Eyetracking and Facial Expression-Driven Human Computer Interaction

May Kalnik Hari Kumaran Kevin Kwan Hengyuan Zhang
Georgia Institute of Technology
Atlanta, GA, USA
mkalnik3@gatech.edu hkumaran3@gatech.edu kkwan9@gatech.edu hzhang902@gatech.edu

1. Abstract

The development of the field of deep learning vastly improved human computer interaction. While effective, we feel some solutions still vastly underrepresented populations with limited mobility, and are also impacted by an inherent bias inherent. Furthermore, traditional computer input devices often prove to be inaccessible to such individuals. We introduce an implementation that integrates several strengths from these prior solutions, namely a previous implementation of the MPIIGaze Paper. To be more specific, our solution consists of a sequence of taking in input image data (via a webcam), image eye segmentation, gaze angle prediction, 2d point estimation, and screen coordinate regression. This approach allows us to effectively control our mouse with our eyes to a degree of consistency. Comparative efforts with other approaches, namely our OpenCV baseline implementation, older CNN architectures trained on the MPIIGaze Dataset, showed that our proposed version of the MPIIGaze model architecture modification and fine-tuning yielded a viable tool of moving the mouse on a computer using eye gaze direction with lower tracking time and jitteriness.

2. Introduction

2.1. The Problem

In the realm of digital interaction, there exists a new problem of upper limb pain due to constant computer use, despite attempts to make ergonomic adjustments [3]. Furthermore, significant challenges persist for individuals with limited mobility due to conditions such as arthritis, paralysis, or amputations. Traditional computer input devices, such as a mouse, often prove to be inaccessible for these individuals. As a possible solution, developers have worked on using a camera to capture a user's facial expressions as input; different facial expressions correspond to different actions. While this solution has been previously explored, their narrowness in scope and recency is a cause for concern, along with bias that hinders their performance for a wide range of individuals. Furthermore, these implementations can often overlook individual differences such as eye size, eye color, age, or the presence of glasses, which leads to inaccurate tracking and control, limiting potential user accessibility [9] [5]. Our overarching goal is to build upon these models, namely, fusing multiple strong suits of these solutions, while experimenting and comparing with various architectural and hyperparameter choices to create a complete end-end model that serves our purpose specifically. If successful, we would enable individuals to control their computers without the need for hand movements or external mice. As we mentioned previously, our team sees this as a pivotal step in combating digital ableism and bias, as well as potentially opening the door for even more creative means of human computer interaction.

2.2. Related Work

Several studies have laid the groundwork for our project which gave us key insights into the successes and pitfalls of other models that we can utilize when creating our own.

In a paper by Pathirana et al [7], we were able to uncover the general approaches to gaze detection. Essentially, there are two general approaches to gaze detection: model-based and appearance based. Model-based approaches are more precise but require calibration, extracting features to construct a geometric model of the eye. Appearance-based approaches directly map the image of an eye to the gaze. Learning about these approaches helped us to brainstorm our different methods, and ultimately led us to an appearance-based approach.

This paper also gives some insight on our chosen dataset. The dataset should contain robust variations in the environment, different illumination intensities and directions, body and face poses, and different backgrounds. It should also have good target and subject variation. Target variation refers to the different types of labels that a dataset should contain for different types of gaze prediction tasks, including 2D gaze detection, 3D, screen coordinate, and gazed target. On the other hand, subject variation refers to diversity in the people included in the dataset, and viewpoint refers to the placement of the camera with respect to the subject.

The most useful part we were able to glean from the paper is the list of some publicly available gaze prediction datasets, the most relevant of which are MPIIGaze and GazeCapture. MPIIGaze is a collection of 200K+ images collected from 15 participants with 2D and 3D annotations. The original dataset contains just the eye region of the face, but there is variation in head pose,gaze angle, lighting conditions, and accessories such as glasses. GazeCapture also deals with 2D gaze-estimation including 2.4M and 1.4k + subjects.

