applications to identify human faces in digital images. The types of algorithms implemented within face detection focus on the identification of frontal faces versus side profiles and other perspectives that obstruct a full view of the face [VJ04]. Face detection has multiple applications in eye-tracking, healthcare, and law enforcement and is already being used in many of these contexts.

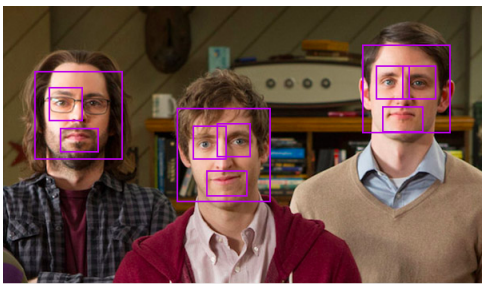

Figure 3.1: Example of face detection being used in `js-objectdetect` [Mar12].

To begin working with applications such as eye-tracking, face detection is needed to separate the face from its background, allowing an algorithm to focus on and extract specific features from the face [HL01]. Before the advent of the 20th century, face detection was one of the hardest problems in computer vision to solve. Algorithms introduced by Paul Viola and Michael Jones in their Viola-Jones object detection framework [VJ01] were based on solving this problem and provided the opportunity for face detection to occur in real-time.

To be the most successful, face-detection systems have to incorporate algorithms that can track faces in real time. The location of a face in natural settings where people may be navigating digital interfaces or engaging in physical activity is bound to vary and face tracking algorithms that assume the known initial face position are bound to fail [BP16]. These types of software are able to detect faces, but cannot account for variation in pose or lighting of faces, making them unrealistic for face tracking applications where video is used.

Skin tone has been proven to have an effect, often negative, on the ability of face detection systems to track faces. This has been shown by Buolamwini and Gebru in their work, Gender Shades and experienced by the author of this paper using `clmtrackr`'s algorithm in `js-objectdetect`. The appearance of skin tone depends on various contexts and is affected greatly by lighting conditions [HAMJ02]. With many commercial face-tracking systems, such as those from IBM, Microsoft and Face++ showing dismal accuracy rates for subjects of darker skin tones, it is obvious that there is a huge problem with the algorithms used to deploy these applications [BG18]. Research has been done to create systems that compensate for lighting conditions and skin-tone color has been shown to be successful in their respective research papers, but more work should be done to expand this research to video face-tracking and integrating this within commercial systems.

3.1 Computer vision face detection algorithms

3.1.1 Kanade-Lucas-Tomasi

The Kanade-Lucas-Tomasi algorithm (KLT) is a feature tracker that derives information from spatial intensity information to search for positions providing an optimal match [TK91]. In a sequence of images, specific points may be identified as feature points and as the angles within these images change, the feature points are displaced. Due to this displacement, their respective positions have to be mapped at every moment to ensure the correct features are being tracked. This ensures that as the viewpoints change, all of these feature points are being mapped into one coordinate system.

The KLT algorithm is an alternative approach to feature extraction, a method that describes a large set of data by reducing the amount of resources needed to do so. It is faster than traditional feature extraction methods because the way it uses spatial intensity information allows it to examine feature points of fewer images in hopes of finding the best fit [TK92].

This algorithm is essential to face detection applications where a face will

be shifting positions continuously. Applications such as video face tracking, surveillance, and activity recognition are examples of this.

3.1.2 Viola-Jones

Unlike the Kanade-Lucas-Tomasi algorithm that track faces based on trained features, the Viola-Jones algorithm uses Haar-like features to detect faces in real-time [VJ04]. Haar-like features use rectangular regions in different viewpoints of an image and calculates the difference of the sum of pixels inside the area of each rectangular region. The difference categorizes certain parts of an image and stores this information for further use.

Viola-Jones is the most widely used face detection algorithm and was the first object detection framework to work in real-time. Like most other face detection algorithms, a full frontal view of the face is needed to begin extracting feature points [JV03]. Viola-Jones is a robust algorithm in the sense that it has a high accuracy rate with a low chance of returning false positive answers. It works in real-time and detects faces as the first step in the process of facial recognition. To be clear, Viola-Jones is not a face recognition algorithm and focuses on the goal of distinguishing faces from other objects.

