Convolutional Neural Network Architectures for Gaze Estimation on Mobile Devices

Matthew Kim
Stanford University
mdkim@stanford.edu

Owen Wang
Stanford University
ojwang@stanford.edu

Natalie Ng
Stanford University
nng1@stanford.edu

Abstract

Gaze estimation has numerous applications in human-computer interaction, psychology and behavioral research, pedestrian direction estimation, and more. Inspired by the results of Krafka et. al. on their iTracker convolutional neural network (CNN), we constructed our own CNN, dubbed Gazelle, with a similar architecture to predict the point on the device screen at which an iPhone user is looking. In addition to the raw images of users' face and eyes that Krafka et. al. used for iTracker, we supplement Gazelle with histogram-of-gradients (HOG) feature vectors computed over a cropped face image. This was done with the motivation of providing the CNN with a more semantically-rich feature than just pixel values that can better capture qualities such as head pose, qualities we believe are important to gaze estimation. We find lower losses using HOG as an input parameter on training and validation sets, when compared to a CNN without HOG.

We obtain an error of 4.85 on an independent test set for this method. These results show that our HOG improves results in the CNN relative to without, and that our method can be used to predict gaze on an iPhone screen.

1. Introduction

Graphical user interfaces on devices today rely heavily on traditional pointing devices like trackpads, touchscreens, and mouses for user input on most common tasks. However, real-time gaze estimation could generate a more fluid workflow for the connected world and lead the transition from screen-based technology to augmented reality. Work being done with gaze estimation include machine vision of facial expression to interact more intelligently with humans [1]. Successful implementations of eye tracking would have many applications in autonomous driving, psychological studies, and human computer interaction.

We will begin by loosely replicating a recent paper by Krafka et al. titled "Eye Tracking for Everyone" [2]. This paper developed a convolutional neural network (CNN) named iTracker in Caffe to predict human gaze onto a screen with from iPhones and iPads using the front-facing camera. Our architecture is inspired by Krafka's work on a public, crowdsourced dataset of video frames of faces where users were instructed to follow dots on their screens. The input to our algorithm are images of faces gazing into a mobile device screen, and we use a CNN to output a predicted gaze.

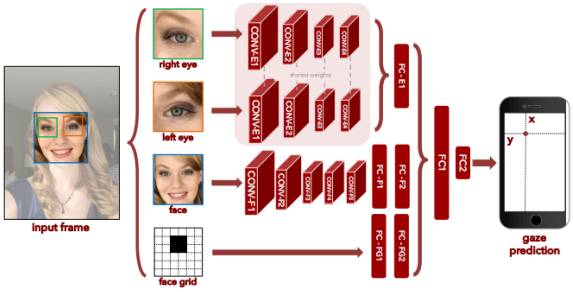

Figure 1. CNN developed by Krafka et.al. The inputs for their CNN are images for the right eye, left eye and face, and a face grid, which represents the location of the face in the FaceGrid. The CNN predicts an output location on an iPhone screen, where x and y are the horizontal and vertical distance from the camera.

2. Related Work

The most prevalent work in gaze estimation has been done using architectures based on screen light reflection on the eye [3,4]. One camera and corneal reflections from light can be used to estimate point-of-gaze [5]. Reflection-based models fall under the category of appearance-based models [4].

Another major category of models are shape-based models where images of eyes are rendered in computer vision to verify the importance of shape variations in eye training data [6][7]. Such approaches can range from voting-based models to model the iris as an ellipse[8,9] to efficient models that model the iris as a circle[10].

There are large benefits to such methods - minute movements in the pupil can be more accurately captured as the cameras are steady and illumination is constant. However, all of the above approaches require sophisticated setups, limited head movement, or formidable calibration tasks.

Finally, feature-based approaches try to identify local features to model eye parameters. While some have opted to detect the region between two eyes [11], other researchers have tried to detect the face region through skin color [12]. Though interesting, many feature-based models have been supplanted by appearance-based models, especially that of the Haar model presented by Viola and Jones [13].

There is a remarkable move towards convolutional neural networks in gaze estimation in the past decade. With appearance-based models finding a natural place in the state-of-the-art due to reduced need of calibration and increased robustness to real-world illumination, several papers have been written on appearance-based models on convolutional neural networks [2,14,15,16]. Convolutional networks have also been used to learn gaze prediction tasks where the task is to predict the location of the subject of an individual's gaze in an image [r8]. This further demonstrates the appeal of CNNs in a wide range and quantity of studies involving human gaze.

