Eye-See-You: Eye Gaze Prediction using Deep Learning

Anita de Mello Koch
ademello

Ji Won Chung
jchung97

Arthur Chen
kchen157

Skye Thompson
rthomp12

December 2022

1 Introduction

Heightened isolation and unscalable health infrastructures during the pandemic has made the need for better social systems acutely clear. Recent research suggests there may have been an increase in suicide rates and depression rates during the pandemic and that preventive interventions are very much lacking. Particularly for those in remote locations or with restricted access to mental health care, individualized care is often too costly.

As a result, virtual assistants have been proposed as alternatives to help facilitate scalable access to health care. While language models like GPT-3 have advanced in creating naturalistic speech, they are often still lacking when fused with a visually virtual agent because the agent lacks body languages cues. For an agent to feel natural, it must *act* natural as well.

We hope to start moving towards this goal by simulating natural eye gaze. In this project, we predict the next eye position based on a given trajectory. One of the factors of natural eye gaze is also blinking. So not only would we need these renderings to simulate the trajectory of a human gaze, but also to blink at appropriate points. As such we also attempt to predict when the rendering should blink to simulate human eye interactions.

2 Methodology

2.1 Data and Data Processing

This project has has two main steps: (1) using WebGazer's dataset to extract webgazer's 120 eye gaze trajectory features and (2) using WebGazer's system to extract MediaPipe's face mesh to calculate Eye Aspect Ratio for blink detection.

2.1.1 WebGazer Features

The WebGazer dataset collects eye gaze data from everyday laptops with relatively normal or low resolution 640 x 480. We deemed this appropriate if we truly wanted to design a scalable model for everyday users. Users completed tasks while their inputs, screens and faces were recorded. These face recordings were collected using the device's webcam and was used to collect eye features using the WebGazer tool. Additionally, the eye-gaze locations were collected using the Tobius Pro X3-120 eye tracker but was not used for this project.

The eye gaze trajectories are made up of eye features which are 120-dimensional vectors. We task our deep learning models with predicting the next eye feature in the trajectory. The trajectories were divided into window-size length groupings, with the next feature in the trajectory used as the prediction label. Due to time restraints, data from 19 participants was used as the eye feature extraction takes a large amount of time.

2.1.2 Eye Aspect Ratio from MediaPipe's Face Mesh

WebGazer also utilizes Google's MediaPipe framework, which extracts 468 key facial features in 3D. Figure 2 demonstrates an example of Media Pipe's face mesh. From these, we extract the a pair of 6 key eye features for each eyes and calculate the eye-aspect-ratio (EAR), a metric created by Soukupová et al.. EAR basically is a simple calculation of the ratio between the horizontal distance of the eye and the vertical distance of the eye.

$$EAR = \frac{\|P2 - P6\| + \|P3 - P5\|}{2 \times \|P1 - P4\|}$$

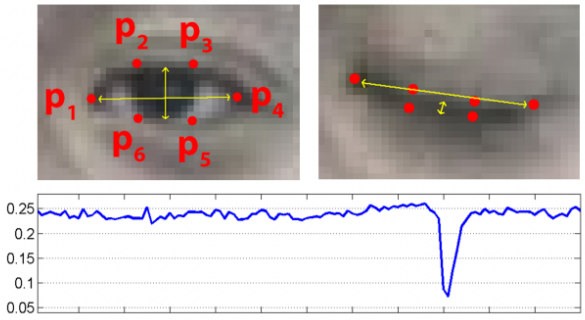


Figure 1: Demonstrates eye aspect ratio (EAR) and corresponding blink detection. Points P1-P6 mark the important key features. The bottom graph shows EAR values over several frames. One can see that the negative peak on the graph indicates a blink because the EAR is less than 0.1. Source: Real-Time Eye Blink Detection using Facial Landmarks

Following Fernando et al.'s study on EAR thresholds, we used the EAR threshold of 0.2 to consider it a blink. Anything above the threshold indicates that eyes are open. Soukupová et al.'s method extracts 6 points, P1-P6 where P1, P4 are the left and right corners of the eye, P2 and P3 are the corresponding upper corners of the pupils on the eyelids, and P6 and P5 that of the lower lids. Figure 1 demonstrates an example of blink detection using EAR.