Viola-Jones starts off by selecting which Haar features it will use in its detection of a face. This provides the algorithm with facial characteristics that are consistent across the average human face [VJ01]. Next, an integral image is created to generate a sum of each rectangular Haar-like features. After this is completed, the most relevant features within a face are found by creating strong classifiers out of weaker ones. The last step groups the most relevant features in a method called "cascading", which determines whether a face is present based on the weight of a strong classifier and the features associated with it [VJ04]. This decreases computation time and improves how fast faces are detected within an image.

3.1.3 Shortcomings

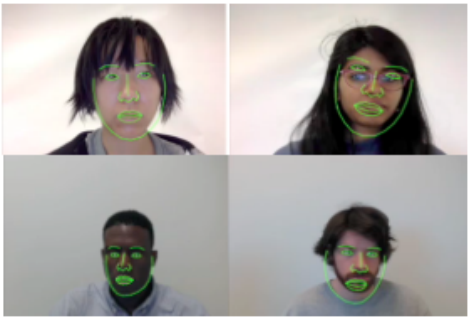

Figure 3.2: Faces not tracked with `clmtrackr`'s algorithm [Mat14].

Although face detection has come a long way, it often fails. Figure 3.2, shows some examples of faces that are not able to be accurately tracked. Many factors such as hair covering faces, glasses, dark skin, and facial hair have been shown to cause face detection to be less accurate.

Chapter 4

Facial Tracking Models

WebGazer is built in a modular form and uses a number of external facial tracking modules to support its software framework and reduce the complexity of its code structure. `clmtrackr`, `js-objectdetect`, and `tracking.js` are all of the modules implemented within WebGazer. These three modules are written in JavaScript and support WebGazer's software framework while reducing the complexity of its code structure.

4.1 clmtrackr

`clmtrackr` is a JavaScript library that fits facial models to faces in videos or images [Mat14]. This method can be used in applications such as emotion detection and individual detection where precise facial positions are needed. It is the most accurate library incorporated within WebGazer, as it is the one that aids most in detecting facial outlines and eye contours.

`clmtrackr` is the most popular JavaScript library that is able to detect a face using a web browser and a laptop camera. The algorithm interacts with key features, which consist of smaller image subwindows that might contain facial features instead of the pixels themselves. `clmtrackr` represents a face using an array of manually placed facial features and their x,y positions. An example of an array of facial features is seen in figure 4.1 on page 16. `clmtrackr`'s algorithm uses these x, y positions to create 70 different classifiers representing specific features on a face. The classifiers search for regions around these points, in pursuit of the best fit on which it converges.

`clmtrackr` uses a procrustes superimposition of two shapes by mapping the new shape onto the old, minimizing the difference between the shapes

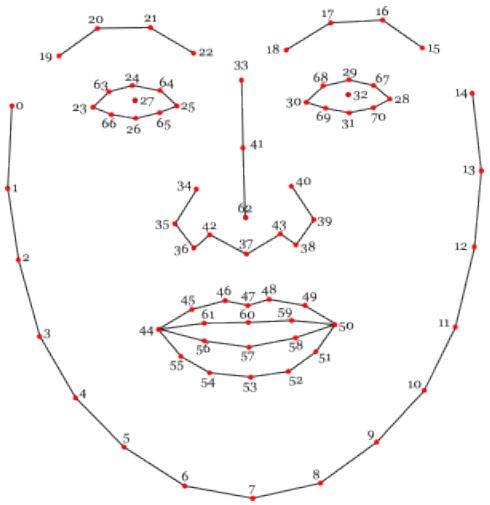

Figure 4.1: Mean face of all faces in the MUCT dataset.

[SLC11]. When using a procrustes analysis to map the face, the algorithm takes into account dimensions like head rotation, tilt, scale, and different expressions.

The existing code within the `clmtrackr` algorithm detects where key features such as the eyes, nose, and mouth are located in the image. For each feature found, small image windows are created around them [Mat14]. The algorithm then groups these smaller image windows into a larger rectangle and assigns the rectangle a confidence score. The rectangle with the highest confidence score represents the best fit box for the face, and superimposes a facial model onto it. Figure 4.2 on page 17 shows a facial model superimposed onto a subject's face.

Figure 4.2: Example of a face detected by the clmtrackr algorithm.

