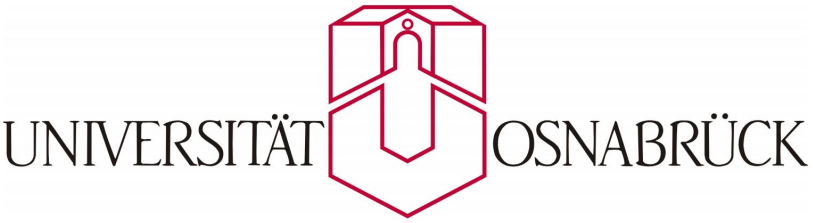


Using Convolutional Neural Network for Development of an Eye Tracking Application, Capable of Running on Mobile Devices

by

Sahar Niknam

Submitted to the University of Osnabrück

in Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Cognitive Science

First Supervisor: Peter König, Prof. Dr.

Second Supervisor: Felix Weber, M.Sc.

September 30th, 2020

Abstract

Researchers at Massachusetts University of Technology (MIT) claim that they have designed a gaze tracking neural network (iTracker) with a competitive performance with the state-of-the-art results. But most importantly, it is capable of running on mobile devices. For training the iTracker, they have built a large-scale dataset (GazeCapture), which is a set of about 2.5 millions images of about 1500 individuals looking at different fixation points on the screen of their mobile devices. The inputs to the network are square crops of both eyes and the face, and a grid of face location within the original frame, while the coordinates of the fixation points are the labels. This thesis is an effort for improving the iTracker performance by first, removing the color channels to reduce the data size and computational load, and secondly, adding the nose crop to the inputs to get a better estimate of the head pose, and finally by processing the left and right eye crops separately. However, due to the lack of computational resources, we ran the experiments on a small subset of the GazeCapture (with 9,990 frames). We got better accuracy in all the experiments, compared to the result of the original iTracker settings, trained on the same small subset of the data.

Acknowledgement

I would like to thank my supervisors, Professor Peter König, for showing me the way and for cutting the red tape at the most stressful moments, and Felix Weber for his great ideas and encouraging and continuous support throughout the entire way. I would also like to especially thank Johannes Schrumpf, whose inspiring idea is the cornerstone of this work.

Table of Contents

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 1

Introduction

Eye tracking has a broad range of applications which we can categorize into two main groups. First, eye tracking with an exclusive interest in the physical movements of the iris and the second, gaze tracking for finding the content of the look. Eye tracking for tracking eyes is pervasive for diagnosis of medical conditions mainly for screening mental disorders like attention deficit hyperactivity disorder (ADHD) [8], autism [4], schizophrenia [11], and Alzheimer's disease [21]. Eye tracking for tracking the gaze, on the other hand, covers a more diverse range of applications which all shares an interest in attention tracking and cognitive behavioral analysis. As some examples, we can mention human-computer/robot interaction[2, 16], marketing [29], natural language processing [20], and virtual teaching/learning assistant [31].

There are two main groups of methods for eye tracking. The oldest methods are the invasive ones. Invasive methods either are head-mounted devices, or their attachments have contacts with skin or eye. Electro-oculography, infrared oculography, scleral contact/search coils, and head-mounted video-based eye tracking systems are some examples of the invasive eye tracking methods. The second group is the non-invasive or remote methods, which are basically video-based remote eye tracking systems [5, 9]. These systems record the eyes with one or two cameras, and process the recorded frames either based on a geometric model of the eye or based on the data and exclusively by studying the recorded images [15]. Using artificial neural networks, or particularly convolutional neural networks (CNNs), for processing images of the eyes is an example of non-invasive, data-based eye tracking systems, which goes back to the 1990s [3],

