Eye Gaze Estimation

Team-19: Chedvihas Punnam
Gnanadeep Settykara
Kavya Alla
Naga Vamsi Bhargava Reddy Seelam
Teja Lam
Vikranth Reddy Tripuram

Introduction:

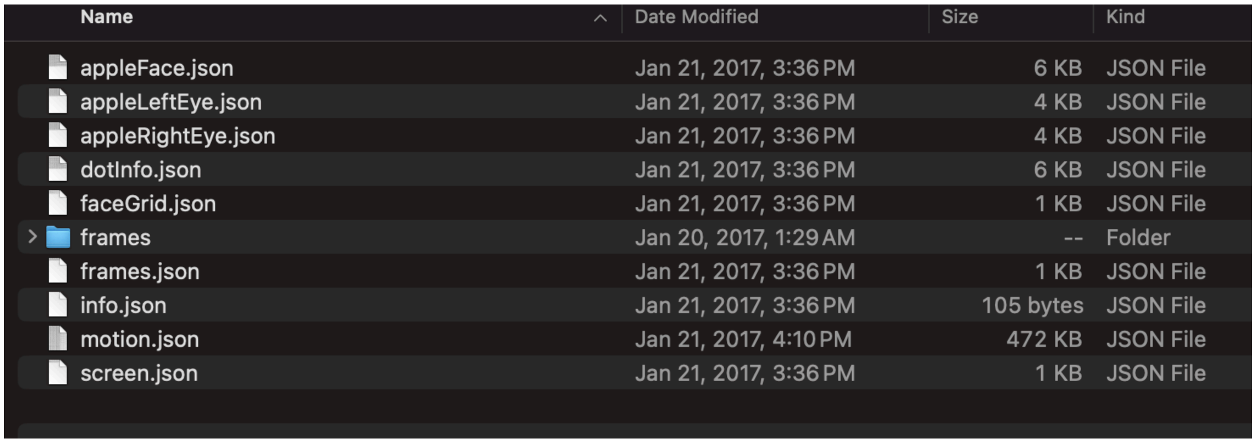
Eye gaze estimation is the process of determining the direction or point of focus of a person's eyes. It involves analyzing the position, movement, and sometimes other characteristics of the eyes to infer where the person is looking. This technology has various uses across different fields.

Eye gaze estimation technology finds application in a variety of domains. In healthcare, it assists in diagnosing neurological disorders such as Parkinson's disease and enables communication for individuals with conditions like locked-in syndrome. In education, it enhances learning experiences by tracking student engagement and attention during lessons, providing valuable feedback to educators. Moreover, in human-computer interaction, it revolutionizes accessibility by enabling hands-free interaction with devices for individuals with motor impairments, facilitating tasks like browsing the web, typing, and controlling assistive devices solely through eye movements.

Dataset [1]:

For this project we are planning to use GazeCapture Dataset. In this dataset there are almost 1474 unique subjects. For each subject there is a directory with a serial number. In each directory there are about 99 frames (in the directory named frames). Apart from this, there are also other JSON files which have the metadata of different elements of the data.

