Dynamic Eye Gaze Tracking for Foveated Rendering and Retinal Blur

Kushagr Gupta, Suleman Kazi, Terry Kong

***Abstract*— This project deals with tracking eye gaze location and predicting future eye movement sequences using classical machine learning techniques and Hidden Markov Models, respectively, for users wearing head mounted displays (HMD). Eye gaze location can enhance the virtual reality experience inside an HMD. In this report, we discuss how we collected a dataset for eye gaze tracking and the features we extracted. Then, we evaluated three learning approaches to estimate eye gaze and evaluate the performance using regularized linear regression, support vector regression (SVR), and neural networks. We also briefly discuss our approach to the prediction problem.**

I. INTRODUCTION

The ability to track and predict eye gaze accurately can have a huge impact in Virtual Reality (VR) headsets since it gives content developers the flexibility to enable foveated rendering or simulate retinal blur to enhance content and improve interactivity using eye-gaze information. A recent HMD project with integrated eye tracking [6] raised well in excess of its funding goal via kickstarter. Foveated rendering is a technique in which the image resolution is not uniform across the image. Retinal blur refers to blurred perception of objects outside the center of gaze (in the peripheral vision). In order to apply these techniques the eye gaze location must be accurately known. To provide enough time for rendering and to account for latency the eye gaze location in the next few frames must also be predicted. This requires high accuracy so that the scene is rendered and blurred at the right locations. We use machine learning techniques to output an estimated eye gaze location in terms of screen x-y coordinates.

This is a joint project for CS229 and CS221. The work shown in this report explains the eye gaze tracking portion of the project which was done for CS229. The approach to predict the eye gaze location constitutes the CS221 portion of the project and is very briefly summarized at the end to connect the two projects. For a much more in depth analysis of the prediction approach, we refer the reader to our report for CS221.

II. RELATED WORK

A large number of the modern eye tracking algorithms we came across [1]–[4] use signal/image processing in conjunction with geometry to accomplish eye gaze tracking. In fact [5] which is a survey of models for eye gaze tracking does not list any machine learning methods. [8] uses Artificial Neural Networks for eye gaze tracking and is able to achieve an accuracy of 1.5°. [9] also uses a probabilistic incremental learning algorithm to estimate eye gaze but instead of looking at images of the user's eyes to do so it relies on pre-computed saliency maps of the videos (showing the prior probabilities of where the user is likely to look at) the user views to estimate eye gaze instead and is able to achieve an accuracy of $<3.0°$. State of the art eye-tracking hardware [7] is advertised as being able to achieve a typical accuracy of accuracy of 0.5°. In our work, instead of relying on signal or image processing algorithms we use machine learning techniques and instead of relying on one, we also compare different learning algorithms and quantify their results as described in the sections below.

III. DATASET AND FEATURES

A. Dataset Acquisition

The dataset consists of images of the user's eye looking at points displayed on a screen with a resolution of 1920x1080 pixels and a size of 16.5x30.5 cm. These points serve as the ground truth data. The user sits 30cm away from the screen, wearing a pair of glasses with a webcam mounted on them 5cm from the eye and off-axis as to not obstruct the screen. By using this method we did not have to worry about inadequate illumination inside a VR headset whilst also getting images similar to what a camera inside a VR headset would produce. We also looked up online datasets but they did not fit our desired input data requirements. For example the dataset in [16] gives the eye gaze location (x,y) values on the screen but doesnt give images of the eye. The dataset in [17] gives images of eye but with camera right in front of the eye thus obstructing content on screen. Our data collection setup is shown in Fig. 1.

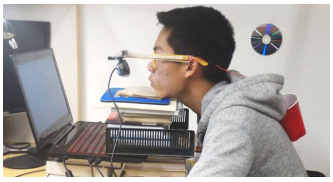

Fig. 1. Setup For Data Collection

1) As a one time procedure user is asked to look straight at the camera and a reference image is captured. A box is manually drawn around the position of the user's eye and the image is cropped to this box and saved.
2) Proceeding to collect a dataset user must take another reference image. The master reference obtained in step 1 is cross correlated with this reference image to find the location of the user's eye and a box of the same size as the master reference is drawn around the eye automatically. This step removes slight variations of user head position between different dataset acquisitions.