3. Methods

3.1. Overview

CNNs are a class of neural network algorithms with an architecture that makes the explicit assumption that inputs are images, which allows an encoding of certain properties that make it more efficient to implement with less parameteres needed than a normal feed-forward network. Because of the traditional success of CNNs on visual deep learning tasks, we decided to try an approach to gaze estimation common to the literature and also build a convolutional neural network from scratch with Tensorflow.

Given an input image, we designed several CNN architectures that are inspired by the architectures of Krafka, et al. [2] and George, et al. [g1]. Our innovative step that we hope to show is the extensions to the architectures, the features, and data preprocessing. Among the four architectures, three types of preprocessed data, and two features, we chose two models out of 24 to perform hyperparameter tuning on and generated test results for both.

3.2. CNN Architecture

Conceptually, for a given sample, the images of the face and two eyes are the most significant in estimating gaze. These images are fed through four convolutional layers with max pooling layers in between (to reduce number of parameters) and one or two fully connected layers. Depending on the model, a Histogram of Gradients feature was fed through one convolutional layer and two fully connected layers. The facegrid, as a boolean mask rather than an image, was fed into two fully connected layers without any convolution.

All layers, after propagating through convolutional layers and fully connected layers, converge upon one fully connected layer with 128 units that in turn map to an output representing our estimated gaze prediction in (X, Y) Cartesian centimeters from the camera.

Given Figure 1 as the **baseline model**, Figure 2 represents our **baseline model + max pooling** architecture. Our **baseline model + batch normalization** added a layer of batch normalization between the convolutional layers and the fully connected layers, and our **baseline model + maxpool + batch** combined both concepts.

3.3. Features

We trained all four iterations of our architecture with and without Histogram of Gradient features on our input.

We created an additional feature by computing a histogram of oriented gradients (HoG) from the cropped face image of each frame to feed into some of our CNN architectures (see Figure 4). This was based off of the implementation of HoG we coded in CS 231A, except adapted to calculated a HOG feature for multiple images (i.e. an NumPy array of dimensions [N, height, width]). For compatibility with our 144 by 144 pixel images, we set the parameters for HOG to be 12 pixels per cell, 2 cells per block, and preserved the number of bins as 9. The stride length of our block window was half the block dimension.

Ultimately, this allowed us to generate histograms (11x11x36) per face image. The removal of the HOG feature from Gazelle involved removing the histogram array as an input, and decoupling any and all layers involved with the HOG pipeline. At the point where fully-connected layers are concatenated to a final layer of 128 units, the HOG FC layer was simply omitted.

4. Dataset

4.1. GazeCapture

We use the GazeCapture dataset published by Krafka et al., who collected the dataset with the help of Mechanical Turk by asking 1474 participants to follow a dot on a device screen. A video using the front-facing camera of their device filmed a video of their face while they were performing this task. The dataset consists of over 2.5 million images, the video clips of each subject's face as they trace the dot broken up and stored as individual video frames. Data reliability was ensured by asking the user to tap either the right or left screen to ensure focus during the activity. The GazeCapture dataset was easily acquirable online; the biggest challenge was dealing with its unwieldy size. Finding stor-