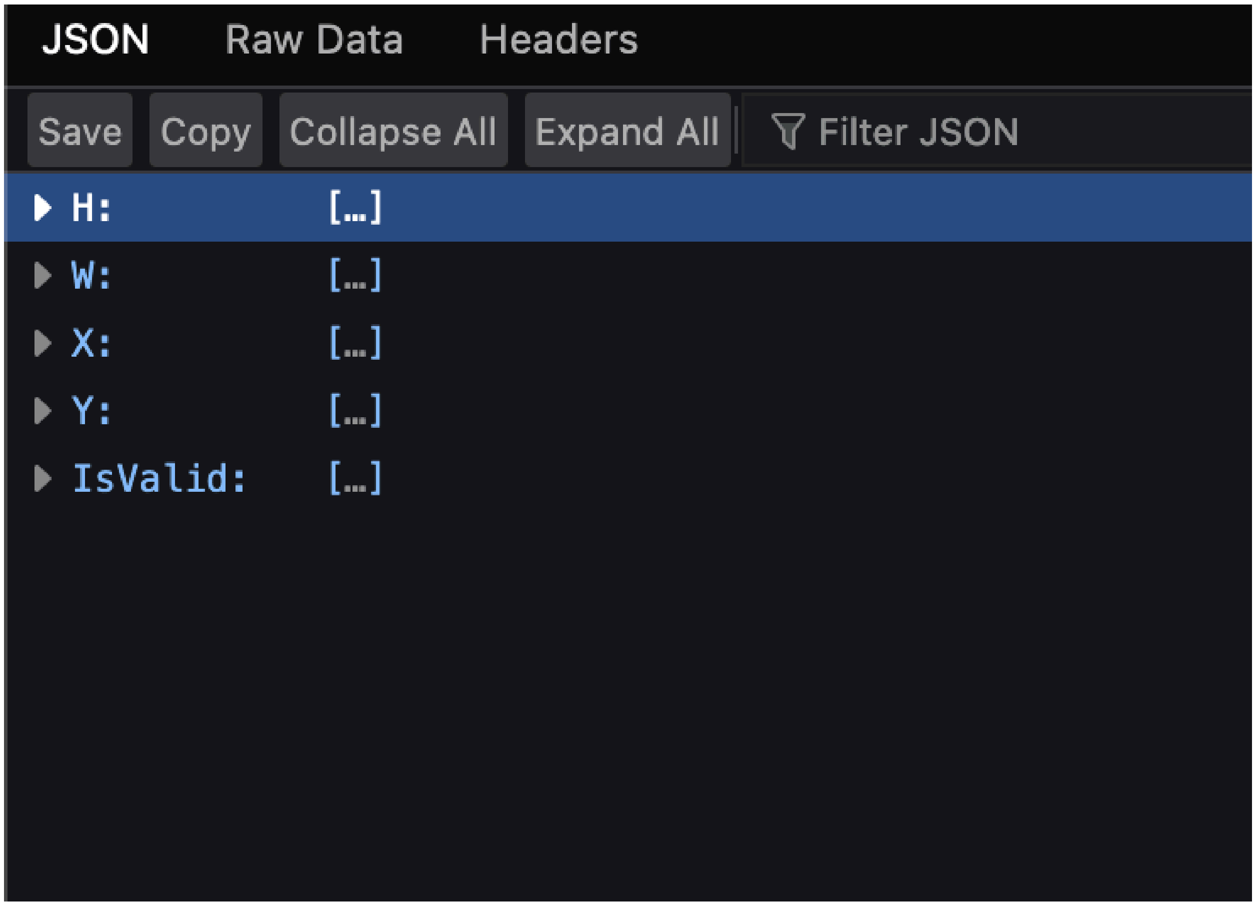
There are almost 2,445,504 total frames and 1,490,959 frames were clear with face and eyes detected properly. In the remaining frames face or eyes werenot detected properly. For this reason, some frames will be "missing" generated data.



Name	Date Modified	Size	Kind
appleFace.json	Jan 21, 2017, 3:36 PM	6 KB	JSON File
appleLeftEye.json	Jan 21, 2017, 3:36 PM	4 KB	JSON File
appleRightEye.json	Jan 21, 2017, 3:36 PM	4 KB	JSON File
dotInfo.json	Jan 21, 2017, 3:36 PM	6 KB	JSON File
faceGrid.json	Jan 21, 2017, 3:36 PM	1 KB	JSON File
frames	Jan 20, 2017, 1:29 AM	--	Folder
frames.json	Jan 21, 2017, 3:36 PM	1 KB	JSON File
info.json	Jan 21, 2017, 3:36 PM	105 bytes	JSON File
motion.json	Jan 21, 2017, 4:10 PM	472 KB	JSON File
screen.json	Jan 21, 2017, 3:36 PM	1 KB	JSON File

Description about each JSON File:

appleFace.json, appleLeftEye.json, appleRightEye.json:

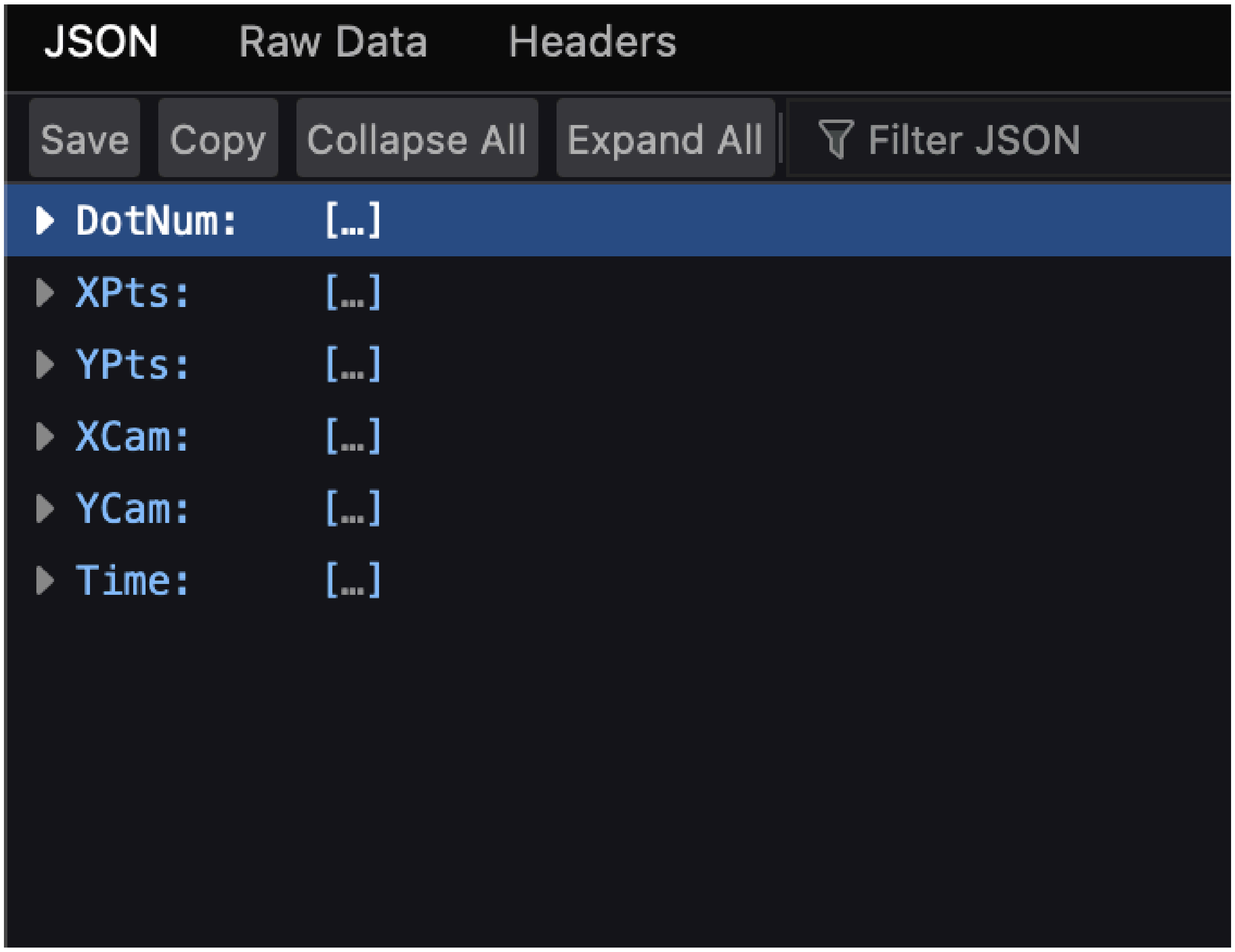


```
H:        […]
W:        […]
X:        […]
Y:        […]
IsValid:  […]
```

appleFace.json, appleLeftEye.json, appleRightEye.json

X and Y represent the coordinates of the top-left corner of the bounding box, measured in pixels. In the appleFace.json file, these coordinates are relative to the top-left corner of the entire frame, while in the appleLeftEye.json and appleRightEye.json files, they are relative to the top-left corner of the cropped face area. W and H denote the width and height of the bounding box, also measured in pixels. IsValid indicates whether a detection occurred: 1 signifies detection, while 0 indicates no detection.

dotInfo.json



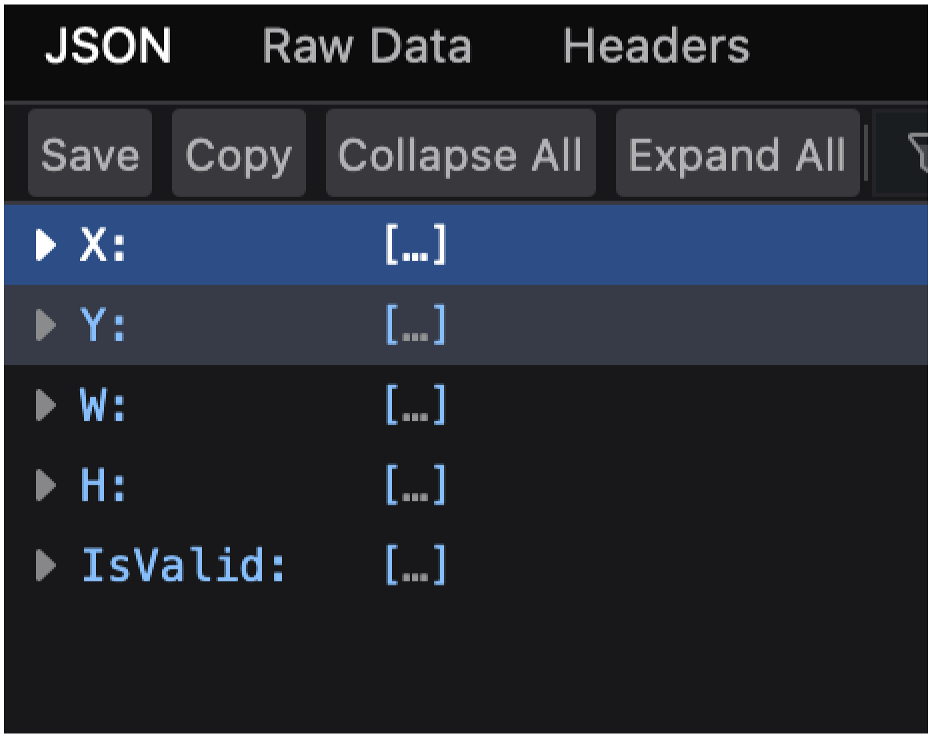
```
DotNum:   […]
XPts:     […]
YPts:     […]
XCam:     […]
YCam:     […]
Time:     […]
```

DotNum: Sequence number of the dot displayed, starting from 0.
XPts, YPts: Position of the dot's center in points, relative to the top-left corner of the screen.