3) The user is then shown a random point on the screen and each key press captures an image and displays a new point on the screen. Each captured image is cropped based on the bounding box calculated in step 2 and saved along with the location of the point shown on screen.

B. Feature Selection

Initially all the images were converted to grayscale and vectorized and used as features but it resulted in too many features compared to the training set size and thus each image was down sampled and vectorized which serve as features for the algorithm. Different down sampling factors for resizing the images were considered and it was observed that the performance gain after a certain feature length was marginal and the training time became too prohibitive. We thus chose a downsampling factor that performed reasonably well giving a final feature length of 300 as compared to the original image size of 300000 pixels as shown in Fig. 2.

We introduced two additional sets of features namely the inner product with the eigen-eyes and the inner product with the fisher-eyes [19] which help as they increase variance using Principal Component Analysis. They are explained as follows.

1) Eigen Eyes : Given the vectorized training images the mean of the entire set μ is removed and output is concatenated into a matrix, say S, of dimension number of pixels by number of training images. Eigen eyes are essentially the eigen-vectors of the scatter matrix $S*S'$. This gives us the directions in which there is higher variance based on the eigen values with an illustration in 2.
2) Fisher Eyes : In eigen eyes we consider the entire dataset and discriminate based on it, we can improve this using fisher analysis. The aim here is to maximize between class scatter while minimizing within class scatter. For the purpose of eye gaze tracking, we considered a finite set of classes based on the true locations of target point where the eyes are looking at and each image (mean removed) is binned into one of these classes. The generalized eigen vectors of the problem are called fisher eyes with an illustration in 2. While the initial computation of all the eigen-eyes and fisher-eyes may take some time, they will only need to be computed once at the beginning using all the training data.

Fig. 2. Raw Image, Downsampled Image, Eigen Eye, Fisher Eye

IV. METHODS: TRACKING APPROACH

To evaluate our tracking approach, we compared three different learning methods: regularized linear regression, SVR, and neural networks. In all three cases we treat the x and y dimensions of the eye gaze location independently. While we could consider the dimensions together, there is no added value. As the results will show, using least squares where the labels are multidimensional (x and y) actually performs worse which justifies our assumption.

A. LEAST SQUARES WITH REGULARIZATION

Least squares, or linear regression, was used as a baseline due to its simplicity and straightforward implementation. Least squares assumes that the data can be modeled as

$$p = Xw$$

where p is the eye gaze location (x or y), X is the design matrix and w is the weight vector. Since least squares can lead to over-fitting, we also considered regularized least squares, which has an extra term augmented to the least squares objective. With $\mu = 0$ it reduces to linear regression without regularization.

$$J_{LSR}(w) = \|p - Xw\|_2^2 + \mu\|w\|_2^2$$

The hyper-parameter μ controls how much over-fitting we will tolerate. This modified objective function is equivalent to assuming a Gaussian prior on the weights with zero mean and non-zero variance.

Taking gradients yields the closed form solution:[1]

$$w_{LSR} = (X^T X + \mu I)^{-1} X^T p$$

where I is the n by n identity matrix and n is the dimension of a feature vector.

Both of these results assume that X is full rank and skinny. While this isn't always the case, we can always perform SVD or QR factorization to remove the redundant columns in the X. Another approach is to pass the optimization problem into a convex solver, which will likely perform Newton's method which doesn't require any priors on the data.

B. SUPPORT VECTOR REGRESSION

Another approach is to use SVR. The problem inherits part of its name from the popular support vector machine problem. Like most regression problems, the hypothesis function is of the form $h(x) = w^T x + b$ where w is the weight vector and b is the constant offset.

To use this model, one must select a tolerance, ϵ, and a hyper-parameter for the slack variables, C, and then the weights are learned to minimize the objective. The physical meaning of ϵ is the amount of deviation that is tolerated when evaluating with the hypothesis function.