The left eye face mesh indices are [362, 385, 387, 263, 373, 380] and [33, 160, 158, 133, 153, 144]. Each element in the array correspond to P1, P2, P3, P4, P5, and P6 for each eye accordingly. So if EAR was less than 0.2, we mark it as 'blink'. Each frame is associated with a binary feature indicating 'blink' vs 'not-blink'. The intention is to use these eye feature vectors as the input for a later task which would generate the rendering. Much of the code to extract EAR points from Media Pipe was referenced and adapted from Driver Drowsiness Detection Using Mediapipe In Python.

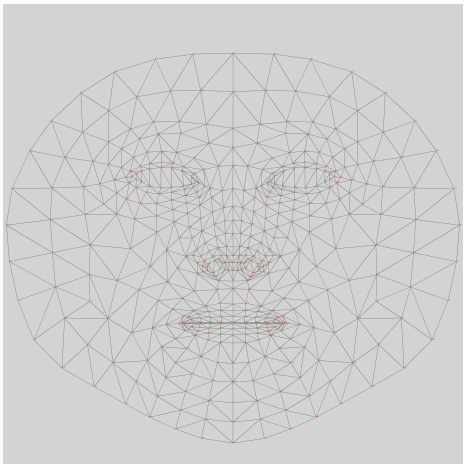


Figure 2: Image of MediaPipe's Face Mesh. Each vertex of the mesh is labelled with a unique index, marking the facial feature. Source: Canonical Face Model UV Visualization

2.2 Eye Gaze Trajectory Prediction

We begin by first predicting eye gaze trajectories without incorporating blink data. This is because the addition of blink prediction is a harder task so we investigate the best architecture without this additional complexity. We investigate three different model architectures

1. CNN: A single 1-dimensional CNN layer takes in the full trajectory. This is followed by another 1-dimensional CNN layer followed by two dense layers. Figure 3 details the network architecture.

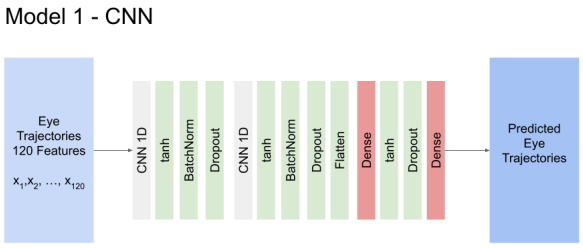


Figure 3: CNN Architecture

2. Split CNN: We create a number of 1-dimensional CNN layers equal to window-size. Each eye-feature in the trajectory is fed into one of these CNN layers. In other words, each CNN takes in a specific timestep of the trajectory. The output of each CNN layer is concatenated together and then fed into two dense layers. Figure 4 details the network architecture.

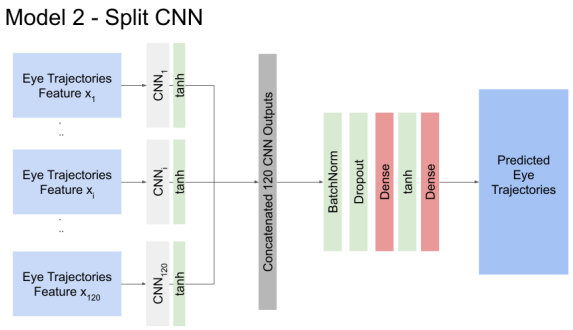


Figure 4: Split CNN Architecture

3. LSTM: An LSTM layer takes in the full eye feature trajectory. The output of this layer is then fed into two dense layers. Figure 5 details the network architecture.