XCam, YCam: Predicted position of the dot's center in centimeters, relative to the camera's center. The values are based on the assumption that the camera remains fixed in space across all device orientations. In portrait mode frames (Orientation == 1), YCam values will be negative since the screen is below the camera, while in upside-down portrait mode (Orientation == 2), YCam values will be positive since the screen is above the camera.
Time: Time elapsed in seconds since the dot first appeared on the screen.

This information helps in understanding the location and timing of displayed dots during eye gaze capture sessions.

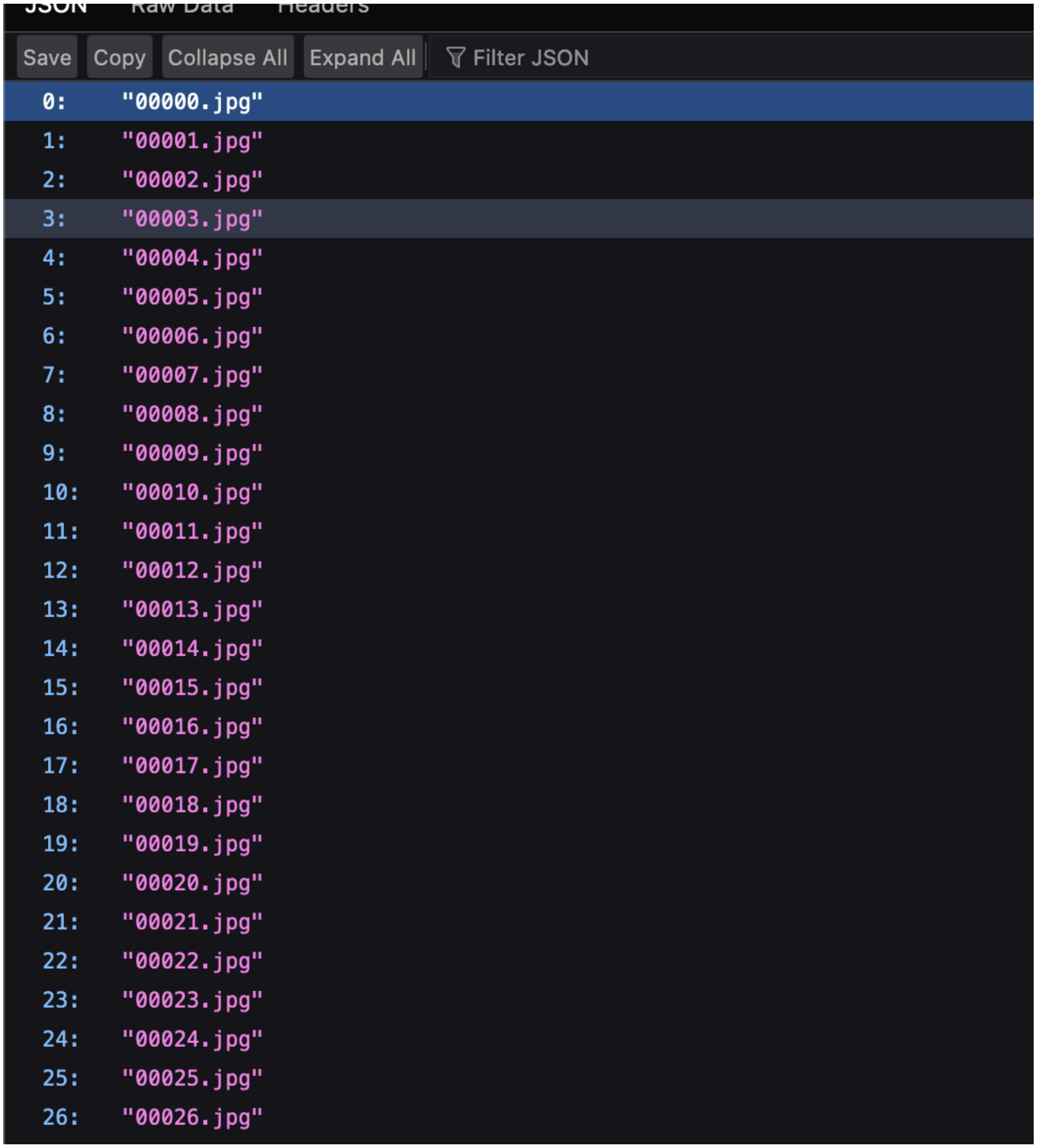
faceGrid.json: Describes "face grid" input features generated from Apple face detections. It specifies the position and size of a detected face within a 25x25 grid.



```
X:          […]
Y:          […]
W:          […]
H:          […]
IsValid:    […]
```

