Comparative Study and Analysis of Eye Gaze Estimation Models

Pavithra Moravaneni
Computer Software Engineering
Arizona State University
Tempe, United States
pmoravan@asu.edu

Kethan Sai Pavan Yeddla
Computer Software Engineering
Arizona State University
Tempe, United States
kyeddla@asu.edu

Loka Kalyan Balla
Computer Software Engineering
Arizona State University
Tempe, United States
lballa@asu.edu

Sowmya Veldandi
Computer Software Engineering
Arizona State University
Tempe, United States
sveldan1@asu.edu

Sai Teja Madha
Computer Software Engineering
Arizona State University
Tempe, United States
smadha1@asu.edu

Venkata Sai Pradeep Nagisetti
Computer Software Engineering
Arizona State University
Tempe, United States
vnagiset@asu.edu

***Abstract*—This paper presents a comprehensive comparative study of three leading deep learning-based eye gaze estimation models: iTracker, FAZE, and ODABE. Through a longitudinal dissection, each model's design, implementation, and performance metrics are thoroughly examined, utilizing prominent datasets such as GazeCapture and MPIIGaze. iTracker, leveraging convolutional neural networks (CNNs), addresses accuracy and cost challenges while demonstrating transferability across diverse datasets. FAZE introduces a unique rotation-aware encoder-decoder network, enabling personalized eye tracking with minimal calibration requirements. ODABE introduces online transfer learning, maintaining numerical stability in prediction. The analysis contributes to a nuanced understanding of the strengths and weaknesses of each model, aiding in overcoming challenges like accuracy limitations and adaptability constraints. This research enriches the field's knowledge base, offering valuable insights for researchers and practitioners seeking to leverage deep learning-based eye gaze estimation effectively.**

***Index Terms*—Eye tracking, Deep learning, Comparative study, Gaze estimation, Convolutional neural networks, Model evaluation**

I. Introduction

Eye tracking has emerged as a crucial technology in various domains including Human-Computer Interaction (HCI), medical diagnosis, and psychological studies. Over the years, several models have been developed to accurately track eye movements, each with its own strengths and limitations. In this paper, we introduce and compare three state-of-the-art models: iTracker, ODABE, and FAZE, each offering innovative solutions to enhance eye tracking performance.

iTracker is a convolutional neural network (CNN) model tailored specifically for eye tracking. Its standout feature lies in its superior accuracy compared to traditional appearance-based and shape-based models. Leveraging the extensive GazeCapture dataset, comprising over 2.5 million frames from diverse individuals, iTracker ensures robust generalization across different datasets while maintaining reliable performance. Notably, to address concerns regarding model size and complexity, dark knowledge integration has been employed to train a smaller, faster network suitable for real-time applications, with minimal loss in accuracy.

ODABE [Online Deep Appearance-Based Eye-Tracking]: ODABE tackles the issue of limited generalization of appearance-based eye tracking models across diverse user combinations, devices, and environments. Leveraging online transfer learning, ODABE initially undergoes pre-training on the MPIIGaze dataset before fine-tuning to adapt to new contexts. This approach significantly reduces prediction error demonstrating the efficacy of online learning in enhancing eye tracking performance across varied scenarios.

FAZE [Few-Shot Adaptive Gaze Estimation] is developed to find the accurate gaze estimation using very few calibration samples. Gaze estimation is the process of determining the person where he is looking, which is essential for many real-time applications such as HCI, Virtual Reality, automotive systems etc. Recent progress on gaze estimation evolving from classical model-based approach to appearance based using CNNs has significantly improved the accuracy and applicability of gaze estimation in the real-world. Despite the advancements, achieving high accuracy especially in personalized settings, insufficiency of person-independent gaze estimation, training person-specific models with thousands of training images per subject remains challenging.

In conclusion, iTracker, ODABE, and FAZE represent significant advancements in the field of eye tracking, each addressing distinct challenges and offering innovative solutions. While iTracker excels in accuracy and generalization, ODABE focuses on online transfer learning for improved adaptability, and FAZE prioritizes personalized gaze estimation with minimal calibration effort. Together, these models contribute to the evolution of eye tracking technology, promising enhanced performance and usability across various applications and environments.

II. Literature Review

Eye tracking technology has garnered significant interest and advancement in recent years, propelled by its applications across diverse fields including Human-Computer Interaction (HCI), psychology, medicine, and marketing research. This literature review provides an overview of the key developments and challenges in eye tracking methodologies, focusing on recent advancements in convolutional neural network (CNN) models for accurate gaze estimation and their applications.

Traditional Eye Tracking Techniques: Traditional eye tracking methods relied on specialized hardware such as infrared sensors and cameras to monitor eye movements. These systems often required controlled laboratory environments and cumbersome calibration procedures, limiting their practicality for real-world applications. Despite their accuracy, the high cost and complexity associated with traditional eye trackers hindered their widespread adoption.

Advancements in Deep Learning Models: The emergence of deep learning techniques, particularly convolutional neural networks (CNNs), revolutionized eye tracking by enabling accurate gaze estimation from raw image data. CNN-based models leverage large-scale datasets to learn complex patterns and features directly from images, eliminating the need for handcrafted features and facilitating real-time gaze prediction.

iTracker: One notable CNN model is iTracker, specifically designed for eye tracking with superior accuracy compared to traditional appearance-based and shape-based models. iTracker utilizes the extensive GazeCapture dataset, comprising millions of frames from diverse individuals, to achieve robust generalization across different datasets and environments. Despite its effectiveness, concerns regarding model size and computational complexity have prompted efforts to optimize iTracker for real-time performance on mobile devices.