All architectures use `tanh()` activations because the eye features can be positive or negative. Batch normalization and dropout are also utilized after all CNN and LSTM layers. Additionally, the different eye features can vary in size from -300 to 300 with some usually in the high hundreds and others usually around 0. Because of this large variation we did not normalize the features as it is unclear how the data should be normalized to prevent loss of information. As such, no activation was used for the last layer of any model. The eye gaze trajectory

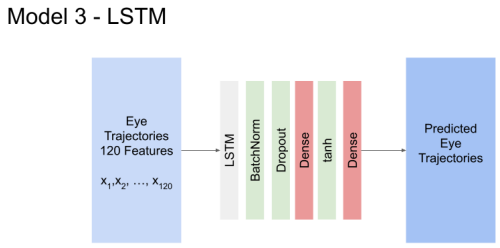


Figure 5: LSTM Architecture

prediction can be seen as a regression problem and so the Huber loss function was used. Additionally, the `Adam` optimizer was used for all experiments.

We do a hyperparameter sweep over the window-size and learning rate to find the optimal architecture out of the three described above. The mean square error (MSE) is used as the performance metric. Note that the mean square error numbers do appear to be high (e.g. between 4000-2000) but due to the large numbers that make up the eye feature vectors this is to be expected. So a MSE of 3000 translates to an average error of around 54 on a range of numbers that can very from -300 to 300.

2.3 Eye Gaze Trajectory Prediction with Blink Detection

We found the Split CNN performed best out of all tested architectures and so modified this architecture to include the blink data. Figure 6 details the network architecture.

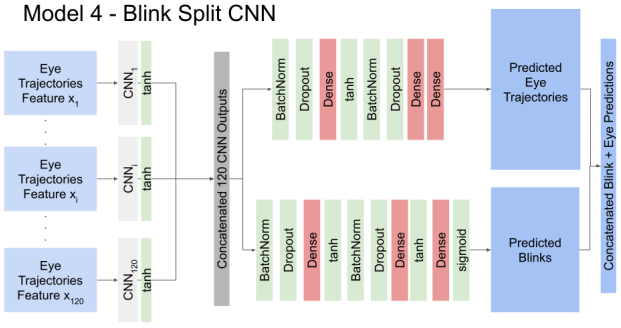


Figure 6: Split Blink CNN Architecture

We append the blink data to the end of the eye feature vector and feed

this into the 1-dimensional CNN layers. We then have two separate sets of dense layers, one of which predicts the next eye feature and the second predicts the next blink state. The loss on the eye feature prediction and blink state prediction are calculated separately and summed. This translates to a model that shares the latent space which comes from the CNN layers but have separate dense layers. Additionally, calculating both losses separately ensures that the blink feature does not get overshadowed by the large number of eye features.

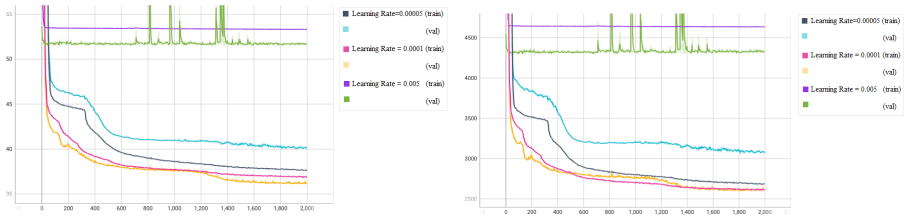
3 Results

3.1 Eye Gaze Prediction Results

We evaluated each model, varying learning rate while keeping window size fixed and varying window size while keeping learning rate fixed. This was done to reduce the number of runs required due to time restraints and while this may not lead to the optimal set of hyperparameters it gives us an indication of where to focus more compute power.

Across all architectures (Figures 7, 9 and 11) we can see that too high a learning rate leads to deteriorated performance. Similarly, we find that larger window sizes are beneficial to all models (figures 8, 10 and 9), particularly the LSTM.

Overall it seems the Split CNN performed best out of the three models. The Split CNN has a dedicated CNN layer for each element in the input trajectory and so has more parameters than the other two models. Additionally, the nature of eye trajectories has a strong dependency on the last few time steps, i.e. requiring a very short memory, which may be harder for the LSTM to capture but easier for a CNN.