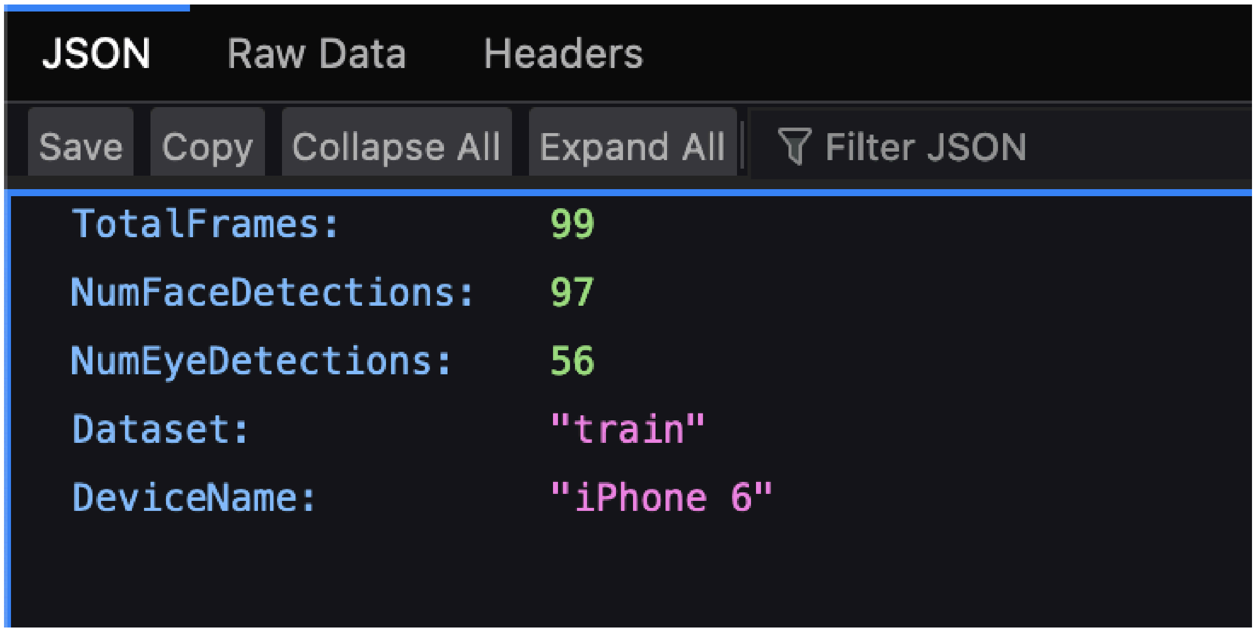
X, Y: Position of the top-left corner of the face box (1-indexed, within a 25x25 grid).
W, H: Width and height of the face box.
IsValid: Indicates whether the data is valid (1) or not (0), corresponding to the intersection of IsValid arrays in apple*.json files.

frames.json: Contains filenames of frames in the frames directory or a sequence number counting from 0 to TotalFrames - 1 (specified in info.json).



```
0:    "00000.jpg"
1:    "00001.jpg"
2:    "00002.jpg"
3:    "00003.jpg"
4:    "00004.jpg"
5:    "00005.jpg"
6:    "00006.jpg"
7:    "00007.jpg"
8:    "00008.jpg"
9:    "00009.jpg"
10:   "00010.jpg"
11:   "00011.jpg"
12:   "00012.jpg"
13:   "00013.jpg"
14:   "00014.jpg"
15:   "00015.jpg"
16:   "00016.jpg"
17:   "00017.jpg"
18:   "00018.jpg"
19:   "00019.jpg"
20:   "00020.jpg"
21:   "00021.jpg"
22:   "00022.jpg"
23:   "00023.jpg"
24:   "00024.jpg"
25:   "00025.jpg"
26:   "00026.jpg"
```