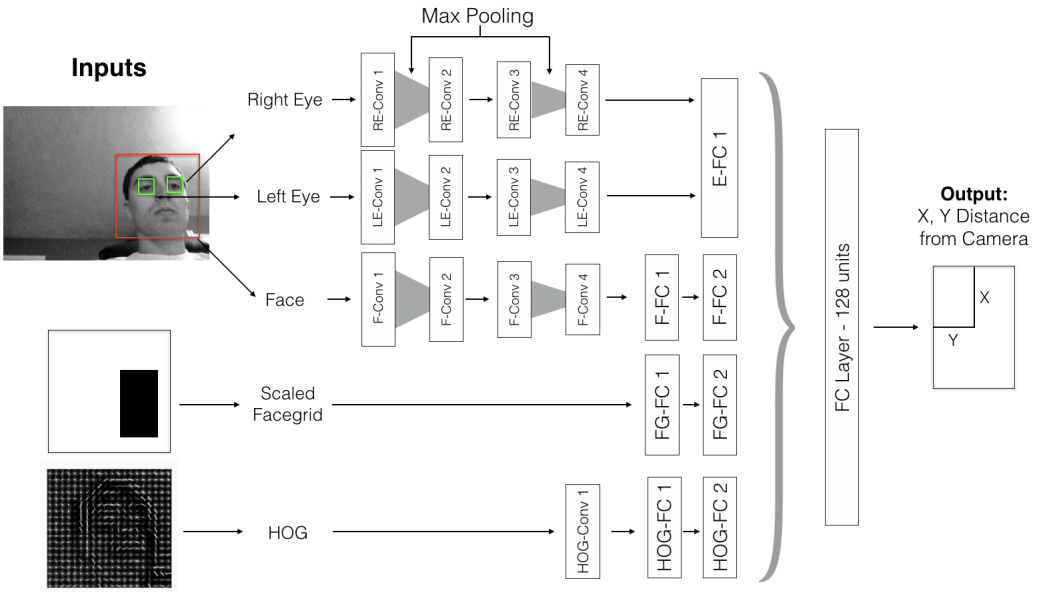

Figure 2. One of our four architectures. This involves feeding the left/right eyes, face, face grid, and optional features (such as HoG) through convolution and max pooling layers in parallel, and then flattening the arrays at the end to combine them into one long vector to pass through fully-connected (dense) layers. The last dense layer has two units, our output x and y distance (cm) from the camera. (CNN configuration for each layer, in filter size/ of filters. F/LE/RE-Conv 1: 11x11/48, F/LE/RE-Conv 2: 5x5/128, F/LE/RE-Conv 3: 3x3/192, F/LE/RE-Conv 4: 1x1/64, HOG-Conv 1: 3x3/64, E/F/FG-FC 1: 128 units, HOG-FC 1: 256 units, F/FG/HOG-FC 2: 64 units, final FC Layer: 128 units, final output (X,Y) layer: 2 units for x and y coordinate predictions.) Models with batch-normalization had such layers between F/LE/RE-Conv 4 and F/E-FC 1

age in the cloud and processing the gigabytes of data was extremely time consuming. This is discussed in more detail in the Results and Challenges section further below.

4.2. Cleaning the Data

The GazeCapture dataset includes the following information for each frame of the recorded video in json form:

1. the front-facing camera image of the subject as they perform the dot-tracking (hereon referred to as the frame),

2. the bounding boxes for the face and eyes in the frame,

3. the coordinates of the red dot to be looked at in centimeters in the X and Y directions from the camera, and

4. a boolean indicating image validity.

The boolean indicating image validity defined whether an input had images for the face and two eyes. Roughly 70% of the images in the GazeCapture dataset successfully had all three. Those that do not are not valid, and are excluded from this project.

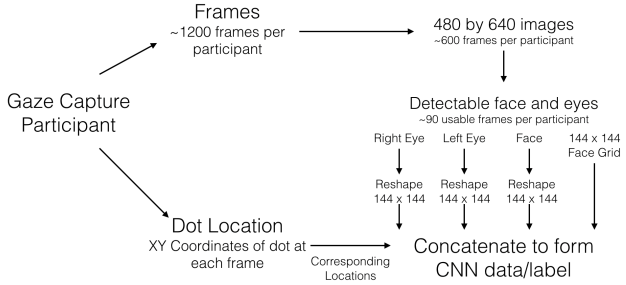

Figure 3. Workflow for extracting data. We divide into two training data sets, one where we rely on the MechanicalTurk from the JSON files, and one in which we use openCV to extract detectable faces and eyes and recreate the dataset from the original images. We then store the corresponding dot locations for our labels.

4.3. Data Inputs

Krafka and the MIT team's work on iTracker relied on bounding boxes of the faces and eyes that were determined by Mechanical Turk. Our goal was to eliminate the manual selection of features present in this step, and produce features automatically with computer vision techniques.

Hence, we chose to implement our own workflow

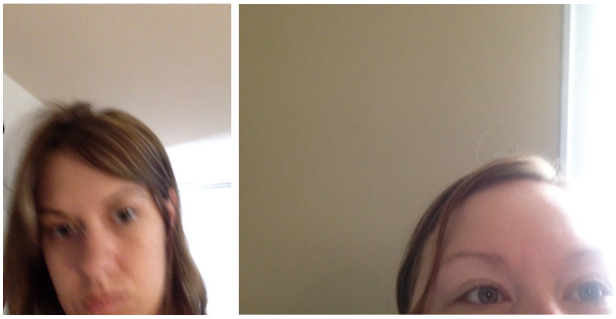

Figure 4. Example of noisy inputs in GazeCapture. GazeCapture has many frames where participants faces are either out of focus or cropped. The Haar detector of OpenCV has low success rate in detecting the eyes and face of these images for this reason.

for retrieving the bounding boxes of the face and eyes (Figure 2). To arrive at bounding boxes for the face and eyes from images from the GazeCapture dataset automatically, we utilize OpenCV (`cv2`) libraries. In particular, the `cv2.CascadeClassifier` module returns Haar cascades that detect various objects such as faces, eyes, eyeglasses, human bodies, and more [3]. We used pre-trained Haar cascades to detect faces and eyes in the image.