ODABE: Another significant advancement is Online Deep Appearance-Based Eye-Tracking (ODABE), which addresses the challenge of generalization across diverse user populations, devices, and environmental conditions. ODABE leverages online transfer learning to adapt pre-trained models to new contexts, resulting in a substantial reduction in prediction error. By fine-tuning the model with new data, ODABE demonstrates the feasibility of real-time, adaptive gaze estimation in various scenarios.

FAZE: The FAZE framework aims to learn a latent embedding space for gaze estimation and addresses several challenges through a three-step approach.

Firstly, [8] FAZE analyzes the rotation-aware latent representation using a disentangling transforming encoder-decoder architecture. This process separates factors such as appearance, gaze direction, and head pose by explicitly decoding latent codes into images of the same person but with different gaze directions. This approach leverages the effectiveness of transforming encoder-decoder architectures in learning complex phenomena.

Secondly, FAZE employs meta-learning for few-shot personalization. Utilizing the latent features, the framework trains a highly adaptable gaze estimation model. This meta-learning approach enables the model to learn person-specific gaze estimators using only a few calibration samples. Each new subject is treated as a separate task, allowing for personalized gaze estimation with minimal data.

Finally, FAZE adapts the meta-learned network to new individuals with very few calibration samples, as low as three. This adaptation process further enhances the framework's ability to generalize and perform accurate gaze estimation across diverse individuals and scenarios.

In summary, recent advancements in CNN-based models have significantly enhanced the accuracy, adaptability, and practicality of eye tracking technology. Models such as iTracker, ODABE, and FAZE offer innovative solutions to overcome traditional limitations, paving the way for widespread adoption across various domains. Moving forward, continued research and development in deep learning methodologies promise to further improve the capabilities and accessibility of eye tracking technology for diverse applications.

III. DATASET

A. *MPIIGaze*

The MPIIGaze Dataset [2] is a widely used resource for studying human gaze estimation. It contains images of human faces captured under various conditions and corresponding gaze directions. The dataset consists of 213,659 images collected from 15 different subjects. Images were captured in a controlled environment with consistent lighting conditions using a high-resolution camera. The MPIIGaze Dataset is commonly used for training and evaluating gaze estimation algorithms, eye tracking systems, and related computer vision tasks. As a part of this research, MPIIGaze dataset is being used for ODABE model for the pre-training of the baseline model.

B. *GazeCapture*

The GazeCapture Dataset [1] is a valuable resource tailored for gaze estimation research in real-world settings, mainly focusing on mobile devices and unconstrained environments. The dataset consists of over 2.5 million images captured from over 1450 participants. The images were captured using mobile devices and corresponding gaze annotations, which offer insights into naturalistic head movements and gaze behavior. Participants used a custom mobile application to perform tasks while their gaze directions were recorded using the device's front-facing camera. The dataset's annotations include gaze directions, head poses, and metadata such as device orientation, facilitating the development of gaze estimation models robust to real-world conditions. The models in this paper will be evaluated using the GazeCapture Dataset.

C. *Downsampling of GazeCapture*

In data analysis and machine learning, the computational burden of large datasets can be significant. Downsampling, the process of reducing the size of a dataset, is often employed to mitigate this challenge. Downsampling involves reducing the size of a dataset by selecting a subset of its original data points. While various downsampling methods exist, systematic downsampling is a structured approach that ensures the preservation of essential data characteristics [5]. Systematic downsampling offers advantages over random downsampling, particularly in maintaining representativeness and improving computational efficiency. Systematic downsampling involves selecting data points from the dataset at regular intervals. This method ensures that the selected subset retains the key characteristics and patterns of the original data, making it more representative while reducing computational burden.

In our research, we utilized systematic sampling on the GazeCapture dataset to alleviate computational burden while ensuring that the downsampled dataset accurately represents the entire dataset. To achieve this, we selected 10% of the whole dataset using systematic sampling. This method involved starting from the first image of each participant and selecting every 10th image thereafter until we reached 10% of the dataset. By employing systematic sampling, we aimed to maintain the representativeness of the original dataset while reducing its size for computational efficiency. Throughout our research paper, whenever the GazeCapture dataset is referenced, it pertains to this downsampled dataset obtained through systematic sampling.

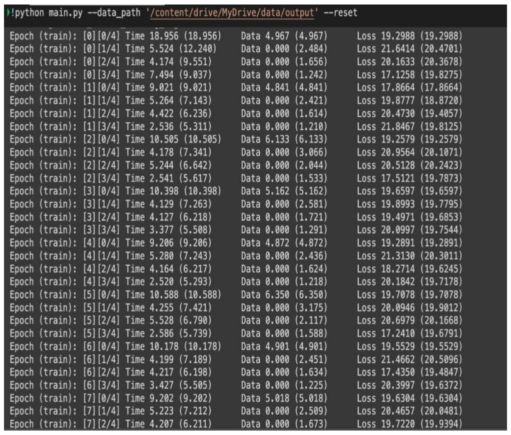
Systematic sampling ensures that the downsampled dataset maintains the essential characteristics and variability of the original dataset. By selecting every 10th image for each participant, we intended to capture a diverse range of gaze behaviors and scenarios present in the original dataset. This approach facilitates fair comparisons and analyses of various eye gaze tracking models, as the downsampled dataset retains the diversity and complexity of real-world gaze behavior. Additionally, systematic sampling provides a structured and reproducible method for downsampling, enhancing the reliability and consistency of our experimental results.