Given the target and subject variation of the MPIIGaze dataset, especially in relation to our unique problem, we identified MPIIGaze as a possible foundation towards building upon a model. Under further analysis by Zhang et al. [11], a proposed method that was used to train the MPIIGaze model showed 6.3 degrees of mean error when training this dataset on the model. Given these findings, we decided to build upon this model and tune its parameters to improve it (more information about this process of which are detailed in the methods section)

As part of our research, we compared our own MPIIGaze model to our own baseline model being an OpenCV implementation (discussed in later sections). Our baseline implementation follows Chunning Meng and Xuepeng Zhao [6] who propose a novel approach to eye movement analysis using a webcam and Convolutional Neural Networks. They identified the relevant problem of low-quality webcam videos limiting the accuracy of eye tracking and eye movement analysis. To address this issue, they proposed an eye movement analysis model based on five eye feature points rather than a single point (such as the iris center) in order to get more useful eye movement information and reduce the dependency on the iris center in low-quality videos. The methodology involves training a Points-CNN for eye feature point detection, detecting those five feature points, and then having six types of original time-varying eye movement signals constructed by the feature points of each frame. Then, a Behaviors-CNN is trained on these time-varying eye movement signals for recognizing different eye movement patterns, an approach that aims to avoid the influence of errors from basic eye movement type detection and artificial eye movement feature construction. The performance of this model was then validated on a webcam-based visual activity dataset (WEActivity) containing 0.5 million frames collected from 38 subjects (with 2 wearing glasses) and achieved a mean of 81.3 percent precision and performed better compared to other models at the time. Given this success, we can confirm that our baseline method functions as a valid metric of comparison.

3. Technical Approach

3.1. Problem Architecture

Our problem was defined as an appearance-based gaze prediction problem. Our work was concentrated in the gaze prediction problem since it is a regression problem with many variables to account for, including lighting, subject diversity, head position and orientation, and camera variations. The problem was broken down into a number of constituent parts shown below in Figure 1, which will help define the specific parameters of each subproblem as well as guide the potential architectures that can be used.

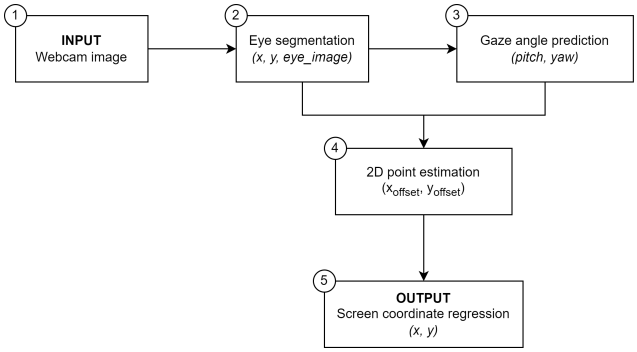

Figure 1. Shows how our problem can be modeled as a sequence of subproblems. Connecting different parts of the graph would represent different kinds of solutions - for example, a purely end-to-end solution can be thought of as a direct connection from the webcam image input (1) to predicted screen coordinates (5).

3.2. Considered Alternative Approaches

We initially considered an implementation featuring the GazeCapture dataset, however, we soon realized the dataset probably wasn't ideal. This dataset took images via mobile devices, which has a different camera location, screen resolutions, and screen-face distance than traditional webcams. This would make the pitch/yaw to screen coordinate calculation more complicated than originally anticipated. Furthermore, the dataset was very massive and would exceed the storage limits in Google Colab, which would've made training our model infeasible.

Our next solution was to create our own dataset using our own computers. We would write a program to capture our faces and screen-coordinates in which we were supposed to be looking. We would then feed this data into the MPIIGaze Estimation model. Unfortunately, this method also fell through as taking and manually labeling our images (which is what the paper did) into the same structure that this model needed would've taken too long, and required a lot of image data that we couldn't create on our own.

3.3. Chosen Method

Instead of augmenting our own data to the MPIIGaze model, we decided to modify the model itself. By augmenting the newly minted model with some additional coding that we wrote we were able to cover all 5 steps from figure 1. The first 3 were all completed by modifying the MPIIGaze model. The general structure of this model is outlined in Figure 2. Starting with the image input (step 1), the model first uses a pretrained Dlib CNN model to identify facial landmarks (step 2). These landmarks are compared to a generic, straight-on face model to retrieve the head position and eye location. This data is then used to segment the eye (focus on just the eye area of the image). To do this, Zhang et al uses the same method that Sugano et al in which they synthesize a new 3D model of just the eye, and translate it to a 2D image using polar coordinates [10]. From there, the program feeds this data into a CNN (Convolutional Neural Network), such as LeNet, whose architecture is seen in figure 3. This ultimately allows us to predict the gaze (pitch and yaw) of the eye.