We observe that the classifiers on OpenCV are not perfect, especially with the low resolution and occasionally cropped images in GazeCapture (Figure 3). The classifiers may detect multiple or no faces and eyes in every frame. Since we have no way of validating which are true faces/eyes and which are false positives, we discard photos that do not have one identified face and two identified eyes.

From here, we crop out the square subarrays of the eyes and face and rescale these images to constant 144x144 images. We also created scaled facegrids which are boolean arrays that represent the location of the face in the 144x144 full image.

We encountered a few roadblocks with this method. In particular, we notice that the success rate of OpenCV is at 15% for GazeCapture images, which means that very few of the samples are used. In effect, only 350,000 images were theoretically usable to train the CNN. In addition, Stanford's corn clusters had no GPU acceleration and faced many connection issues, so detecting the face and eyes from images took a very long time.

FILL THIS IN HERE ABOUT THE 2 types of data

4.4. Data Outputs

The coordinate system of the GazeCapture X, Y labels indicating the true location of the subject's gaze was devised to be a normalized space that is generalizable to multiple devices, such as smartphones, tablets, or laptops, and also in different orientations (landscape or portrait). The X value indicates the distance (cm) to the left or right of the camera on the image plane that the true location is, and the same is true of the Y value in the up and down direction. This coordinate system takes advantage of the fact that the image plane is typically nearly normal to the line of sight between the camera and the subject's face. We elect to use this coordinate system instead of a more traditional one that measures the pixel distance because it is less noisy for the CNN and more compatible between different types of devices, which may also have different physical pixel dimensions.

4.5. Feature Vectors in CNNs

iTracker does not apply any geometric or feature detection techniques to the inputs of the CNN, due to the author's belief that the CNN can learn anything from the images [2]. In fact, there have been few studies that attempt to use feature detection as an input to a CNN. One notable example uses feature vectors in a classification problem to find humans in images. They found that while CNNs give excellent results without feature vectors, the addition of this additional input did provide a competitive advantage [9]. This motivates our investigation of using Histogram of Ordered Gradients (HOG) feature vectors as inputs to our CNN.

5. Experiments and Results

5.1. Training

For each of the two data types (MechanicalTurk from JSON and openCV on original), we chose among 8 models or combinations of four architectures (model, + batch-norm, + max-pool, + both) and two features (with and without HoG).

Our error metric was Root-Mean-Squared-Error on a mini-batch. Below are the relevant hyperparameters that we chose as a sufficient balance between rate of convergence and loss. We held these hyperparameters constant only in the task of choosing among the eight models.

Hyperparameter	Value
Epochs	10
Learning rate	0.00001
LR decay coeff.	0.9^{epoch}
Mini-batch size	20
Optimizer	SGD
Num Training	21761

On a Google Cloud virtual machine instance with 52GB RAM, 8 vCPUs, 2 NVIDIA Tesla K80 GPUs, training on the roughly 21,000 samples took about 4 hours on models with max-pool and/or batch normalization. The vanilla "baseline" that we modeled after iTracker from Krafka, et

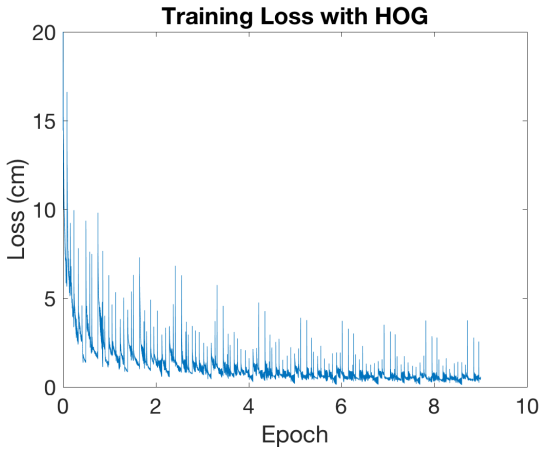

Figure 5. Training loss over 10 epochs over our training data, on the max-pooling model.