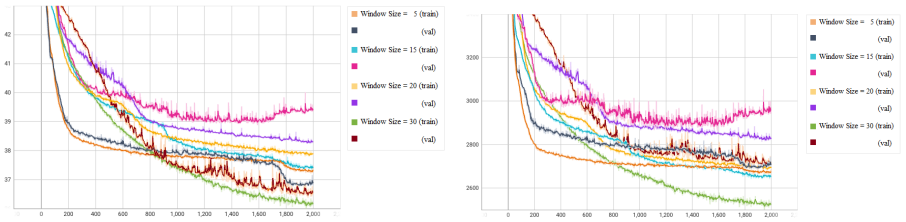


(a) CNN Model Huber Loss (b) CNN Model MSE

Figure 7: CNN model performance with varied learning rates. Window Size kept at 10.

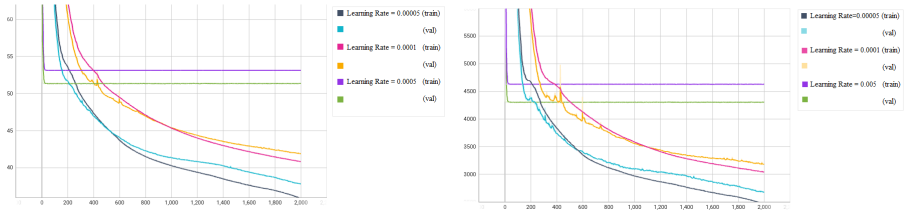
3.2 Eye Gaze Trajectory Prediction with Blink Detection Results

For the eye gaze trajectory prediction with blink detection we measure the MSE for the eye features and train on the Huber loss. For the blink state prediction



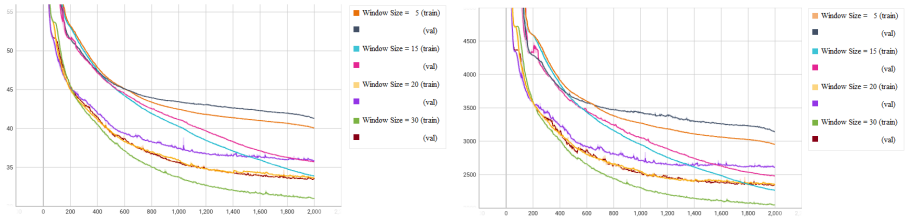
(a) CNN Model Huber Loss (b) CNN Model MSE

Figure 8: CNN model performance with varied window sizes. Learning rate was kept at 0.001.



(a) Split CNN Model Huber Loss (b) Split CNN Model MSE

Figure 9: Split CNN model performance with varied learning rates. Window size kept at 10.

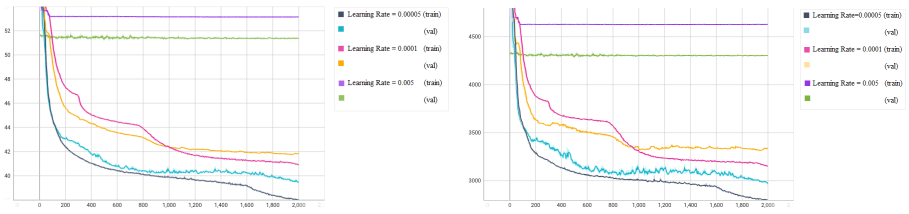


(a) Split CNN Model Huber Loss (b) Split CNN Model MSE

Figure 10: Split CNN model performance with varied window sizes. Learning rate was kept at 0.001.

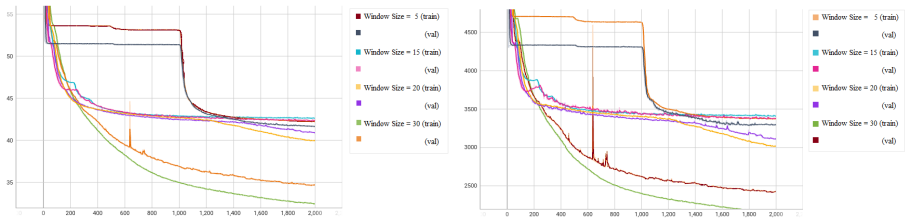
we train on the MSE loss as well as measuring MSE for performance. We very the learning rates to find the optimal learning rate while using a fixed window length of 30.