This problem can be solved more efficiently in the dual domain and is explained more thoroughly in [14]. The corresponding dual problem is

$$\begin{aligned} &\text{maximize} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) < x_i, x_j > \\ -\epsilon\sum_{i=1}^{m}(\alpha_i - \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) \end{cases} \\ &\text{subject to} \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \tag{1}$$

[1]This also assumes that the design matrix is skinny and full rank.

where $x^{(i)}$ is a training feature vector, $y^{(i)}$, m is the number of training examples, α_i and α_i^* are the dual variables. One nuance to the dual problem is that there is no longer a mention of the constant term b. Thus, it needs to be inferred from the solution of the dual problem. For the purpose of this report, we have set[2]

$$b = \frac{b_{max} + b_{min}}{2}$$

where $b_{max} = \max\{-\epsilon + y^{(i)} - < w, x^{(i)} | a_i < C \text{ or } a_i^* > 0\}$, $b_{min} = \min\{-\epsilon + y^{(i)} - < w, x^{(i)} | a_i > 0 \text{ or } a_i^* < C\}$, and $< \cdot, \cdot >$ represents a kernel.

Using the solution of the dual problem (α_i and α_i^*), we can construct the weight vectors as follows:

$$w = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) < x^{(i)}, x^{(i)} >$$

Like SVM, SVR can also be kernelized.

C. NEURAL NETWORKS

Neural Networks consist neurons which take in inputs (weighted by a weight vector which may be different for each neuron) and process them using a transfer function to give an output. A neural network consists of a number of interconnected neurons. We use a feed-forward neural network in which the information flow is unidirectional. If we have a neuron j with a sigmoid transfer function, a weight vector $w_j = [w_{1j}....w_{nj}]^T$, bias term b and inputs o_k, from the previous neurons then the output for neuron j will be:

$$o_j = \phi(z) = \phi(net_j) = \frac{1}{1 + exp(-z)}$$

Where:

$$z = \sum_{k=1}^{n} w_{kj} o_k$$

The network is trained using a back-propagation algorithm which uses gradient descent to optimize the weights. The squared error in this case is:

$$E = \frac{1}{2}(t - y)^2$$

Where t is the desired output for an input vector and y is the actual output.From [18] we see that if we take the partial derivatives using the chain rule we get:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

Where:

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \begin{cases} (o_j - t_j)\phi(net_j)(1 - \phi(net_j)) \\ \text{if } j \text{ is an output neuron} \\ \sum_{l \in L} \delta_l w_{jl} \phi(net_j)(1 - \phi(net_j)) \\ \text{if } j \text{ is an inner neuron} \end{cases}$$

Then the weight update equation is simply:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

[2]This is inspired from [14]

Where α is the learning rate. We used MATLAB's neural network framework to design, train, visualize, and simulate the network. We have a two-layer feed forward neural network. The hidden layer neurons have a sigmoid transfer function and the output layer neurons have a linear transfer function. A diagram of the final neural network selected is shown in Fig. 3

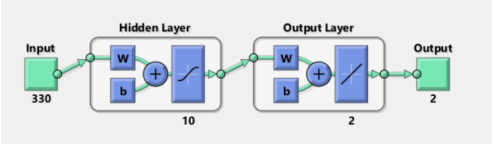

Fig. 3. Neural Network

V. RESULTS AND ANALYSIS

All our algorithms find average MSE in terms of the x and y locations of the points shown on screen. This is generalized to yield values in terms of viewing angle error. Horizontal and Vertical field of view is calculated using the screen size and distance of user from the screen which came out to be 52° and 30° respectively. Using the discretized number of bins for both directions we convert the L2 error in terms of viewing angle error as explained for one axis in [].

$$\theta_x = \tan^{-1}\left(\tan(HorFOV^\circ)\frac{AvgL2Dist}{\#HorBins}\right) \quad (2)$$

The dimension of the feature vector for all of the testing and training data used for all cross validation is 330. The makeup of the 330 features is:

- 300 features make up the vectorization of the 15x20 downsampled image of the eye from the original 480x640
- 20 features are the coefficients of the top 20 eigen eyes
- 10 features are the coefficients of the top 10 fisher eyes

A. HYPER-PARAMETER SELECTION

To select the hyper-parameters for each model of each approach, we perform hold-out cross validation where we train on a random subset of 70% of the data and test on 30% on the data. We then select the hyper-parameters that correspond to the model that has the smallest test error.

We use hold-out cross validation instead of k-fold cross validation because the time to train the SVR model and the neural network on MATLAB was prohibitively long.