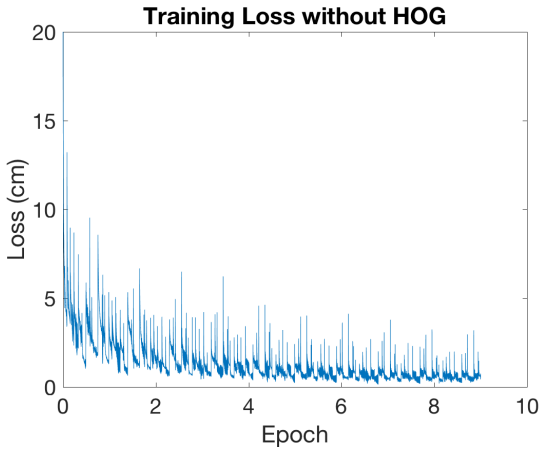

Figure 6. Training loss over 10 epochs over our training data, ran on the tweaked max-pooling model without HoG.

al. took more than 7 hours. Given more time to train on larger training sets, we would expect lower training losses as the model reshapes neurons to become more robust to new inputs.

6. Results

In Figures 5 and 6, we see an example of the training loss with the architecture **model + max pooling**. It is clear that the model has overfit to some degree, as the test and validation losses hover around 5cm by epoch 10, the training losses are close to 1cm. For every architecture that we tested, training loss neared 0.75 average error loss (cm) by epoch 10.

With a difference in about 0.31 cm average loss between models with Hog as a feature and without, we chose to keep HoG as a feature to test on our four architectures on the Turk and openCV-manipulated datasets.

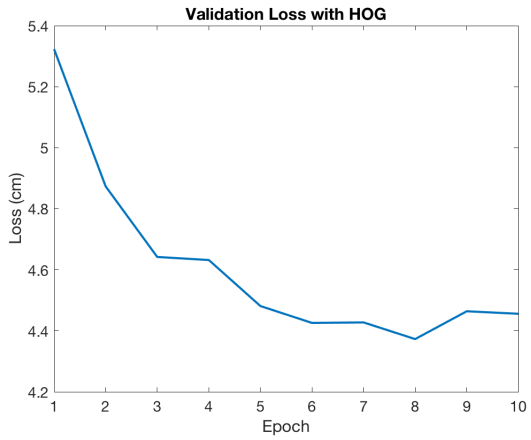

Figure 7. Average loss on the validation set at each epoch, for max-pooling with HOG. We reach the minimum at epoch 8, with loss 4.3729 cm.

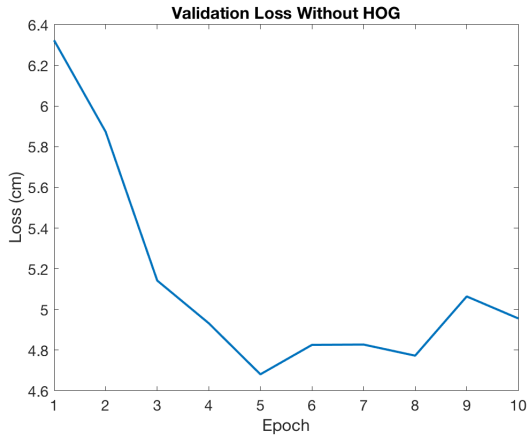

Figure 8. Average loss on the validation set at each epoch, for Gazelle without HOG. We reach the minimum at epoch 5, with loss 4.6811 cm.

We find that on the model with max-pooling and batch-normalization with HoG on the Turk dataset yielded the best results at 3.32 test error.

7. Conclusions, Challenges, and Extensions

We find that we were able to build a workflow from the ground up that takes images from GazeCapture, performs face and eye detection, extracts feature vectors, and trains a convectional neural net to predict gaze upon an iPhone screen. While our error was higher than that of GazeCapture's iTracker (4.85 cm versus 2.58 cm), we observe that this error is usable for predicting gaze upon a screen, particularly when generalized to larger devices. We observe that HOG does seem to improve the results of the CNN, but

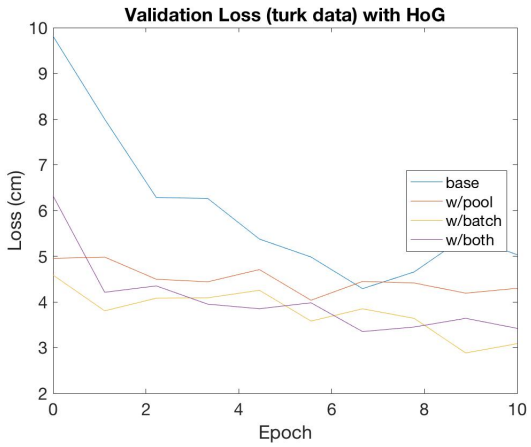

Figure 9. Validation loss on the Turk dataset with HoG as a feature.