IV. Methodology

A. *iTracker Model*

1) *Preprocessing:*

- Metadata Initialization: Initializes metadata structure to store information about the dataset, including label record numbers, frame indices, dot coordinates, and face grid.
- Processing Each Recording: Iterates through each recording in the dataset, reading various JSON files containing information about face, eyes, dot coordinates, face grids, and frames.
- Preprocessing and Image Cropping: For each frame in each recording, it loads the corresponding image file, crops it based on the bounding boxes of the face and eyes, and saves the cropped images.
- Metadata Collection: Collects metadata information such as label record numbers, frame indices, dot coordinates, and face grid bounding boxes for each frame.
- Split Assignment: Classifies data into training and testing sets based on whether the input data originates from the train or test directory.
- Writing Metadata: Writes out the processed metadata to a `metadata.mat` file in the specified output path.



```
!python main.py --data_path '/content/drive/MyDrive/data/output' --reset
Epoch (train): [0][0/4] Time 18.956 (18.956)    Data 4.967 (4.967)      Loss 19.2988 (19.2988)
Epoch (train): [0][1/4] Time 5.524 (12.240)     Data 0.000 (2.484)      Loss 21.6414 (20.4701)
Epoch (train): [0][2/4] Time 4.174 (9.551)      Data 0.000 (1.656)      Loss 20.1633 (20.3678)
Epoch (train): [0][3/4] Time 7.494 (9.037)      Data 0.000 (1.242)      Loss 17.1258 (19.8275)
Epoch (train): [1][0/4] Time 9.021 (9.021)      Data 4.841 (4.841)      Loss 17.8664 (17.8664)
Epoch (train): [1][1/4] Time 5.264 (7.143)      Data 0.000 (2.421)      Loss 19.8777 (18.8720)
Epoch (train): [1][2/4] Time 4.422 (6.236)      Data 0.000 (1.614)      Loss 20.4730 (19.4057)
Epoch (train): [1][3/4] Time 2.536 (5.311)      Data 0.000 (1.210)      Loss 21.8467 (19.8125)
Epoch (train): [2][0/4] Time 10.505 (10.505)    Data 6.133 (6.133)      Loss 19.2579 (19.2579)
Epoch (train): [2][1/4] Time 4.178 (7.341)      Data 0.000 (3.066)      Loss 20.9564 (20.1071)
Epoch (train): [2][2/4] Time 5.244 (6.642)      Data 0.000 (2.044)      Loss 20.5128 (20.2423)
Epoch (train): [2][3/4] Time 2.541 (5.617)      Data 0.000 (1.533)      Loss 17.5121 (19.7873)
Epoch (train): [3][0/4] Time 10.398 (10.398)    Data 5.162 (5.162)      Loss 19.6597 (19.6597)
Epoch (train): [3][1/4] Time 4.129 (7.263)      Data 0.000 (2.581)      Loss 19.8993 (19.7795)
Epoch (train): [3][2/4] Time 4.127 (6.218)      Data 0.000 (1.721)      Loss 19.4971 (19.6853)
Epoch (train): [3][3/4] Time 3.377 (5.508)      Data 0.000 (1.291)      Loss 20.0997 (19.7544)
Epoch (train): [4][0/4] Time 9.206 (9.206)      Data 4.872 (4.872)      Loss 19.2891 (19.2891)
Epoch (train): [4][1/4] Time 5.280 (7.243)      Data 0.000 (2.436)      Loss 21.3130 (20.3011)
Epoch (train): [4][2/4] Time 4.164 (6.217)      Data 0.000 (1.624)      Loss 18.2714 (19.6245)
Epoch (train): [4][3/4] Time 2.520 (5.293)      Data 0.000 (1.218)      Loss 20.1842 (19.7178)
Epoch (train): [5][0/4] Time 10.588 (10.588)    Data 6.350 (6.350)      Loss 19.7078 (19.7078)
Epoch (train): [5][1/4] Time 4.255 (7.421)      Data 0.000 (3.175)      Loss 20.0946 (19.9012)
Epoch (train): [5][2/4] Time 5.528 (6.790)      Data 0.000 (2.117)      Loss 20.6979 (20.1668)
Epoch (train): [5][3/4] Time 2.586 (5.739)      Data 0.000 (1.588)      Loss 17.2410 (19.6791)
Epoch (train): [6][0/4] Time 10.178 (10.178)    Data 4.901 (4.901)      Loss 19.5529 (19.5529)
Epoch (train): [6][1/4] Time 4.199 (7.189)      Data 0.000 (2.451)      Loss 21.4662 (20.5096)
Epoch (train): [6][2/4] Time 4.217 (6.198)      Data 0.000 (1.634)      Loss 17.4350 (19.4847)
Epoch (train): [6][3/4] Time 3.427 (5.505)      Data 0.000 (1.225)      Loss 20.3997 (19.6372)
Epoch (train): [7][0/4] Time 9.202 (9.202)      Data 5.018 (5.018)      Loss 19.6304 (19.6304)
Epoch (train): [7][1/4] Time 5.223 (7.212)      Data 0.000 (2.509)      Loss 20.4657 (20.0481)
Epoch (train): [7][2/4] Time 4.207 (6.211)      Data 0.000 (1.673)      Loss 19.7220 (19.9394)
```