1) REGULARIZED LEAST SQUARES: For regularized least squares, there is one hyper-parameter μ, which controls how much the model cares about over-fitting. The trade-off curve of hyper-parameters to test error is show in Fig. 4 The minimum on this curve is achieved with the model where $\mu = 0.003232$. Note that the model where $\mu = 0$ corresponds to the least squares model, without regularization. The fact that the minimum occurs when μ is non-zero means that the regularization drove some of the weights down and that improved the model. This means that there are a number of

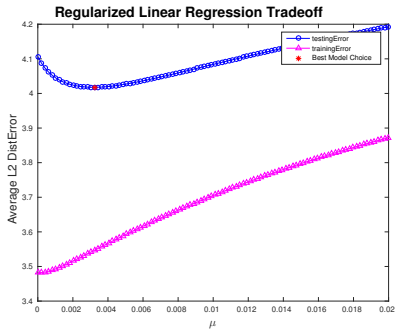

Fig. 4. Trade-off curve for least squares of testing and training error vs. μ. The minimum test error is marked with a red indicator.

redundant features, which was expected. This also means that our choice of downsampling the image to a 15x20 resolution still produced redundant features meaning that we have not downsampled the image too much.

2) SUPPORT VECTOR REGRESSION: There are two hyper-parameters to select for SVR are C and ϵ. Because we must optimize over two variable models, we present multiple trade-off curves superimposed in Fig. 5.

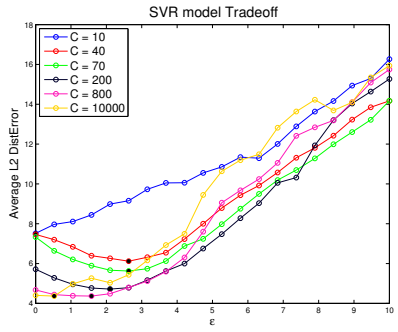

Fig. 5. Trade-off curve for SVR (linear kernel) of testing and training error vs. ϵ. Different values of C are presented as different lines. The minimum test error is marked with a black indicator.[4]

The minimum over all of the curves is achieved with the model where $\epsilon = 0.53$ and $C = 10000$. This result is surprising since the largest C value that was tested was used in the best model. This illustrates how important the slack variable is in SVR.

We did not present the results for other kernels, e.g., polynomial kernels, since the results seemed worse. Rather, we instead focused on using neural networks which seemed to yield a better result.

3) NEURAL NETWORK: With neural networks, our variable parameter was the number of neurons in the hidden layer. We tested out different values for the number of neurons and plotted the test and train error vs. number of neurons as shown in Fig. 6. As we see, the training error is minimized at when we have 10 neurons in the hidden layer, beyond this number although the training error decreases the test error increase which means the network is over-fitting. If we have less than 10 neurons both training and testing error increase which means the network is under fitting.

[4]The training error is not shown in the SVR trade-off plot in order to magnify the difference between the hyper-parameter settings.

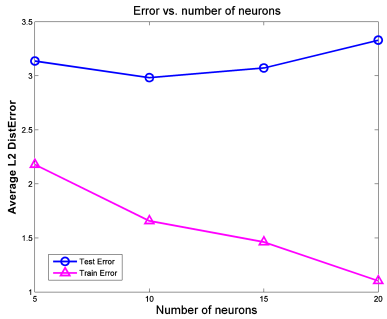

Fig. 6. Trade-off curve for neural nets, showing number of neuron vs. test and train error

B. SAMPLE OUTPUT

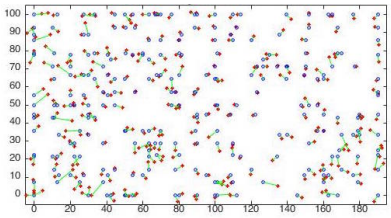

Fig. 7. Visualization using neural nets for training, showing actual points and estimated points

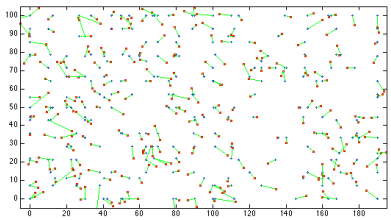

Fig. 8. Visualization using Regularized Least Squares for training, showing actual points and estimated points

Fig. 7 and 8 show the 30% of the outputs of the testing set to avoid a cluttered illustration. The blue points are the ground truth points shown to the users and the red points are the ones estimated by our algorithm (in MATLAB plot units). The green line between points is the distance between the actual and estimated points.