Some interesting points to note in Figure 13 and Figure 14 is the addition of blink data makes model training a lot more unstable. While this may be due to the dual loss function used it is unclear. Additionally, the models are much more likely to overfit to the training data, obtaining much lower MSE scores than the Split CNN without blink data but also performing poorly on the validation



(a) LSTM model Huber Loss (b) LSTM Model MSE

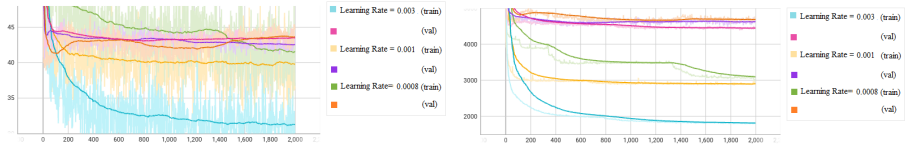
Figure 11: LSTM model performance with varied learning rates.Window size was kept at 10.



(a) LSTM Model Huber Loss (b) LSTM Model MSE

Figure 12: LSTM model performance with varied window sizes. Learning rate kept at 0.0001.

set. This shows that to get good eye gaze predictions may be challenging when incorporating blink information and may require a different approach to what was attempted in this project. The eye gaze trajectory prediction with blink detection also shows that higher learning rate lead to deteriorated performance.



(a) Split CNN Model for Blinks Huber loss (b) Split CNN Model for Blinks MSE

Figure 13: Split CNN with Blinks on Eye Features with varied learning rates. Window size was kept at 30.

v

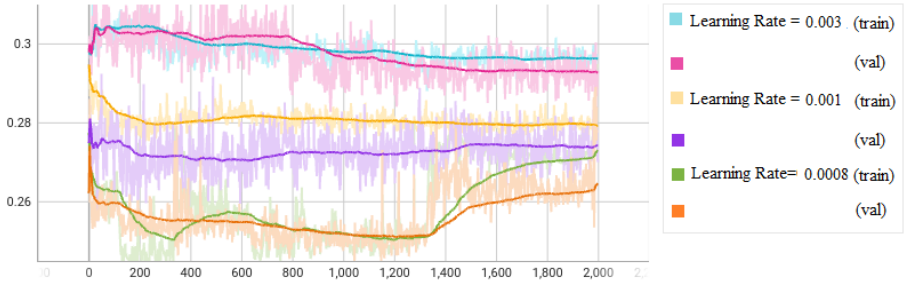


Figure 14: Split CNN model for blinks mean squared error on Blink State for varied learning rates. Window size was kept at 30.

4 Challenges

One challenge encountered during this project was the data processing. The data from the WebGazer dataset must be fed into the WebGazer application to obtain the eye features. This process however takes a long time and there were several software compatibility issues. Additionally extracting the eye information from blinks required identifying specific coordinates that could be matched to MediaPipe's face mesh. The integration of using MediaPipe's API proved to be a bit challenging given the dependency issues.

For training the models, we did have noisy eye features as the WebGazer application does not perfectly extract these features. Additionally, because we did not normalize the eye feature vectors in any way, this did lead to difficulties in training the model and achieving a good MSE.

5 Reflection

It is clear that we have developed models that can learn to predict the next eye gaze trajectory with decent accuracy especially given the highly noisy data. Additionally we tested a total of 3 architectures and modified the best architecture to handle blink information which was more than we had initially expected to achieve. We had, however, hoped to achieve a test MSE below 2000, but never managed to get that low. We had initially planned on using a classification framing for the problem. However, as the project went on it became clear that the eye feature sets are on a continuous spectrum and so regression made more sense for the task.

One aspect that would have been nice to address, however, would have been testing different normalizing techniques for the eye features so that the variation in each feature was not as large would definitely impacted the performance of the model. Additionally, investigating more ways to incorporate blink data into the models could have been a good improvement. Finally, we used the eye features as extracted from WebGazer which are noisy. We could have instead used the

more stable Tobius eye tracker camera outputs for training the model but would have limited the models to requiring this hardware for future training.