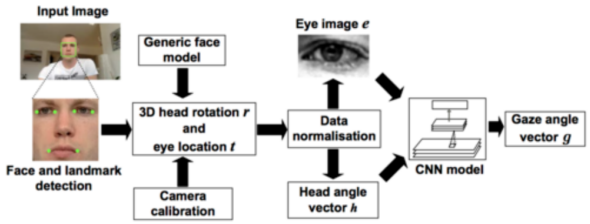

Figure 2. This diagram is taken from [11] it outlines the Preprocessing done to get from the raw data to the segmentation of the eye.

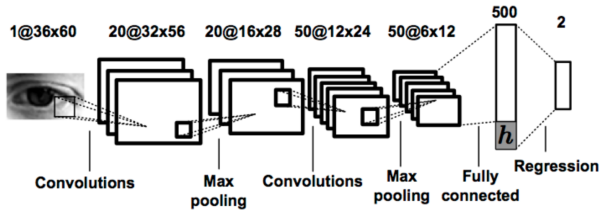

Figure 3. This figure, taking from [11], represents the LeNet architecture.

Sewell and Komogortsev [8] used a multimodal CNN model to learn the mapping from the input features to gaze angles in the normalized space, the input features including both eye image and head pose information.

For step 3, previous methods and implementations utilized different CNN architectures trained on the MPIIGaze Dataset, and for our methodology, we will be using CNNs as well while experimenting with different hyperparameters and architectural choices that will be discussed in our Experiments and Results section. Our justification for continuing to use CNNs is because they have the ability to automatically learn hierarchical representations and features from raw pixel data and have been shown to be effective at finding patterns in images. In fact, Dlib's face and landmark detection models are based on CNNs for their ability to extract such features. CNNs can also learn hierarchical representations, where lower layers capture local features such as edges and textures, and higher layers capture more abstract aspects. This is beneficial for gaze detection, as the direction of gaze can be influenced by both local features (such as the shape of the eye) and more global features (such as the position of the head). CNNs are also robust to variations in scale, rotation, and translation, which is beneficial when the appearance of the eyes can vary based on how far a person is from the webcam, the angle of the head pose, and of course the direction of the person's gaze. Finally, CNNs can be trained end-to-end, so the entire model can be trained and learn to map directly from raw image (video frames) input to gaze output angles in terms of pitch and yaw. Thus, based on proven performance and usage of CNNs in the field of deep learning and the possible benefits that it could provide, we believe that CNNs should be well-suited for gaze detection in our project.

Moving onto Steps 4 and 5 , our goal was to translate our pitch and yaw (gaze direction) towards actual input movement on our screens. To do this we took a running average of the pitch and yaw and subtracted it from their current values (this essentially normalized the data). Thus, we were able to map pitch and yaw to screen coordinates thereby completing step 4. Finally, using the PyAutoGui library we were able to move the mouse towards the specified coordinates that we had translated from pitch and yaw in the prior step.

With those steps completed, we concluded our model from steps that covered steps 1-5.

3.4. Baseline Method for Comparison

In order to test our new model, we needed an established one for comparison. For our baseline implementation, we used OpenCV and Google's MediaPipe facial landmarker, [4], which is a pre-trained end-to-end resnet-style model that predicts dense facial landmarks in real time. Similar to Meng et. al, we use five landmarks, the left and right corners and the top and bottom landmarks of the eye and the center of the pupil, to predict the gaze of the user. Multiplying the relative position of the pupil by the viewport size, we can predict the coordinates of the user's gaze. To calibrate the tool, the user looks at the corners of the screen and the relative pupil position is recorded. A screenshot of the landmarks that are generated by MediaPipe is shown in

Figure 4.