4.2 Js-objectdetect and tracking.js

`js-objectdetect` is another JavaScript library that uses classifiers to identify the front of a face and both eyes. After doing this, it uses `tracking.js` to request a users webcam and detect their face [Mar12]. They both work together to detect the face and eyes within a video stream, allowing WebGazer to focus on capturing the location of the pupil and begin eye-tracking. Figure 3.1 on page 11 is an example of bounding boxes being drawn around multiple sets of eyes, noses, and faces in an image.

Chapter 5

Datasets

5.1 MUCT

The MUCT dataset is one of the most diverse datasets used to train current face detection algorithms, most notably `clmtrackr`. The MUCT dataset consists of 3755 images of 276 individuals [MMN10]. It was created at the University of Cape Town in South Africa and provides more diversity of skin tone, lighting, gender, and facial expressions than currently available landmarked 2D face databases. The images within this dataset can be seen in figure 5.1 on page 20.

The images within the MUCT dataset are annotated with 76 manual landmarks situated around various features of the face such as eyes, nose, and mouth (See figure 6.2 on page 25). These landmarks were used to crop faces from the images in this dataset and was helpful in the method used to process skin tone colors from the images.

Figure 5.1: Examples of images within the MUCT dataset.

5.2 Pilots Parliaments Benchmark

In early 2018, Buolamwini and Gebru released the Pilots Parliaments Benchmark, a dataset created to achieve better intersectional representation between gender and skin color [BG18].

This dataset was created from two other datasets, IJB-A and Adience. The IJB-A dataset was released by the Intelligence Advanced Research Projects Activity in conjunction with the National Institute of Standards and Technology. It consists of 500 individuals who are public figures in their respective countries. At the time of its release, this dataset was the most diverse set of faces from a wide range of geographical areas [KKT+15]. Adience is a benchmark released in 2014 and is used to classify subjects by age and gender [LH15]. It consists of 2,284 subjects, but is not very diverse with 86.2 percent of the subjects classified in the first three Fitzpatrick [Fit88] skin types.

The Pilots Parliaments Benchmark dataset solves most of the problems related to gender and skin-tone diversity within the IJB-A and Adience datasets. PPB consists of 1270 photos, each representing one individual from three African countries (Rwanda, Senegal, and South Africa) and three European countries (Iceland, Finland, and Sweden). These countries were chosen based on their high levels of gender parity in their respective national parliaments.

The PPB dataset was used to analyze the accuracy of three commercial

gender classifiers from IBM, Microsoft, and Face++. These classification systems showed the highest failure rates on female subjects with darker skin tones (those in the last three Fitzpatrick skin types) [BG18].

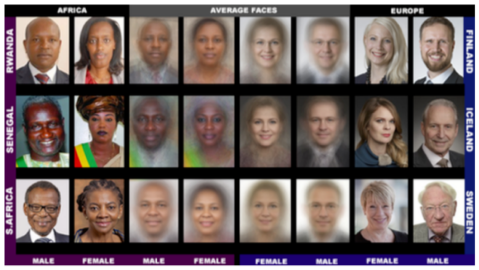

Figure 5.2: Images from the Pilots Parliaments Benchmark dataset.

Chapter 6

Methods

The goal of determining why `clmtrackr` fails to track certain faces started with analyzing the dataset used to train it. The color analysis described in the methods of this paper began with the MUCT dataset.

Several Python scripts were coded for different tasks in this analysis. The first Python script was created to crop the images and extract the foreheads in each photo in the MUCT and PPB datasets while another script was created to sort the colors obtained from the images. Once each script was tested and correctness was guaranteed, both of these scripts were combined into one. From the combined script, a copy was created for the MUCT dataset and another copy the PPB dataset. Since the datasets were contained in different directories on the author's computer, the separate scripts helped to keep the enormous amounts of images organized. This increased the efficiency of the computational pipeline created to analyze the MUCT and PPB datasets and helped streamline the process of extracting and analyzing skin tone colors from the images in each respective dataset.

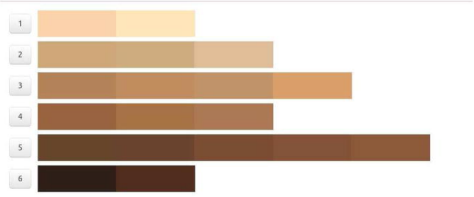

Figure 6.1: Color bar tool created by Ho and Robinson [HR15].