One thing we observed was that error is typically larger near the edge, regardless of which algorithm is used. We suspect that this is because the subject used to create the dataset needed to strain his eyes to reach the points in the far corners. As a result, the subject could have possibly not rotated his eyes all the way to view the indicator.

C. ABLATIVE ANALYSIS

We also carried out the analysis of varying the number of features as well as including the eigen eyes and fisher eyes

for one of the tests. The images were downsampled at four scales which gave different feature vector lengths and then training and testing was carried out on the baseline algorithm i.e. regularized least squares. According to Table I, the performance improvement beyond feature vector length of 300 i.e. after doubling feature vector length was marginal but with adding 30 PCA components there was improvement and hence 330 features were finally used for different algorithms.

TABLE I
TESTING ERROR (IN DEGREES) VARYING FEATURE LENGTH FOR REG. LINEAR REGRESSION

# feat	**99**	**300**		**588**	**972**
		no PCA	**+30 PCA**		
Error	1.857°	1.470°	1.452°	1.362°	1.333°

D. ALGORITHM COMPARISON

Using the optimal model for each algorithm, we have tested all three algorithms on the 30% validation data. The results are tabulated in Table II. As can be seen from the table the training and testing error are of the same order and comparable which implies that we are not over-fitting the training data.

TABLE II
TESTING ERROR (IN DEGREES) OF ALL APPROACHES

Algorithm	**Train Error**	**Test Error**
Linear Reg	1.250°	1.477°
Regularized Linear Reg	1.276°	1.444°
SVR	1.452°	1.570°
Neural Net	0.600°	1.071°

One conclusion we can make from these results is that neural networks work best compared to linear regression and SVR (using a linear kernel). The purpose of neural networks is usually to implicitly solve for relationships between the features, which makes it a more sophisticated algorithm compared to least squares, which simply minimizes the square loss objective. We suspect the reason SVR performs worse than least squares is that the data is not "linearly separable" in the regression sense.

VI. FUTURE WORK AND EXTENSIONS

A. PREDICTION APPROACH

This is a part of our CS 221 project wherein using past eye gaze locations we predict the future eye gaze location. Our models of prediction depend on the framework of Hidden Markov Models. Much like [10], our approach is also to model the true motion of the eye as a hidden random variable. Where our models differ is that our observed variables are (noisy) measurements from the eye tracker and the hidden variables are the true (noiseless) measurements. An illustration of a discrete HMM as a Bayesian network is shown in Fig. 9.

[6]Also called "evidence" in some literature.

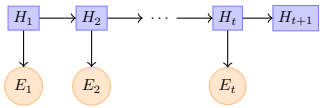

Fig. 9. An HMM with hidden variables H_i which each emit one discrete observed variable[6] E_i. Here the variable t is an integer that indexes time.

As Fig. 9 suggests, each variable H_i and E_i are distributed as follows:

$$H_i \sim P(H_i|H_{i-1}) \tag{3}$$

$$E_i \sim P(E_i|H_i) \tag{4}$$

When a Bayesian network has the form in Fig. 9 and has variables distributed as above, it is called an HMM. For the rest of this report we will refer to $P(H_i|H_{i-1})$ as the transition probability and $P(E_i|H_i)$ as the emission probability.

An HMM is completely specified by giving the hidden and observed variables as well as the transition and emission probabilities. First we will cover the positional HMM, which uses eye gaze location as variables and then we will cover the angular HMM which uses the angle of the eye gaze's velocity as variables.

1) POSITIONAL HIDDEN MARKOV MODEL: The first Hidden Markov Model implemented uses true eye gaze locations, i.e., (x, y) positions, in the scene as the hidden states and the sensor output as the noisy observations, i.e. ,$(\tilde{x}, \tilde{y})$ positions, of these hidden states.

2) ANGULAR HIDDEN MARKOV MODEL: The second Hidden Markov Model used for prediction uses true discretized directions (or angles) of the eye gaze's velocity as hidden variables, H_i, and the estimation of the eye gaze's velocity from the eye tracker as the evidence variables.