Fig. 1. iTracker Testing

2) *Model Training:*

- Epoch-wise Iteration: The training routine operates over a series of epochs, iterating through the dataset multiple times to enable the model to learn from the entire data distribution.
- Training Mode Activation: Prior to commencing the training loop, the model is set to training mode, a crucial step for activating functionalities like dropout and batch normalization layers.
- Data Loading and Preprocessing: Training data is loaded and preprocessed in each iteration, including data augmentation techniques like random crops, flips, and color jittering to enhance model generalization.
- Forward Pass and Loss Computation: A forward pass of the model on the input data computes predictions for gaze coordinates, followed by the calculation of the Mean Squared Error (MSE) loss between predicted and ground truth gaze coordinates.
- Backpropagation and Gradient Descent: Gradients of the loss function with respect to the model parameters are computed through backpropagation, enabling efficient optimization via stochastic gradient descent (SGD).
- Parameter Update: Model parameters are updated using the computed gradients, with the learning rate adjusted dynamically to guide the optimization process towards convergence.
- Performance Monitoring and Reporting: Performance metrics such as batch processing time, data loading time,

and loss values are continuously monitored and reported to gauge training progress and model efficacy.

- Epoch Progress Tracking: Progress information including epoch number, batch number, and relevant metrics are systematically logged and displayed, providing insights into the training dynamics.
- Iterative Optimization: The iterative nature of the training process enables the model to progressively learn complex patterns and representations from the data, ultimately leading to improved performance.

B. FAZE Model

1) Preprocessing: This preprocessing aims to estimate the gaze direction from facial images. By following the below preprocessing steps, the input data is transformed into a suitable format for training the FAZE model, enabling it to effectively learn the mapping between facial appearance and gaze direction.

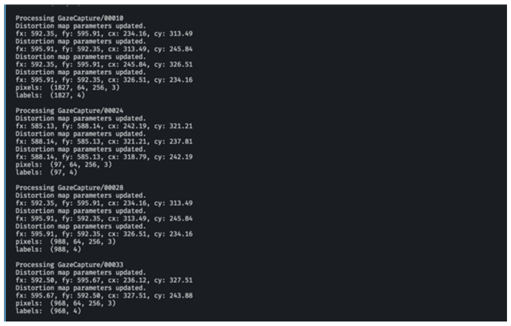
- Image Undistortion: Facial images captured by camera may suffer from distortion caused by camera lens, undistorting these would make sure to get accurate gaze estimation. This step involves correcting lens distortions using camera calibration parameters such as distortion coefficients.
- Head Pose Correction: This involves determining the orientation of the head in the image and adjusting to standardized pose.
- Normalization of Gaze direction: This involves transforming gaze direction vectors to common coordinate space, such as aligning them with head orientation or converting to pitch and yaw angles.
- Image patch extraction: This extract around the eyes to focus on the facial features for gaze estimation.
- Normalization: Finally, the preprocessed images and gaze direction labels are normalized to have zero means and unit variance.

2) Training: [8] FAZE model is trained using both the above techniques, leveraging configuration, training strategies to achieve effective gaze estimation performance.

- **DT-ED**: This is trained with batch size of 1536 images and are processed in each training iteration. FAZE utilizes mixed-precision training with NVIDIA's Apex library. It helps to speed up the training and reduce memory usage. The DT-ED is trained for 50 epochs where it allows model to learn from data iteratively. Smaller learning rates ensures smoother convergence and weight regularization helps prevent over fitting.
- **GazeMLP training**: This technique where the model learns how to learn from few samples. the outer learning rate is used to update the parameters of the model based on meta-learning objective and inner loop is to update based on small number of calibration samples.

C. ODABE Model

1) Dataset Preprocessing: For the ODABE model, we utilized the MPIIGaze dataset for pre-training and the down-sampled GazeCapture dataset for online learning. To prepare the GazeCapture dataset, we focused on specific metadata files, namely dotInfo.json and Screen.json, to extract dot and screen information related to each image. These files provided essential data for our gaze estimation. We adjusted and normalized the coordinates based on the orientation of the device's screen, considering both portrait and landscape orientations to ensure accuracy.



```
Processing GazeCapture/00010
Distortion map parameters updated.
fx: 592.35, fy: 595.91, cx: 234.16, cy: 313.49
Distortion map parameters updated.
fx: 595.91, fy: 592.35, cx: 313.49, cy: 245.84
Distortion map parameters updated.
fx: 592.35, fy: 595.91, cx: 245.84, cy: 326.51
Distortion map parameters updated.
fx: 595.91, fy: 592.35, cx: 326.51, cy: 234.16
pixels:  (1827, 64, 256, 3)
labels:  (1827, 4)

Processing GazeCapture/00024
Distortion map parameters updated.
fx: 585.13, fy: 588.14, cx: 242.19, cy: 321.21
Distortion map parameters updated.
fx: 588.14, fy: 585.13, cx: 321.21, cy: 237.81
Distortion map parameters updated.
fx: 588.14, fy: 585.13, cx: 318.79, cy: 242.19
pixels:  (97, 64, 256, 3)
labels:  (97, 4)

Processing GazeCapture/00028
Distortion map parameters updated.
fx: 592.35, fy: 595.91, cx: 234.16, cy: 313.49
Distortion map parameters updated.
fx: 595.91, fy: 592.35, cx: 313.49, cy: 245.84
Distortion map parameters updated.
fx: 595.91, fy: 592.35, cx: 326.51, cy: 234.16
pixels:  (988, 64, 256, 3)
labels:  (988, 4)

Processing GazeCapture/00033
Distortion map parameters updated.
fx: 592.50, fy: 595.67, cx: 236.12, cy: 327.51
Distortion map parameters updated.
fx: 595.67, fy: 592.50, cx: 327.51, cy: 243.88
pixels:  (968, 64, 256, 3)
labels:  (968, 4)
```