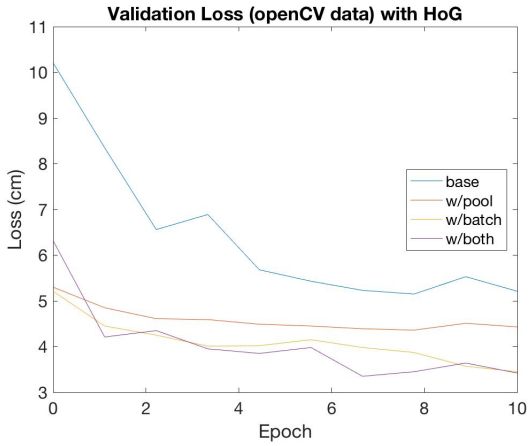

Figure 10. Validation loss on the openCV dataset with HoG as a feature.

due to restrictions in data quantity, this conclusion is not definitive. Moreover, given a more robust implementation of Haar, changes in hyperparameters, and a greater number of training samples to feed into our CNN, we are confident we could generate lower errors on our testing data.

Method	Average Error (cm)
Center	7.54
Gazelle (both on openCV)	3.63
Gazelle (both on Turk)	3.32
AlexNet	3.09
iTracker	2.58

The largest challenge associated with Gazelle was in data preprocessing. We implemented our workflow using OpenCV's Haar detectors, which severely limited the usable images in GazeCapture. Additionally, we were running our method on Stanford's Corn clusters, which do not have a GPU, and have very sporadic connections. Thus, compared to GazeCapture which used upwards of 1.5 million valid images in training, we were limited to 25,000 images. We were able to get surprisingly good results despite this small dataset size, especially since GazeCapture showed that the accuracy of the method scales with more images from different participants. The obvious next step would be to find a cluster with a GPU and use a face and eye detector with more accuracy than OpenCV to greatly improve dataset usage.

Inherent challenges with the technique itself involve low quality images in the dataset that often crop off portions of the face, and therefore make creating a face grid or inputting an image of the face into the CNN impossible.

Due to the promise of using HOG as part of the CNN, we envision using other techiques from CS231A as inputs to our CNN. We could combine corner detectors with Haar detectors from OpenCV, for example, to detect lip and eye corners. These would then provide 4 points that could be used to create a homography between the quadrilateral in the image, and square. This homography would be indicative of the the head pose, which would be useful information for the CNN.

Here, we present an architecture built from start to finish that inputs an image and predicts gaze upon an iPhone screen. While we were not able to obtain the accuracy of GazeCapture, this can be attributed to fixable issues of data quantity. We also perform an important investigation of the impact of feature vector inputs to CNN architectures. This work is generalizable to larger devices and is a step towards improving human-device interactions.

8. References

1. M. Bartlett, G. Littlewort, I. Fasel, and J. Movellan."Real time face detection and facial expression recognition: Development and applications to human computer interaction." (CVPR) 2003.

2. A. Duchowski. Eye tracking methodology: Theory and practice. Springer Science & Business Media, 2007.

3. K.Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik and A. Torralba. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016.

4. R Lienhart and J Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. Semantic Scholar.

5. D. W. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. PAMI, 2010.

6. K.-H. Tan, D. J. Kriegman, and N. Ahuja. Appearance-based eye gaze estimation. In WACV, 2002.

7. Nian Liu, Junwei Han, Dingwen Zhang, Shifeng Wen, and Tianming Liu.Predicting Eye Fixations using Convolutional Neural Networks. IEEE Computer Vision Foundation. 2015.

8. Srinivas S S Kruthiventi, Kumar Ayush, and R. Venkatesh Babu. DeepFix: A Fully Convolutional Neural Network for predicting Human Eye Fixations. IEEE. 2017.

9. Adria Recasens, Aditya Khosla, Carl Vondrick Antonio Torralba. Where are they Looking. Advances in Neural Information Processing Systems (NIPS), 2015.

10. Yunsheng Jiang and Jinwen Ma. Combination Features and Models for Human Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015.

11. A. Krizhevsky, S. Sutskever, and G Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.