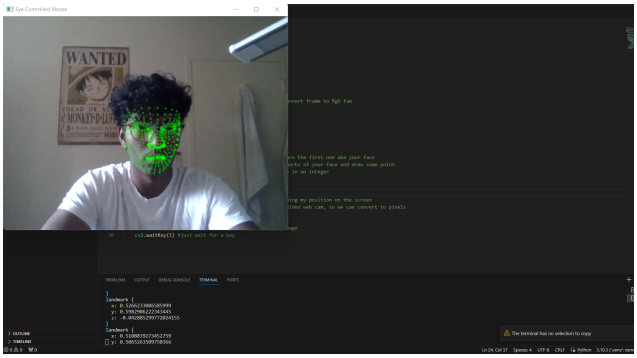

Figure 4. Landmarks generated by MediaPipe which we use for eye segmentation and pupil detection.

3.5. Evaluation Metrics

In order to evaluate the effectiveness of the model, we recorded two metrics: jitteriness and tracking time. Jitteriness refers to the precision of the model's prediction taken across time when the user is focused on a stationary point on the screen, and we express it as the standard deviation of the distance each prediction is from the previous. Tracking time refers to the average amount of time it takes for the user to successfully navigate the cursor to a desired location on the screen with a radius of 100 pixels, starting from an arbitrary point.

We measured tracking time by running an OpenCV program that displays 20 random targets on the screen that the user has to navigate to with their cursor. We then took the average time it took to navigate to each target. To measure jitteriness, we asked the user to fixate to the center of the screen without moving their eyes. The variance in the predictions in both the horizontal and vertical directions were then logged to calculate jitteriness. It is important that both tracking time and jitteriness are low in order to ensure a smooth user experience and the model has a practical application. We used this, along with mean error to evaluate the effectiveness of our model.

4. Data

The dataset we have used in our approach is the MPIIGaze Dataset [11] ,a collection of 213,659 images from 15 participants (with varied number of images provided from each client). These images were collected using a custom software that (while running in the participant's background) required participants to look at a randomly generated sequence of 20 on-screen positions every 10 minutes. These onscreen positions were visualized as white dots encircled by a shrinking gray circle. As no other instructions were given in this procedure, the dataset is quite diverse as a result; with images ranging from various forms of lighting, camera quality, glasses, angling and even distance from screen. Each datapoint had 41 dimensions indicating landmark positions in the eye, headpose, etc.

This data set was chosen because the manner in which the photos were taken is of a similar use case to a computer webcam, thus the motivation of its creation closely matched ours and its collection process was similar to the user input in our program. The composition of the data set was also key. It included multiple raw instances of different environments, lighting, diverse people and backgrounds, which could then be preprocessed to segment the eye. These are all key characteristics outlined by Patharina et al [7] as well.

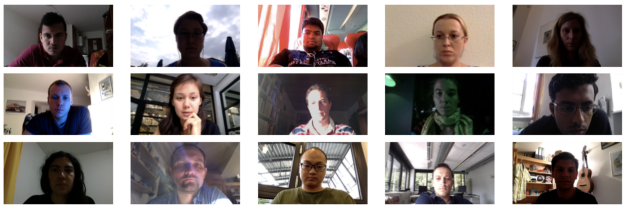

Figure 5. Taken from [11], this is a sample of images from our dataset

5. Experiments and Results

5.1. Overall Model Accuracy

To reach our proposed version of the MPIIGaze model, we began by exploring previously published methods/implementations of the MPIIGaze paper such as "An unofficial PyTorch implementation of MPIIGaze and MPIIFaceGaze" which can be found here [2] that trained on older architectures such as AlexNet and ResNet-18 on the MPIIGaze Dataset. Whiler these architectures had impressive results, with the Mean Test Gaze Angle Errors of 5.06 and 4.83 degrees, respectively, we thought that we could use a more modern/newer architecture such as ResNet-50 to utilize more convolutional layers/depth, residual learning, a new bottleneck design, and fine-tune it to increase the accuracy and performance on the MPIIGaze Dataset to achieve a lower Mean Test Gaze Angle Error compared to older, less deep architectures. Thus, to improve the MPIIGaze model, using the deep learning framework PyTorch and the previous implementation code as a base, we modified the model implementation and trained the ResNet-50 architecture using the MPIIGaze dataset while fine-tuning the hyperparameters such as a higher learning rate being 0.05 to prevent the training process from getting stuck at a set of sub-optimal set of weights, batch size of 32 to get a better accurate estimate of the gradient, optimizer being Standard Gradient Descent (SGD) to update the model's weights using the gradient of the loss function with respect