Fig. 2. FAZE model output depicting the gaze directions

During preprocessing, each image underwent several steps. Firstly, we utilized the Dlib library [6] in Python to detect facial landmarks and crop the image to focus on the face area, isolating relevant features for gaze estimation. Subsequently, we normalized the pixel values of the cropped image to the range [0,1] by dividing each pixel value by 255.

Furthermore, we conducted channel-wise normalization to enhance data quality. This involved calculating the mean and standard deviation for each color channel (red, green, blue) across all images in the dataset. For each image, we subtracted the mean value of each color channel from the corresponding pixel value and divided it by the standard deviation of that channel. This normalization ensured the data was appropriately centered and scaled, facilitating better convergence during model training.

The final processed image served as input for training the ODABE model. This preprocessing pipeline ensured that our model received high-quality input data.

2) Model Training:

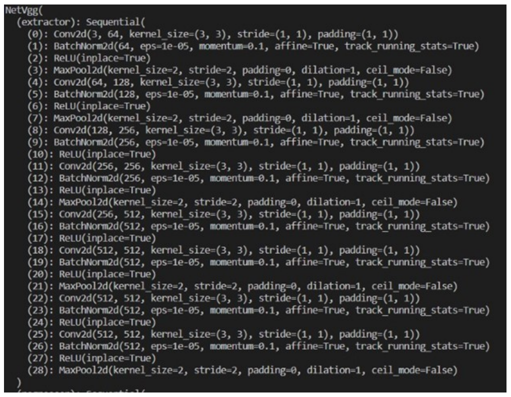
a) Pre-training: The baseline model is pre-trained using the MPIIGaze dataset to establish a foundational understanding of gaze patterns. The entire MPIIGaze dataset is employed to train the model, utilizing the VGG11 architecture with added batch normalization. Pre-existing weights from ImageNet [7] are used to initialize the network's parameters. The dataset is split into train, validation, and test sets with proportions of 70%, 15%, and 15%, respectively.

The extractor consists of a series of convolutional layers followed by batch normalization and ReLU activation functions. This component processes the input image and learns hierarchical representations of the input data. On the other hand, the regressor comprises fully connected (linear) layers followed by ReLU activation functions and a final sigmoid activation function. It takes the features extracted by the extractor and maps them, performing regression to predict two output values.

In training neural networks with mini-batches, each mini-batches statistic (mean and variance) may significantly differ from those of the entire dataset, leading to Internal Covariate Shift (ICS). This refers to the change in the distribution of network activations due to changes in model parameters during training, which can slow down the training process and hinder convergence.

To address this issue, Batch Normalization normalizes the activations of each layer within a mini-batch by subtracting the mean and dividing by the standard deviation, effectively centering and scaling the activations. This helps stabilize the training process by ensuring consistent distribution of activations across different mini-batches.