VII. CONCLUSIONS

The purpose of this paper was to evaluate different learning approaches to the task of eye gaze tracking. Our results have shown that a simple two layer feed-forward neural network outperforms linear regression, and a seemingly more complex approach: SVR. As we expected from using an image of the eye as a feature, there is are many relationships that can be made between pixels, which is the kind of data that works well in neural networks.

For future work, we would like to combine the CS221 approach for eye gaze prediction with the neural network approach to build a fast real-time application. We would also like to explore what performance gains we can expect by increasing the neural net layer, or by using convolutional neural networks.

REFERENCES

[1] Li, Dongheng, David Winfield, and Derrick J. Parkhurst. "Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches." Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on. IEEE, 2005.

[2] Merad, Djamel, Stphanie Mailles-Viard Metz, and Serge Miguet. "Eye and gaze tracking algorithm for collaborative learning system." ICINCO-RA 2006. 2006.

[3] Ji, Qiang, and Xiaojie Yang. "Real-time eye, gaze, and face pose tracking for monitoring driver vigilance." Real-Time Imaging 8.5 (2002): 357-377.

[4] Kim, Kyung-Nam, and R. S. Ramakrishna. "Vision-based eye-gaze tracking for human computer interface." Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on. Vol. 2. IEEE, 1999.

[5] Hansen, Dan Witzner, and Qiang Ji. "In the eye of the beholder: A survey of models for eyes and gaze." Pattern Analysis and Machine Intelligence, IEEE Transactions on 32.3 (2010): 478-500.

[6] https://www.kickstarter.com/projects/fove/fove-the-worlds-first-eye-tracking-virtual-reality/description

[7] SensoMotoric Instruments GmbH, (September 2015) SMI Eye Tracking Glasses 2 Wireless. Available at: http://goo.gl/NZjvwO

[8] Baluja, Shumeet, and Dean Pomerleau. Non-intrusive gaze tracking using artificial neural networks. No. CMU-CS-94-102. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[9] Chen, Jixu, and Qiang Ji. "A Probabilistic Approach to Online Eye Gaze Tracking Without Explicit Personal Calibration." Image Processing, IEEE Transactions on 24.3 (2015): 1076-1086.

[10] Yunlong Feng; Gene Cheung; Wai-tian Tan; Yusheng Ji, "Hidden Markov Model for eye gaze prediction in networked video streaming," in Multimedia and Expo (ICME), 2011 IEEE International Conference on , vol., no., pp.1-6, 11-15 July 2011

[11] H. Hadizadeh, M. J. Enriquez, and I. V. Baji, "Eye-tracking database for a set of standard video sequences," IEEE Trans. Image Processing, vol. 21, no. 2, pp. 898-903, Feb. 2012.

[12] O. V. Komogortsev and J. Khan "Kalman filtering in the design of eye-gaze-guided computer interfaces" Proc. 12th Int. Conf. Hum.-Comput. Interact. (HCI), pp. 1-10, 2007

[13] John Findlay and Robin Walker (2012) Human saccadic eye movements. Scholarpedia, 7(7):5095.

[14] A. Smola and B. Sch,lkopf, "A tutorial on support vector regression" Statistics and Computing, 2001

[15] Yunlong Feng; Cheung, G.; Wai-tian Tan; Le Callet, P.; Yusheng Ji, "Low-Cost Eye Gaze Prediction System for Interactive Networked Video Streaming," in Multimedia, IE EE Transactions on , vol.15, no.8, pp.1865-1879, Dec. 2013

[16] H. Hadizadeh, M. J. Enriquez, and I. V. Baji, "Eye-tracking database for a set of standard video sequences," IEEE Trans. Image Processing, vol. 21, no. 2, pp. 898-903, Feb. 2012.

[17] B.A. Smith, Q. Yin, S.K. Feiner and S.K. Nayar,"Gaze Locking: Passive Eye Contact Detection for HumanObject Interaction,"ACM Symposium on User Interface Software and Technology (UIST),pp. 271-280, Oct. 2013.

[18] "Backpropagation." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 10 December 2015. ¡http://en.wikipedia.org/wiki/Backpropagation¿

[19] https://www.web.stanford.edu/class/ee368/Handouts/Lectures/2015_Autumn/10-EigenImages_16x9.pdf