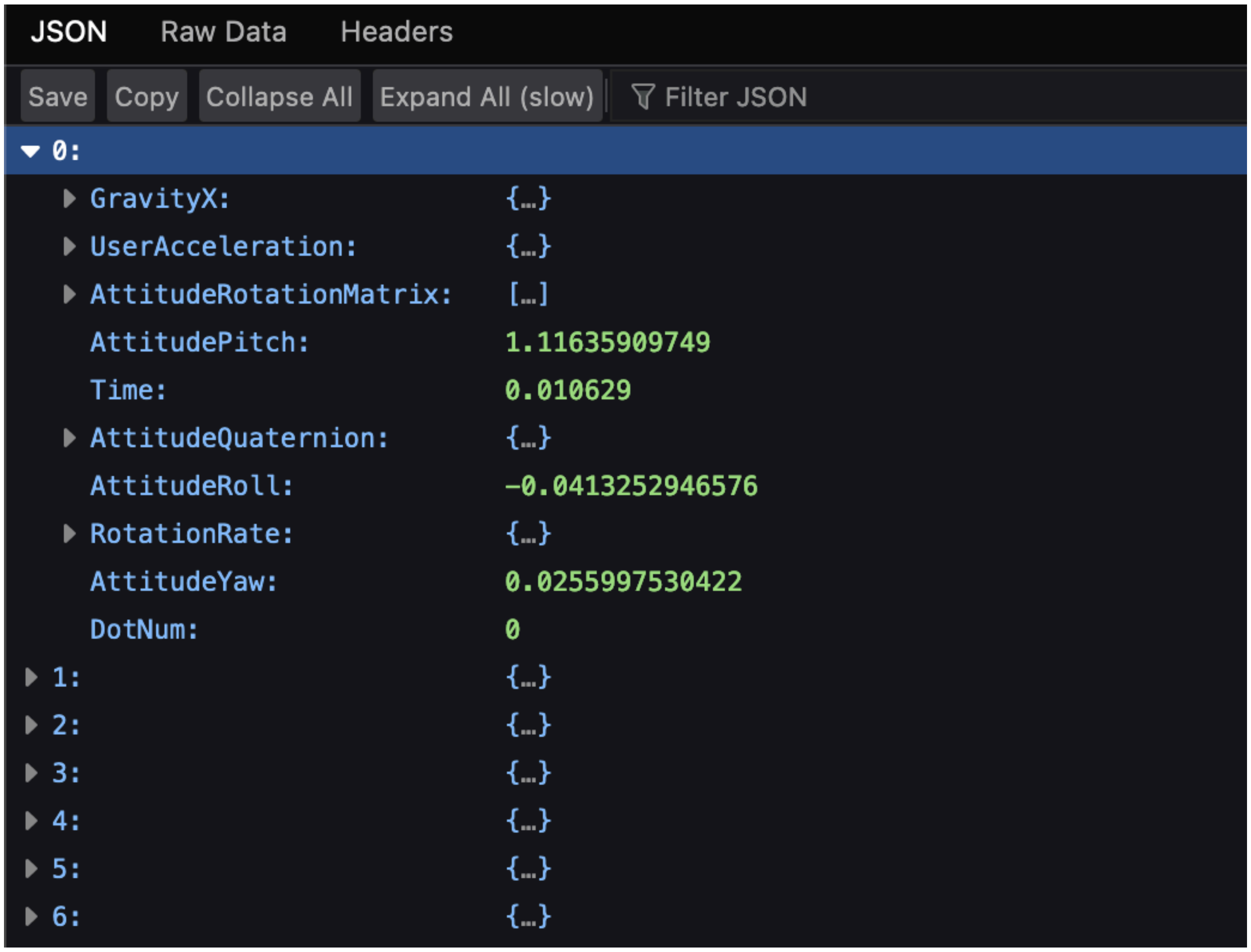
info.json: Provides metadata about the dataset:



```
TotalFrames:        99
NumFaceDetections:  97
NumEyeDetections:   56
Dataset:            "train"
DeviceName:         "iPhone 6"
```