to the weights and increase the speed of convergence, loss function being SmoothL1, which is a combination of L1-loss and L2-loss, to prevent exploding gradients, and 15 epochs to give the model ample enough of time to train and decrease the loss as much as possible. The training angle error of the ResNet-50 model is shown in Figure 6.

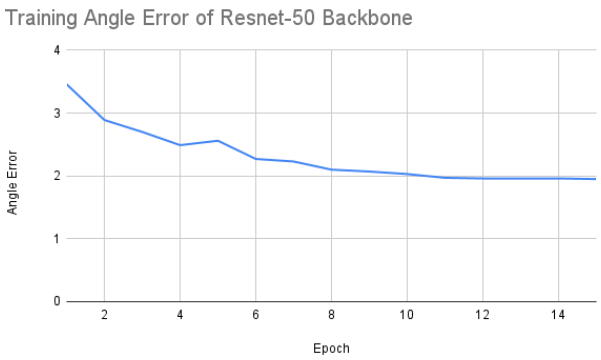

Figure 6. The angle error of the predicted pitch and yaw values of the Resnet-50 backbone trained on the MPIIGaze dataset

The input image is 224x224 with 3 channels being BGR so we reverse the channel order of the first convolutional layer since the pretrained model for ResNet-50 was trained using images with RGB channel order. For forward pass, we extract relevant features from the input image such as the facial features and landmark detection using a pre-trained dlib CNN model as discussed in our gaze estimation pipeline, process these features using convolutional operations and non-linear activation functions, focus on the most relevant parts of the image using an attention mechanism, and finally predict the direction of the gaze using a fully connected layer. After training and tuning, the lowest Mean Test Gaze Angle Error after evaluation with our ResNet-50 architecture model was 3.40 degrees, which is a good improvement.

Architecture	AlexNet	ResNet-18	Our Model
Mean Test Gaze Angle Errors (Deg)	5.06	4.83	3.40

Figure 7. Compares the Mean Test Gaze Angle Errors in Deg between different architectures

Using "A demo program of gaze estimation models", which can be found here [1] as existing code to start with to visualize the live performance of our model for gaze detection by bringing in our own trained checkpoint to test, we also had to experiment with different methods in projecting pitch and yaw towards screen coordinates. The primary issue was modeling the pitch correctly relative to mapping the mouse onto the screen. We were first able to do the yaw sufficiently well because the angle of our eyes relative to the webcam was less varied than the pitch moving horizontally. However, when we were moving our eyes vertically (pitch) on the screen , the movement wasn't as accurate or consistent because, in general, left to right eye movements are much more obvious and clear than up and down movements, as there is more space to move horizontally rather than vertically. This is further exemplified in the results below where the jitteriness for pitch is consistently much higher than for yaw. To reduce this, we used our aforementioned running average/normalization solution, and we were able to map pitch and yaw to screen coordinates by making the eye movements proportional to the screen size.

Figure 8. Example of working gaze detection using ResNet-50 Checkpoint

In our evaluation process, we utilized the baseline model and assessed its performance based on the proposed metrics. Our test subjects consisted of volunteers from our apartment and our team, providing a diverse range of individuals for a more comprehensive evaluation. The data we collected about jitteriness and tracking time for the baseline approach from our milestone is shown in Table 1. We then collected these metrics for different architectures using the MPIGaze dataset (ResNet-50, ResNet-18, AlexNet) as well as for our baseline displayed in Figure 1.

Method	Jitteriness (pixels)	Tracking Time (seconds)
Baseline	201 px	36 seconds

Table 1. Jitteriness (variance of each prediction) and average tracking time per target (20 targets, 100 pixel radius) metrics for the Baseline OpenCV/MediaPipe+5 landmarks model.