Pre-training is conducted for four epochs with a batch size of 128. Stochastic Gradient Descent is used with a learning rate of 0.001 and momentum of 0.9 [4]. The same learning rate is employed during online learning. Based on the observed performance on the validation set, the hyperparameters for the network were chosen. Once an optimal model is converged, testing was done using the test split on this converged model.



```
NetVgg(
  (extractor): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace=True)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): ReLU(inplace=True)
    (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

Fig. 3. ODABE model depicting the extractor used

b) Fine-tuning: We have utilized two different approaches or phases of fine-tuning the model on the GazeCapture dataset. Offline: In this phase, the pre-trained model is fine-tuned using the GazeCapture dataset without updating the weights of the earlier layers (often referred to as freezing). Only the weights of the later layers, typically closer to the output layer, are adjusted to adapt the model to the new dataset. This approach is called ”offline” because it’s typically done before deploying the model for real-time inference. Online: In this phase, the model is fine-tuned on the GazeCapture dataset with the flexibility to update the weights of all layers, including the earlier ones. Unlike the offline stage, here, the weights of all layers are adjusted based on the new dataset. This approach is called ”online” because it’s often done during or after deployment, allowing the model to adapt further to new data encountered during inference.

Given the pre-trained model, the fine-tuning process aims to fine-tune certain layers of the same network used for pre-training. This adaptation is intended to make the network more suitable for a new context of images to which the model is being introduced.

During fine-tuning, the training procedure follows a similar approach as pre-training. Images from GazeCapture dataset are normalized using statistics from MPIIGaze, as the model is only being fine-tuned. Unlike pre-training, there is no feedback of overfitting during online learning, such as the validation and test split. Instead, each new image is passed through the network, and its loss is computed. If the loss exceeds a predefined threshold, the model is updated by backpropagating for that specific image. Otherwise, the image is discarded, and the model remains unchanged. The threshold used is the final mean loss for the validation set obtained during pre-training for MPIIGaze model. This heuristic is based on the observation that the training/validation loss during pre-training can serve as an indicator of the network’s modeling capabilities.

D. Enhancements

- By dynamically adjusting the learning rate, we aim to achieve faster convergence, potentially reducing the number of epochs required for training.
- In the case of the ODABE model [4], fine-tuning was performed using a custom dataset comprising only 1000 images. However, in our study, we utilized the downsampled GazeCapture dataset, which encompasses data from 1476 participants and includes approximately 200,000 images. This indicates a substantial expansion in the scope of our experimentation compared to the original ODABE model.
- Given the size of our dataset, we implemented strategies such as early stopping and model checkpointing to prevent overfitting. These measures ensure that the model converges effectively and efficiently, even when exposed to the entirety of the data.

V. Results

A. RMSE as a metric

Root Mean Square Error (RMSE) is a widely employed metric in machine learning, particularly in regression tasks, to quantify the accuracy of predictive models by measuring the difference between predicted and actual values. RMSE calculates the square root of the average of squared differences between predicted and observed values, providing a comprehensive assessment of model performance. Its intuitive interpretation, where lower values indicate better model accuracy, makes it a preferred choice for evaluating regression models across diverse domains. The Root Mean Square Error (RMSE) formula is given by:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad (1)$$

In this formula:

- RMSE represents the Root Mean Square Error,

- n denotes the number of data points,
- y_i represents the actual value for the i^{th} data point,
- $\hat{y}_i$ represents the predicted value for the i^{th} data point.

We have employed RMSE as the evaluation metric to calculate loss for the purpose of our research to evaluate the models.

B. iTracker

1) Epoch-wise Loss:

- Loss values are reported for individual epochs, summarizing the overall training progress at each epoch.
- Trends in loss values across epochs offer insights into the model's optimization trajectory and convergence patterns.

2) Training Loss Analysis:

- Training loss trends are evaluated to assess the model's ability to minimize prediction errors on the training data.
- Fluctuations or trends in training loss values indicate the effectiveness of the model optimization process and its adaptation to the training dataset.

3) Model Optimization:

- Loss values are utilized as optimization criteria to update model parameters and improve predictive performance.
- Optimization strategies, such as gradient descent, are employed to minimize loss values iteratively during training.

4) Convergence Evaluation:

- Loss trends are analyzed to determine whether the model has converged to an optimal solution.
- Convergence is inferred based on the stabilization of loss values over successive epochs, indicating that the model has sufficiently learned the underlying patterns in the training data.

5) Performance Assessment:

- Training loss trends serve as key indicators of model performance and optimization progress, guiding decisions regarding model refinement and hyperparameter tuning.
- Performance gains or deficiencies are assessed based on changes in loss values over epochs, facilitating informed adjustments to improve overall model performance.

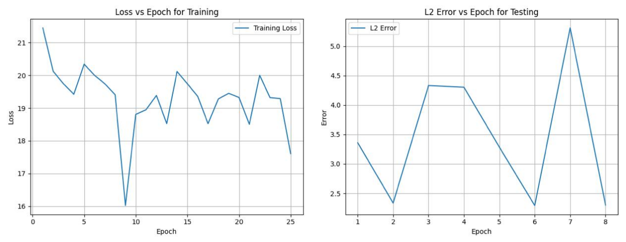


Fig. 4. iTracker model graphs