The goal of writing these scripts was to sort the images in the MUCT and PPB datasets into groups using the color bar scale defined by Ho and Robinson [HR15]. In this paper, six different categories of skin tone colors

are assigned ranging from light to dark (based on the Fitzpatrick skin types [Fit88]). Unfortunately, this scale is limited to the range of skin tones obtained from the subjects in Ho and Robinson's experiment. This scale can be seen in figure 6.1 on page 23). By sorting the images from each dataset into categories, the aim is to identify anomalies in skin tone that can help determine why the computer vision algorithms integrated within `clmtrackr` often fails to track darker faces. Coupled with the fact that the MUCT dataset was the one used to train `clmtrackr`, finding out why WebGazer does not work for all faces in the dataset is even more important.

6.1 Automated Face Detection

6.1.1 First Cropping Attempt

When trying to sort the images in the dataset, it was thought that finding the dominant color within each image would be best. The problem with this was that there is a large blue background in every picture of the subjects in the MUCT dataset (See figure 6.3 on page 26). This skewed the extracted colors to being either blue, gray, or a bluish-purple. To fix this, the next step was to crop the faces to certain regions defined by the landmarks that were manually selected by the authors of the MUCT dataset. These landmarks range from (x_0, y_0) to (x_{75}, y_{75}) and target specific features around the face such as the chin, eyebrows, eye shape, mouth, and nose. These landmarks were input into an Excel csv file and were imported into the Python environment for use. Since not all 76 coordinates were of importance, the points (x_0, y_0) and (x_7, y_7) in Python's `crop` function were used to crop a square of the face that included the sides of the face down to the chin. The first coordinate pair specifies the top left-hand corner of the square and the second coordinate pair specifies the bottom right-hand corner of the square that will be used to crop an area of the photo. The results of this cropping attempt can be seen in figure 6.4 on page 26.

Figure 6.2: Example of the labeled landmarks in an image from the MUCT dataset.

After cropping the images using these coordinates, a Python function that finds the most dominant color within each image and creates a color palette of the most dominant colors within the respective image [Gil14]. After running this function on the cropped images, the top three dominant colors were retrieved from each image. Unfortunately, the function still picked up the gray-blue backgrounds even though they were mostly cropped out of the images. To automatically crop the faces within each image, a Python library called `autocrop` [Fra18] was used. Unfortunately, the same problem with the background color persisted (See figure 6.5 on page 27). Due to this, a new cropping method had to be devised. It was noticed in all of the images that the most consistent and realistic skin-tone color was found on the forehead. From this, a new method was devised where the color from the pixel in the middle of a subject's forehead was extracted.

Figure 6.3: Example of an image (uncropped) in the MUCT dataset.

Figure 6.4: First attempt at cropping images in the MUCT dataset.

6.1.2 Second Cropping Attempt

This realization helped develop a new method of obtaining skin-tone colors. To begin the process of obtaining more realistic colors from the MUCT dataset, the foreheads in each image were cropped. With this in mind, the original coordinate points used in the first cropping attempt had to be adjusted. The coordinate pairs (x_{16}, y_{16}) and (x_{22}, y_{22}) were used as a starting point for cropping. When testing these points, the wrong section of the faces were cropped. This meant that the original coordinate points for landmark 22 were too low to create the square needed for cropping foreheads. To adjust for the forehead height, 50 pixels were subtracted from the y-coordinate of (x_{22}, y_{22}). This new method was successful and accurate in creating an ideal representation of the forehead. After obtaining the forehead of each subject in the dataset, the RGB color of the middle pixel in each forehead image was extracted. All of the RGB pixels for each image were added to a list and the list was converted to represent the colors of these pixels in hexadecimal format.

Figure 6.5: An image cropped with the `autocrop` function.

Figure 6.6: Example of a forehead cropped using the second cropping technique.

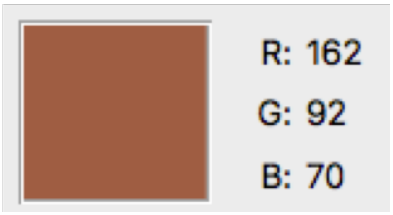

Figure 6.7: Color extracted from the first cropping attempt.