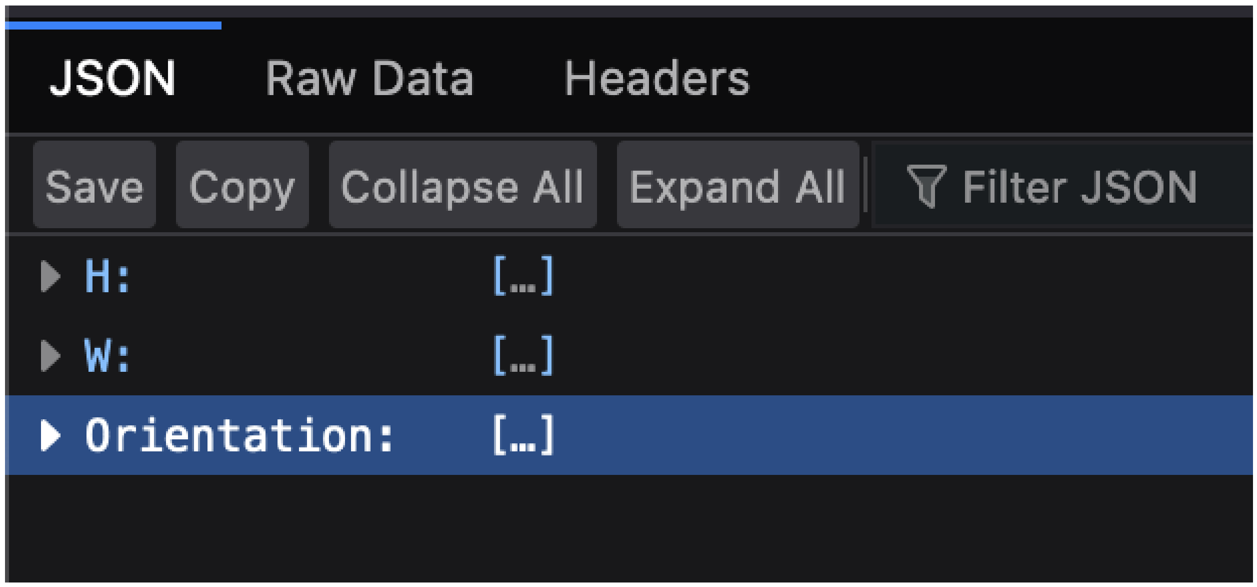
TotalFrames: Total number of frames for the subject.
NumFaceDetections: Number of frames with detected faces.
NumEyeDetections: Number of frames with detected eyes.
Dataset: Indicates if the data belongs to the training, validation, or testing set.
DeviceName: Name of the recording device.

motion.json: Contains a stream of motion data recorded at 60 Hz (accelerometer, gyroscope, and magnetometer) during frame recordings. It also records DotNum (starting from 0) and Time (in seconds from the beginning of the dot's recording).



```
▼ 0:
    ▶ GravityX:                 {…}
    ▶ UserAcceleration:         {…}
    ▶ AttitudeRotationMatrix:   […]
      AttitudePitch:            1.11635909749
      Time:                     0.010629
    ▶ AttitudeQuaternion:       {…}
      AttitudeRoll:             -0.0413252946576
    ▶ RotationRate:             {…}
      AttitudeYaw:              0.0255997530422
      DotNum:                   0
▶ 1:                            {…}
▶ 2:                            {…}
▶ 3:                            {…}
▶ 4:                            {…}
▶ 5:                            {…}
▶ 6:                            {…}
```

screen.json: Provides information about the active screen area of the app:



```
▶ H:            […]
▶ W:            […]
▶ Orientation:  […]
```

H, W: Height and width of the active screen area (in points).
Orientation: Orientation of the interface represented by UIInterfaceOrientation enumeration (portrait, portrait upside down, landscape with home button on the right, landscape with home button on the left).

Models:

We would like to try 4 models that are effective in predicting Eye Gaze and compare how these models work and which one would be better in which scenario.

iTracker [2]:

Introduction:

- **Objective:** The purpose of iTracker is to accurately estimate eye gazing in real surroundings. To this end, it makes use of deep learning techniques.
- **Motivation:** In many applications, such as assistive technology, virtual reality, and human-computer interaction, gaze estimation is essential. The goal of iTracker is to offer a dependable and effective method for determining a person's gaze direction from visual input.

Dataset:

- **Training Data:** A sizable dataset is probably needed for iTracker's training. Videos or pictures labeled with the appropriate gaze direction are frequently included in datasets used for gaze estimation.
- **Annotation:** To enable the model to learn the mapping between visual characteristics and gaze directions, each data point in the dataset would normally contain information on the place where the individual is looking.

Implementation:

- **Framework:** TensorFlow or PyTorch, two deep learning frameworks, may be used in the development of iTracker. The choices of the participating researchers or developers determine the framework to be used.
- **Programming Language:** Python is frequently used for implementations due to its widespread use in the deep learning field.

Architecture:

- **Model Type:** Given that convolutional neural networks (CNNs) are good at capturing spatial relationships in visual input, it is likely that iTracker uses a CNN design.
- **Input:** The architecture of the model is made to extract pertinent information for gaze estimate from pictures or video frames.
- **Output:** The output layer forecasts the direction of gaze, often as a combination of vertical and horizontal angles.

Training:

- **Loss Function:** The difference between the anticipated gaze direction and the ground truth gaze direction is measured by a loss function, which is minimized throughout the training phase.
- **Optimization:** One possible method for optimization is to employ stochastic gradient descent (SGD) or its derivatives, such as Adam.
- **Data augmentation:** During training, methods like rotation, scaling, and flipping may be used to enhance generalization.