6. Conclusion

Analyzing the results from Table 2, we can see that architectural differences in conjunction with using the MPIIGaze dataset showed little to no change in terms of jitteriness in

Approach	Jitteriness (x/yaw) (px)	Jitteriness (y/pitch) (px)	Tracking time (s)
MPIIGaze dataset + ResNet-50	15.43	21.79	4.76
MPIIGaze dataset + ResNet-18	17.28	22.61	4.33
MPIIGaze dataset + AlexNet	13.23	20.29	4.50
Google MediaPipe Baseline (no smoothing)	119.77	178.40	13.23
Google MediaPipe Baseline (smoothing)	33.99	153.94	4.85

Table 2. Comparison of the different architecture implementations relative to our baseline implementation

both pitch and yaw as well as tracking time. However, when it came to the differences in comparison to our baseline implementation, we found that our implementation ultimately did better when it came to these metrics as indicated in Figure 5. In other words, it was more accurate/consistent than the baseline when it came to predicting the general direction of the user's gaze and moving the mouse in that direction. However, both approaches have their advantages and disadvantages. The MPIIGaze models are more robust and do not require calibration - however, they take a long time to train and run slowly at 5-10 FPS, which makes them less practical as a real-world tool. The MediaPipe baseline requires calibration every time it is used and generates jumpy predictions, but it is faster and uses a lightweight, completely pretrained backbone, making it a more widely applicable approach that can be used even on mobile devices. Ultimately, we were able to solve our problem to some degree. However, for future extensions or new applications of our ideas, if we had more time, we would like to be able to utilize deep learning models to detect blinking and winking and translate them to left and right clicking. This would allow for entirely-hands free computer navigation and substitute for use of a physical mouse device.

7. Team Contributions

Kevin Kwan: Technical Portion: Trained ResNet-50 architecture and Fine-Tuned Model Parameters on the MPII Dataset Completed a paper review Completed Evaluation Metrics Found Data set Wrote about method Wrote about experiment

Hengyuan Zhang: Technical Portion: Implemented method which gave the ability to evaluate the two models Transcribed our approaches section and also created the approaches diagram Completed a paper review

Hari Kumaran Technical Portion: Implemented Baseline method. Completed a paper review Worked on intro sections Wrote about dataset Worked on Poster

May Kalnik: Technical Portion: Created functionality to map eye gaze to computer screen movement Completed a paper review Worked on intro sections Wrote about Dataset Transcribed References Worked on Poster

References

[1] A demo program of gaze estimation models (mpiigaze, mpiifacege, eth-xgaze) [computer software]. 2022.

[2] An unoffical pytorch implementation of mpiigaaze and mpiifacegaze [computer software]. 2022.

[3] Christina Lassen abd Sigurd Mikkelsen, Ann I kryger, and John H Andersen. Risk factors for persistant elbow, forearm and hand pain among computer workers. *Scandinavian Journel of Work, Environment Health*, 31(2):122–131, 2005.

[4] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus, 2019.

[5] Eugenia Kim, De'Aira Bryant, Deepak Srikanth, and Ayanna Howard. Age bias in emotion detection: An anlysis of facial emotion recognition performance on young, middle aged, and older adults. *Proceedings of the 2021 AAAI/ACM Conference of AI, Ethics, and Society*, pages 638–644, 2021.

[6] Chunning Meng and Xuepeng Zhao. Webcam-based eye movement analysis using cnn. *IEEE Access*, 5:19581–19587, 2017.

[7] Primesh Pathirana, Shashimal Senarath, Dulani Meedeniya, and Sampath Jayarathna. Eye gaze estimation: A survey on deep learning-based approaches. *Expert Systems with Applications*, 199:116894, 2022.

[8] Westeon Sewell and Oleg Komogortsev. Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network, 2010.

[9] Abdallah Sham, Kadir Aktas, Davit Rizhinashvili, Danila Kuklianov, Fatih Alisinanoglu, Ikechukwu Ofodile, Cagri Ozcinar, and Gholamreza Anbarjafari. Ethical ai in facial expression analysis: racial bias. *Signal, Image and Video Processing*, 17, 05 2022.

[10] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Learning-by-synthesis for appearance-based 3d gaze estimation, 2014.

[11] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild (mpiigaze), June 2015.