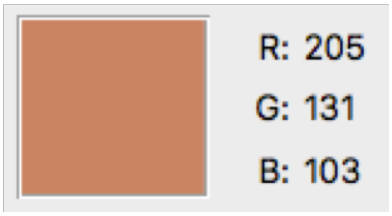

Figure 6.8: Color extracted from the second cropping attempt.

6.1.3 Sorting Extracted Image Colors

Converting the RGB pixel list to hexadecimal format helped the process of sorting these colors. It is easier to sort colors using their hexadecimal representation versus their RGB format. While RGB tuples can be sorted numerically, this does not sort them in a rainbow-like order that is visually pleasing. Hexadecimal format provides the best representation of RGB colors and converting to HSV provides the best format for actually sorting the images based on intensity. HSV accurately captures the color of an object by quantifying its hue, saturation, and brightness value. To sort the colors obtained from the foreheads, the hex pixel list of each dataset was converted to its respective HSV number and the new list was sorted using Pythons inbuilt `sort` function. To visualize this newly sorted list, a color palette was created using Seaborn, a Python visualization library based on Python plotting libraries [Was12].

After plotting the list of colors extracted from each image, the differences in both the unsorted and sorted color lists were observed. Python's built-in `sort` function was used to initially sort the colors, but it was not perfect. The list output by `sort` demonstrates the sorted list with a few outliers (See figure 6.9 on page 29). To get a more accurate representation of the distribution of skin tone colors within the datasets, these outliers were fixed

by manually moving the colors to their best assumed position.

6.1.4 Organizing Images and Classifying Colors

After manually adjusting the colors in the sorted list, the images corresponding to these colors have to be known. Fortunately, the image names correspond with the indices of the original unsorted list, but there needs to be a correlation when converting from the unsorted list to the sorted list. Next, a mapping between the image name and its respective color was created to keep track of what hexadecimal and RGB values correspond to each image in the dataset.

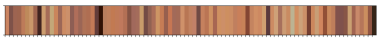

Figure 6.9: Unsorted color list from the images in the MUCT dataset.

The colors obtained from each image in the MUCT and PPB datasets were sorted into six categories based on the six Fitzpatrick skin types. This was done through a uniform classification where the dataset was split into six even buckets. (See figure 6.10).

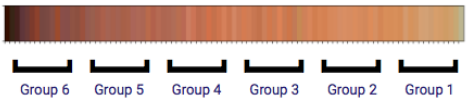

Figure 6.10: Sorted color list from the images in the MUCT dataset.

Chapter 7

Experiment

7.1 Comparison with Gender Shades

After starting the preliminary work for this project, the students in the Human-Computer Interaction (HCI) Lab at Pomona College found similar work being done at the MIT Media Lab [BG18] by Buolamwini and Gebru. The project, Gender Shades, focuses on testing the ability of software from Face++, IBM, and Microsoft to guess the gender of a face. All of these gender classification products use artificial intelligence and claim high levels of accuracy.

The authors curated a set of 1270 images as benchmark for their experiment. The original dataset (IJB-A) includes subjects from multiple countries around the world, but the Pilots Parliaments Benchmark focuses on subjects from parliaments in 3 African countries and 3 European countries. The images in this dataset were then categorized by gender, skin type, and the intersection of these two. Based on the binary gender classification labels output by all three classification software, Buolamwini and Gebru's analysis used two labels for male and female subjects.

While the premise of their work focused on skin color, its analysis showed alarming disparities between images of people with lighter and darker skin. Out of all of the faces misgendered by Microsoft's software, almost 94 percent of these faces were of darker skin tones [BG18]. Out of the faces misgendered by Face++'s software, almost 96 percent were images containing female subjects. By taking the intersection of these results, it was found that women with darker skin were disproportionally misgendered, having a failure rate of 33 percent [BG18]. With this, the authors (Buolamwini and Gebru) reached the conclusion that the software from IBM, Microsoft, and Face++ had the lowest accuracy on darker females.

Buolamwini and Gebru have been extremely helpful in allowing the students in the HCI Lab at Pomona College to access their curated dataset (PPB). The skin color distribution of the images within the Pilots Parliaments Benchmark dataset provided us with additional data to compare their results with the results the author of this paper received from testing the MUCT dataset.