Gazel Eye Gaze Tracking Model:

The GAZEL framework introduces a novel approach to gaze estimation by utilizing personalized models based on facial appearances, especially focusing on individuals wearing glasses or other eye equipment [3]. Instead of using a pre-trained TensorFlow Lite (tflite) model, the framework offers training source codes and a data collection tool for developers to build custom models. This framework employs an innovative 5-point calibration method with translation and rescaling. Additionally, it provides the option to use Support Vector Regression (SVR) for calibration, although this method is found to be less effective than linear calibration. The TensorFlow Lite (TFLite) configuration of the framework allows for flexible testing of various model inputs, including options for input modes such as Euler angles, facial positions, and eye grids. Although the framework is initially designed for tablet computers, it includes guidelines on how to modify it for use with smartphones, including settings for device configuration.

Unlike other gaze estimation techniques, the GAZEL framework uses front camera frames and mobile embedded sensors instead of relying on large picture datasets for training. This approach leverages user-specific facial traits and sensor outputs to improve the accuracy of gaze estimation. As the framework is freely available, it represents a significant addition to the field, providing researchers and developers with an adaptable platform for testing gaze tracking on mobile devices. To enhance model training, it is crucial to emphasize the value of collecting a variety of data under different lighting conditions and head postures. Overall, GAZEL offers a revolutionary framework for tailored and precise gaze estimation on mobile devices by combining sensor data with facial features. Compared to other eye tracking techniques like gaze estimator and eye tracker, GAZEL stands out for its unique approach and objective, offering a new perspective for eye tracking models in smart phones, especially for individuals with eye disabilities or weaker eyesight who require protective eye equipment such as spectacles.

iMon:

The iMon gaze tracking system presented in the research paper [4] is a novel appearance-based gaze tracking system designed for mobile devices. Here is a technical summary of the key features and contributions of the iMon system:

1. End-to-End Pipeline Approach: iMon considers the entire end-to-end pipeline involved in mobile appearance-based gaze tracking. It emphasizes that the overall performance of gaze tracking systems is impacted not only by the gaze estimation model but also by carefully engineering and fully exploiting all pipeline components.

2. Error Correction: The paper identifies three common sources of error within the gaze tracking pipeline and proposes solutions to correct them:
- Leveraging a novel 2D heatmap-based probabilistic representation of human gaze to mitigate errors in ground-truth labels caused by microsaccade eye movements.
- Improving the pre-processing pipeline by detecting eye regions, enhancing visual details, and removing motion blur in input images.
- Applying calibration to alleviate the effect of individual differences in Kappa angle, which can cause gaze tracking errors.

3. Orthogonal Improvements: The improvements introduced in iMon are orthogonal and complementary to advancements in model architecture or CNN model backbone. This means that future enhancements in individual components can further improve the accuracy of the gaze tracking system.

4. Availability: The source code of iMon and a video demonstrating its operation as part of a real application are available at a specified GitHub repository.

5. Applications: Gaze tracking is highlighted as a key input method for various applications in domains such as entertainment/games, personal productivity, human-computer interaction, medical diagnosis, and behavioral studies. The system's accuracy in determining where the user is focusing on the screen is crucial for user-driven context-sensitive applications.

Overall, the iMon system introduces novel techniques to improve the accuracy of appearance-based gaze tracking on mobile devices by addressing common sources of error in the gaze tracking pipeline. The system's comprehensive approach and orthogonal improvements pave the way for enhanced performance and potential applications in interactive, mobile, wearable, and ubiquitous technologies.

GazeEstimator

The GazeEstimator is an innovative architecture designed for accurate eye gaze estimation on mobile devices, developed by Liu Jigang and his colleagues at Nanyang Technological

University, Singapore [5]. This research, focusing on appearance-based methods, enhances human-computer interaction by accurately predicting gaze location in the presence of free-head movement, a challenge not fully addressed by previous methods.

The GazeEstimator framework utilizes a two-step training process employing convolutional neural networks (CNNs). The initial step involves training an eye landmarks localization network on the 300W-LP dataset. This network precisely localizes eye landmarks within an image, despite potential occlusions or variations in facial expressions. The subsequent step involves training a gaze estimation network on the GazeCapture dataset, leveraging eye images and eye grids as inputs to enhance robustness and accuracy.

A significant advantage of the GazeEstimator is its ability to function effectively even when the full face is not detectable, surpassing the state-of-the-art iTracker method in accuracy. This is particularly relevant for mobile devices where camera angles and user interaction can result in partial facial visibility.

Through rigorous experiments, the GazeEstimator demonstrated superior performance, with a validation error of 1.25cm, significantly lower than the iTracker's 2.44cm. Such precision in gaze estimation opens up new possibilities for applications in various fields, ranging from psychological research to the gaming industry, and represents a substantial advancement in the domain of eye-tracking technology.

References:

[1] https://gazecapture.csail.mit.edu/download.php

[2] https://www.researchgate.net/publication/343356401_Convolutional_Neural_Network-Based_Methods_for_Eye_Gaze_Estimation_A_Survey

[3] https://ieeexplore.ieee.org/document/9439113

[4] https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=7711&context=sis_research

[5] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8669057