In the combined graph fig 4, the left subplot represents the "Loss vs Epoch for Training", where the training loss decreases over epochs, indicating that the model is learning and improving its performance on the training data. The fluctuation in the loss values across epochs suggests some variability in the training process, which could be due to factors like learning rate, model architecture, or data characteristics.

On the right subplot, we have the "Loss vs Epoch for Testing", which is labeled as "L2 Error" but plotted as loss. Here, the loss values for the testing data show some variation across epochs, with a general trend of increasing at certain points and decreasing at others. This behavior could indicate fluctuations in model performance on unseen data over epochs. The testing loss values are generally higher than the training loss, which is expected as the model may not generalize perfectly to unseen data.

Overall, the combined graph provides insight into both the training and testing performance of the model over epochs, highlighting the training process's progression and the model's behavior on unseen data.

C. FAZE

1) EPOCH-WISE Loss:

- Both training and test losses decreased across epochs, indicating improved model performance and convergence.

2) Training Loss Analysis:

- Training loss trends showed consistent decreases across epochs for GazeCapture dataset, indicating that the model effectively minimized prediction errors on the training data.

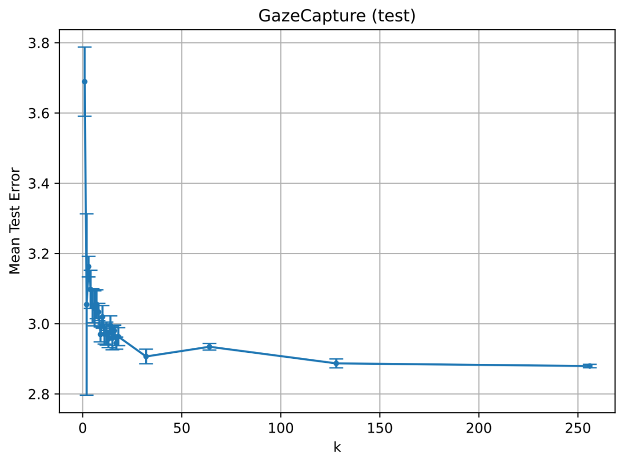


Fig. 5. FAZE model graph

Fig 5 depicts the evaluation of a predictive model using the GazeCapture dataset, as reflected by the Mean Test Error across varying values of a parameter 'k'. The plot illustrates a significant decrease in error with initial increments of 'k', followed by a plateau, indicating an optimal 'k' range for minimal error. Error bars suggest diminishing variability with increased 'k', underscoring enhanced model stability.

In fig 5 ’k’ represents the calibration samples refer to the number of labeled data points or instances available for each task during the testing phase. The FAZE model utilizes the calibration samples (if k=2 samples per task) to adapt its parameters or update its internal representations to better suit the characteristics of the new task.

3) *Performance Assessment:*

- This represents the performance metrics of the model evaluated on GazeCapture datasets and for different values of K, which is defined as number of calibration samples used for personalised gaze estimation.
- Testing has been done for different values of k. Output shows the test errors for each value of K. This represents the average performance of the model across multiple trials or evaluations.
- It also shows the standard deviation of the test errors for each value of k which indicates the variability or consistency of models performance.
- A lower train loss indicates better performance on the training set
- A higher standard deviation indicates greater variability in the test loss values, which could inconsistency in the model’s performance across different test datasets or runs.
- By specifying different values of k, you can control how quickly the model adapts to new tasks. Smaller values of k may lead to faster adaptation but may also result in less accurate models, while larger values of, k may lead to slower adaptation but potentially more accurate models.
- Therefore, selecting an appropriate value of k involves a trade-off between adaptation speed and model accuracy, and it depends on factors such as the complexity of the tasks, the availability of training data, and the desired balance between adaptation speed and performance accuracy.

D. *ODABE*

1) *Training Losses:*

- For the initial training phase on the MPII dataset, the training loss decreases initially from epoch 1 to epoch 2 but then increases slightly by epoch 3 and further by epoch 4.
- This indicates that the model might be overfitting or encountering difficulties in generalizing to the validation dataset. The final training loss after the MPII dataset training phase is approximately 0.257.

2) *Validation Losses:*

- The validation loss on the MPII dataset initially decreases from epoch 1 to epoch 2 but then increases notably by epoch 3 before slightly decreasing again by epoch 4.
- This behavior might indicate overfitting or instability in the model training process. The final validation loss after the MPII dataset training phase is approximately 0.228.

3) *Test Loss:*

- The test loss after the MPII dataset training phase is approximately 0.262.

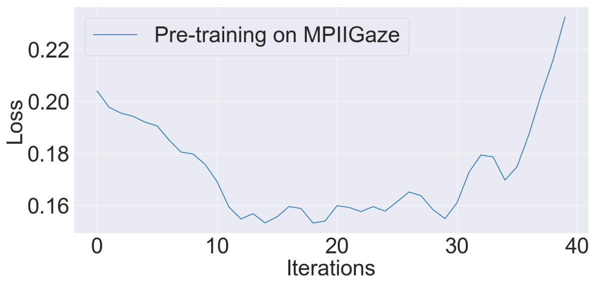


Fig. 6. Pre-training of ODABE model using MPIIGaze

The graph 6 depicts the training loss vs number of iterations for the Pre-training phase done using the MPIIGaze dataset.

4) *Fine-tuning on GazeCapture Dataset:*

- After the initial training on the MPII dataset, the model undergoes fine-tuning on the GazeCapture dataset. During the fine-tuning process, both the training and validation losses show a decreasing trend.
- The training loss decreases from approximately 0.434 to 0.359, while the validation loss decreases from approximately 0.435 to 0.353.
- The fine-tuning process on the GazeCapture dataset takes a significant amount of time, approximately 6311.74 seconds (or about 105 minutes).

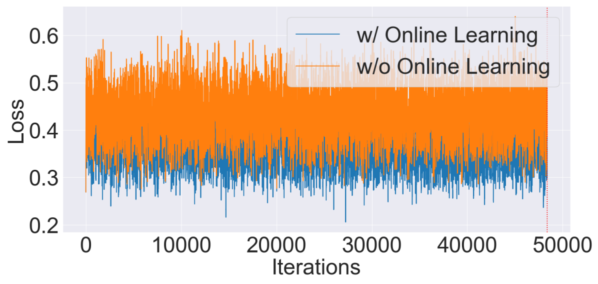


Fig. 7. Fine tuning of ODABE model using GazeCapture

TABLE I
ODABE COMPARISON

stages	Env 1 [4]	Env 2 [4]	proposed work
w/o Online learning (valid)	0.330	0.266	0.228
with Online learning (train)	0.150	0.188	0.359
with Online learning (valid)	0.150	0.140	0.353

The table I, depicts the comparison between our proposed ODABE model implemented using MPIIGaze dataset and GazeCapture dataset. Where as the original ODABE model was implemented using MPIIGaze dataset and custom dataset in two different environments. It can be inferred that with online learning, the average loss reported for custom dataset is less than the loss achieved for GazeCapture dataset. This can be associated to custom dataset which only used 1000 images and the model may have learnt the associated fluctuations and noise from the data. However, for our proposed ODABE model the training and validation loss are not drastically different, hence it is evident that the our proposed model generalizes well to the new data.

5) Observations:

- The model seems to initially overfit to the MPII dataset, as evidenced by the increasing validation loss after epoch 2.