7.2 Fusion with Complementary Work

The work for the author's senior project was completed in conjunction with Alex Clemens, another student working on his senior project in the HCI Lab at Pomona. His part of the research project focused on actually determining if the faces in the MUCT and PPB datasets were able to be found by the `clmtrackr` algorithm.

Clemens' work included analyzing the CLM algorithm, handwriting debugging tools that included a bounding box around any face that the CLM algorithm could detect, and writing a Python script to gather all the picture names to iterate through the folders containing images from the MUCT and PPB datasets. Clemens wrote a JavaScript function to automate the process of loading an image, adding the bounding box around the face in each image, downloading the image, manually placing images into folders, and adding manual annotations to see which faces were detected.

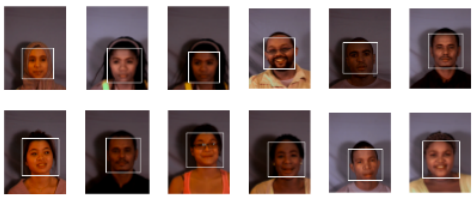

Figure 7.1: Faces extracted from images within the MUCT dataset.

His JavaScript tool implemented a model that would converge a box around faces in an image if the face was found. From this, he created four categories for each dataset: "Converged", "Converged But Not Well" (CBNW), "Lost", and "Not Found". "Converged" represents the images in each dataset where his model was able to find a face. "Converged But Not Well' represents the images where the `clmtrackr` algorithm was able to find a face, but the best fit box was slightly offset from the face. Lost means that the model was initially able to guess where the face was and draw a

bounding box around the face, but was not able to match the exact location of the face in the image and draw a green outline. “Not Found” means that the face was not detected from running the `clmtrackr` algorithm.

Chapter 8

Results

Automation Results				
Dataset Name	Converged	CBNW	Lost	Not Found
MUCT	391	51	55	1
PPB	323	39	116	24

Table 8.1: Results from automation

The automation of Clemens' work was run on 500 images from each dataset. The MUCT Dataset had 394 images in the "Converged" folder, 51 images in the "Converged But Not Well" folder, 55 images in the "Lost" folder, and 1 image in the "Not Found" folder. The PPB dataset had 323 images in the "Converged" folder, 39 images in the "Converged But Not Well" folder, 116 images in the "Lost" folder, and 24 images in the "Not Found" folder.

The research in the Gender Shades paper displayed significant results in regards to women with darker skin tones (those in Fitzpatrick groups 4-6) having the least accuracy when tracked by commercial face detectors [BG18]. With this in mind, the author wanted to find out if `clmtrackr` would produce similar results. From the folders output by Clemens' code, the "Not Found" folders from each dataset were analyzed. To determine if there were any significant anomalies in what faces were detected, the same skin color analysis initially completed on the MUCT was ran on the PPB dataset. No additional analysis was completed on the MUCT dataset because there was only one image in the "Not Found" category folder. This is not surprising because the MUCT dataset was used to train `clmtrackr`. This outlier was most likely due to the overexposure and awkward head pose in the image (See figure 8.1 on page 36).

Figure 8.1: Image of a face in the MUCT dataset categorized to the Not Found folder.

To determine if similar results could be found in the MUCT dataset, a similar color analysis was completed on the Pilots Parliaments Benchmark dataset. Unlike the MUCT dataset, the PPB dataset does not contain any landmark annotation, which would make it difficult to find the coordinates of the forehead to crop. Instead of extracting the foreheads from each image in the PPB dataset, a face in each image was obtained using autocrop [Fra18]. Since this library was used before, the author knew that it was pretty accurate in finding faces and cropping them as close as possible.

To begin the second round of skin-tone color analysis, the color from the middlemost pixel on the face was retrieved. The color pixels were extracted in RGB format, converted into hexadecimal format, and assembled into a list of these colors and their corresponding image name. The hexadecimal colors were sorted from darkest to lightest using Python's `sort` function. Again, the outliers were organized to align as closely as possible with the sorted color palette and divided to create a uniform classification of six categories. From this, the categories the untracked images were in could be determined.

Of the 24 images in the "Not Found folder", 15 were male and 9 were female. 20 out of 24 images in this folder were classified in the darker

Fitzpatrick skin tones (Groups 4-6) with 6 images being in Group 5 and 14 being in Group 6. Surprisingly, 4 out of 24 images in the "Not Found" folder were classified in Group 1.

The lack of consistency in pose, lighting, and image size within the PPB dataset may have contributed to the disparity between the results of the "Not Found" folders from both datasets. Although the sample size is small, a clear trend is noticed in images with subjects of darker skin-tone colors less likely to be successfully recognized.

Chapter 9

Discussion

WebGazer, an open-source library provides a novel approach to auto-calibrated eye-tracking through webcams. `clmtrackr`, the face detection library integrated within WebGazer captures faces, but fails when it comes to images of subjects with darker skin tones. This paper explored the history of eye-tracking, face detection techniques, webcam-based eye-trackers, facial tracking modules, and used a skin-tone color analysis to investigate why certain face detection algorithms fail. Despite its performance, `clmtrackr` needs to improve its ability to track users with darker skin tones, those who wear glasses, those with beards, and users who operate in low light conditions. Understanding the tracking modules implemented within WebGazer and finding ways to improve how subjects with darker skin tones are tracked will allow WebGazer to become more inclusive and fully democratize its use.

9.1 Limitations

The MUCT dataset, while consistent, contained images with much lower quality than the PPB dataset. For many of the photos, there seemed to be gray-purple or gray-blue filter cast upon them (See figure 6.3 on page 26). This contributed negatively to the accuracy and realism of the skin tone colors extracted from each subject photographed in this dataset. Despite this, all of the photos were categorized into Fitzpatrick groups [Fit88] since a uniform classification was used.

The quality of images within the PPB dataset were not consistent and hindered the ability of the code to recognize faces within these respective pictures. All of the images from the European countries in the dataset

(Finland, Iceland, and Sweden) were taken in settings where the lighting was consistent and the head pose was similar to standard headshot positions (See figure 5.2 on page 21). Some photographs from the African countries in the dataset had these characteristics, but some were taken during the subject's speeches or in a position too far from the face. This produced outliers within our data and made it hard to eliminate what factors may have contributed to problems in face detection.

Chapter 10

Future Work

Looking back on the work done this semester, weve realized a few things:

Face detection is a very large problem with many different dimensions - it is unlikely that the author would have been able to make it perfect within just one semester.

The result of this experimentation produced some non-conclusive results, with some outliers showing up in the outputs. We do believe that skin color is a significant factor in face detection, but still do not have very definitive results. Having more time would give us the ability to test `clmtrackr` with more data and/or with different hypotheses.

Regardless, the goal of this project was to draw attention to and improve problems that might contribute to face detection failing - and this goal was accomplished. This research project was a small, but positive step in the path to identifying what problems contribute to poor face detection.

Chapter 11

Conclusion

With this research, the author hopes to improve the way computer vision algorithms are implemented to track faces. The author was inspired to conduct this research due to personal experiences of WebGazer not working on her face. Face detection has the potential to improve the quality of life for people around the world, but if it is not able to track faces of people with varying skin tones and features, this potential becomes limited. The significance of this project lies in the ability to expose problems within WebGazer that relate to the current issues of diversity and inclusion in computer science. It is known that machine learning and artificial intelligence algorithms have been affected by inherent bias due to a lack of diverse datasets and outright bias from engineers encoding these algorithms. Ensuring that software is inclusive of the population at-large and not just small subsets within the computing community is a process that should be employed far more often. Producing quantifiable modifications in `clmtrackr` would improve WebGazer and provide an opportunity for scientists to integrate the author's findings into their future research.

Bibliography

[AGVG13] Fares Alnajar, Theo Gevers, Roberto Valenti, and Sennay Ghebreab. Calibration-free gaze estimation using human gaze patterns. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 137–144, 2013.

[BCM09] Georg Buscher, Edward Cutrell, and Meredith Ringel Morris. What do you see when you're surfing?: using eye tracking to predict salient regions of web pages. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 21–30. ACM, 2009.

[BG18] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91, 2018.

[BP16] Ritesh Boda and M Jasmine Pemeena Priyadarsini. Face detection and tracking using klt and viola jones. *ARPN Journal of Engineering and Applied Sciences*, 2016.