- However, the fine-tuning process on the GazeCapture dataset seems to improve the generalization performance of the model, as both training and validation losses decrease during this phase.
- The test loss after fine-tuning on the GazeCapture dataset is not provided, so it's unclear how the model performs on unseen data after this process.

Overall, it seems that while the model struggles with overfitting during initial training on the MPII dataset, fine-tuning on the GazeCapture dataset helps improve its performance.

E. Comparative analysis

For the purpose of this research, we have conducted comparison and analysis of the three models using training time and percentage loss. For the ODABE model, the screen size is normalized so the maximum loss that can be achieved is $\sqrt{2}$. The other models, iTracker and FAZE use L2 error to calculate the loss. To compare the models uniformly, we have used percentage loss of the models.

The training loss percentage and the testing loss percentage for iTracker model is 13.24 % and 4.35%.

The training loss percentage and the testing loss percentage for FAZE model is 20.8 % and 3.055%.

The training loss percentage and the valid loss percentage for ODABE model is 17.29 % and 18.85%.

Observations from the above loss percentage data of various models.

- Overfitting: Training loss percentages are consistently higher than testing/validation loss percentages across all models.
- Indicates potential overfitting where models may be memorizing the training data instead of generalizing well to unseen data.
- Model Performance: Lower testing/validation loss percentages indicate better generalization performance.
- The FAZE model has the lowest testing loss percentage (3.055%), suggesting superior generalization compared to other models.
- Comparative Performance: FAZE model outperforms the other models in terms of testing loss percentage.
- iTracker model exhibits the highest testing loss percentage (4.35%), indicating relatively poorer generalization compared to the other models.

TABLE II
COMPARISON OF MODELS USING TRAINING TIME

Model	Training time
iTracker	54.32 min
FAZE	72.9 min
ODABE	105.19 min

From the table II, it is evident that the iTracker model exhibits the shortest training time among the three models. This phenomenon can be attributed to the computationally intensive preprocessing required by the iTracker model. In this preprocessing step, the entire dataset undergoes substantial reduction to less than half of its original size. Consequently, during training, only the preprocessed dataset is utilized, resulting in significantly reduced computational demands for training.

VI. Challenges Faced

A. ***Hardware Requirements***

- Deep learning models, which involves large-scale training processes that requires relevant computational resources.
- NVIDIA Apex is a library that provides tools for mixed-precision training, which can significantly speed up training on NVIDIA GPUs.
- However, it was challenge to configure NVIDIA Apex in local machines. To solve this we have used Google colab which have worked as expected.

B. ***Preprocessing the dataset [GazeCapture and MPIIDataset]***

- As we were using Gogle colab to run the entire datatset initially, it took lot more time for us to figure out how to train and test the entire dataset.
- We have come up with a solution to divide the whole dataset into 10% each and train and test all the batches.

C. ***Dependency issues while configuring the models***

- These deep learning frameworks have numerous dependencies, and configuring them with the versions was challenging.
- This issues arise due to conflicts between different versions, compatability issues with the software or difference in operating systems. however, using Google collab to train and test the models gave us a chance to run the models as expected.

D. ***Limited resources to understand the models***

- Deep learning models, especially state-of-the-art architectures like iTracker, FAZE, ODABE, and GAZEL, can be complex and challenging to understand fully. These models often involve intricate network architectures, optimization algorithms, and training methodologies.
- Limited documentation, lack of tutorials, or specialized terminology may further hinder understanding, made it difficult to implement the models at initial stage.

VII. Conclusion

The implementation details provided underscore the systematic approach taken to prepare datasets, preprocess images, and train deep learning models for eye gaze estimation. While iTracker and FAZE focus on general model training, ODABE introduces fine-tuning mechanisms to adapt to new contexts, emphasizing adaptability and robustness in real-world scenarios. Each model's architecture and training methodologies are tailored to address specific challenges in eye tracking, contributing to advancements in HCI and related fields.

Future plans in eye gaze estimation include refining model architectures, integrating additional modalities, optimizing real-time performance, enhancing adaptability, exploring personalized methods, expanding cross-domain applications, addressing ethical considerations, and promoting standardized benchmarks.

References

[1] K. Krafka et al., "Eye Tracking for Everyone," www.cv-foundation.org, 2016.

[2] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, "MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 1, pp. 162–175, Jan. 2019, doi: https://doi.org/10.1109/tpami.2017.2778103.

[3] S. Park, S. D. Mello, P. Molchanov, U. Iqbal, O. Hilliges, and J. Kautz, "Few-Shot Adaptive Gaze Estimation," openaccess.thecvf.com, 2019.

[4] Bruno Klein Salvalaio and Gabriel, "Self-Adaptive Appearance-Based Eye-Tracking with Online Transfer Learning," Oct. 2019, doi: https://doi.org/10.1109/bracis.2019.00074.

[5] S. A. Mostafa and I. A. Ahmad, "Recent developments in systematic sampling: A review," Journal of Statistical Theory and Practice, vol. 12, no. 2, pp. 290–310, Aug. 2017, doi: https://doi.org/10.1080/15598608.2017.1353456.

[6] D. E. King, "Dlib-ml: A machine learning toolkit," Journal of Machine Learning Research, vol. 10, pp. 1755–1758, 2009.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in CVPR09, 2009.

[8] Seonwook Park and Shalini De Mello and Pavlo Molchanov and Umar Iqbal and Otmar Hilliges and Jan Kautz "Few-Shot Adaptive Gaze Estimation," "International Conference on Computer Vision (ICCV)"

[9] Youtube videos for Few-Shot Adaptive Gaze Estimation by NVIDIA.