[eye14] Eye tracking through history. `http://eyesee-research.com/blog/eye-tracking-history/`, 2014. Online; accessed 11 November 2017.

[Fit88] Thomas B Fitzpatrick. The validity and practicality of sun-reactive skin types i through vi. *Archives of dermatology*, 124(6):869–871, 1988.

[Fra18] Franois Leblanc. autocrop. `https://github.com/leblancfg/autocrop`, 2018. Online; accessed 29 February 2018.

[Gil14] Ze'ev Gilovitz. Finding dominant image colours using python. `https://github.com/ZeevG/`

`python-dominant-image-colour`, 2014. Online; accessed 2 February 2018.

[HAMJ02] Rein-Lien Hsu, Mohamed Abdel-Mottaleb, and Anil K Jain. Face detection in color images. *IEEE transactions on pattern analysis and machine intelligence*, 24(5):696–706, 2002.

[HKN+16] Michael Xuelin Huang, Tiffany CK Kwok, Grace Ngai, Stephen CF Chan, and Hong Va Leong. Building a personalized, auto-calibrating eye tracker from user interactions. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5169–5179. ACM, 2016.

[HL01] Erik Hjelmås and Boon Kee Low. Face detection: A survey. *Computer vision and image understanding*, 83(3):236–274, 2001.

[HR15] Byron K Ho and June K Robinson. Color bar tool for skin type self-identification: a cross sectional study. *Journal of the American Academy of Dermatology*, 73(2):312, 2015.

[HWB12] Jeff Huang, Ryen White, and Georg Buscher. User see, user point: gaze and cursor alignment in web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1341–1350. ACM, 2012.

[JK03] RJ Jacob and Keith S Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2(3):4, 2003.

[JV03] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3(14):2, 2003.

[KKT+15] Brendan F Klare, Ben Klein, Emma Taborsky, Austin Blanton, Jordan Cheney, Kristen Allen, Patrick Grother, Alan Mah, and Anil K Jain. Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1931–1939, 2015.

[LH15] Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 34–42, 2015.

[Mar12] Martin Tschirsich. Js-objectdetect: Computer vision in your browser - javascript real-time object detection. `https://github.com/mtschirs/js-objectdetect/`, 2012. Online; accessed 29 October 2017.

[Mat14] Audun Mathias. clmtrackr: Javascript library for precise tracking of facial features via Constrained Local Models. `https://github.com/auduno/clmtrackr`, 2014. Online; accessed 29 October 2017.

[MMN10] S. Milborrow, J. Morkel, and F. Nicolls. The muct landmarked face database. *Pattern Recognition Association of South Africa*, 2010. `http://www.milbo.org/muct`.

[PB06] Alex Poole and Linden J Ball. Eye tracking in hci and usability research. *Encyclopedia of human computer interaction*, 1:211–219, 2006.

[PSL+16] Alexandra Papoutsaki, Patsorn Sangkloy, James Laskey, Nediyana Daskalova, Jeff Huang, and James Hays. Webgazer: Scalable webcam eye tracking using user interactions. In *IJCAI*, pages 3839–3845, 2016.

[SASM+10] Javier San Agustin, Henrik Skovsgaard, Emilie Mollenbach, Maria Barret, Martin Tall, Dan Witzner Hansen, and John Paulin Hansen. Evaluation of a low-cost open-source gaze tracker. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, pages 77–80. ACM, 2010.

[SLC11] Jason M Saragih, Simon Lucey, and Jeffrey F Cohn. Deformable model fitting by regularized landmark mean-shift. *International Journal of Computer Vision*, 91(2):200–215, 2011.

[SMS10] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Calibration-free gaze sensing using saliency maps. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2667–2674. IEEE, 2010.

[TK91] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.

[TK92] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[VJ04] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[Was12] Michael Waskom. Seaborn: statistical data visualization. `https://seaborn.pydata.org/tutorial/color_palettes.html#palette-tutorial`, 2012. Online; accessed 13 March 2018.

[XEZ+15] Pingmei Xu, Krista Ehinger, Yinda Zhang, Adam Finkelstein, Sanjeev Kulkarni, and Jianxiong Xiao. Turkergaze: Crowdsourcing saliency with webcam based eye tracking. *arXiv preprint arXiv:1504.06755*, 2015.