All these methods have one thing in common, and that is the fact that their functionality depends on professional, sensitive, and expensive equipment and settings. Working with these equipment, consequently, needs an expert who supervises the measurements, as well. Under these circumstances, eye tracking would be more laborious than it is supposed to be, regarding its widespread application. With the growing computational power of mobile devices like smartphones and tablets, that motivates a recent and revolutionary trend that aims to turn

every phone to a potential eye tracking device that is relatively cheap and everyone without any professional knowledge of eye tracking can operate it. Among many efforts for designing an eye tracking application, a group of researchers at the Massachusetts Institute of Technology (MIT) ran a project to use neural networks for the eye tracking task. Khosla et al. first built an unprecedented large-scale dataset specialized for gaze prediction for their project. And then, they trained a hybrid neural network consisting of CNNs and fully connected subnets on their dataset. They published their results in 2016 in the paper "Eye Tracking for Everyone" [15][1]. In the paper, they reported that their model achieved competitive results with the state-of-the-art accuracies, especially among the video-based remote eye tracking methods. But the importance of their work lies in the claim they have made that their model is capable of running on mobile devices and consequently putting "the power of eye tracking in everyone's palm" [15].

[1] Literally speaking, the term *eye tracking* covers methods which exclusively concerns the irises position with reference to the head. *Gaze tracking* or *eye gaze tracking* on the other hand is interested in *attention*, therefore it also takes the head pose into account to find the direction of the look. However, *eye tracking* covers and is in use for both concepts. And that is the case in the "Eye Tracking for Everyone" paper.

Chapter 2

"Eye Tracking for Everyone"

Khosla et al. [15] tried to realize a software-based gaze tracking method, using the ever-growing potentials of neural networks. The authors of the paper, "Eye Tracking for Everyone," [15] claim that their gaze tracking model, the iTracker, which they trained on a specialized dataset, has an equally satisfactory performance as the current methods and devices. Additionally, they tested their trained model on other available datasets and achieved state-of-the-art results. Therefore, considering the fact that all the commonly used gaze tracking devices work with sensitive equipment, and consequently their usage is costly and needs expert's attendance, one can consider the development of an entirely software-based gaze tracker with a competing performance, as a valuable achievement. Especially that the software, according to the authors, is capable of running on a regular smartphone, and in real time.

To this end, the authors gathered and augmented a large-scale dataset for their experiment. Khosla et al. especially designed the dataset, known as the GazeCapture dataset, for the purpose of gaze tracking, rather than simply tracking the iris or the pupil as in eye tracking tasks.

2.1. GazeCapture

For an efficient training of a neural network, the high variability of the dataset is essential. To fulfil this criterion, Khosla et al. decided to build their own dataset instead of using the available ones which are all small and not sufficiently diverse. One of the undesired but inherent characteristics of the lab-produced datasets is uniformity. For collecting the data, researchers invite the subjects to their labs for the measurements and recording. Thus, the same environment with the same solid background, the same lighting conditions, and roughly the same head poses is coded in the entire dataset. In addition, bringing subjects to a lab, that is, the necessity of the physical presence in a specific place, will inevitably reduce the diversity of the population, not to mention the size of it. To overcome these issues, the authors decided to

collect their dataset using crowdsourcing methods. By doing so, they could recruit individuals across the globe, from a widely diverse population. And since the subjects collaborated from their own residence, the diversity of the backgrounds, lighting conditions, and the head poses is outstanding (e.g. some subjects recorded themselves while lying in their beds).

Achieving this goal, the authors developed an iOS application, called *GazeCapture*[2], and recruit subjects through the Amazon Mechanical Turk [1] which is a global crowdsourcing platform. They asked the subjects to download and install the application and run the experiment on their own Apple devices. The application displays black dots with pulsing red circles around (to keep the subject attentive) at random locations on the device screen. Each dot appearance lasts 2 seconds, and starting from 0.5 second after the appearance of the dots, the application takes pictures of the subject using the front facing camera. Furthermore, the authors instructed the subjects to change the orientation of the device after every 60 dots, to add further to the variability of the dataset. They also encouraged the subjects to move their head during the time they are looking at a dot.

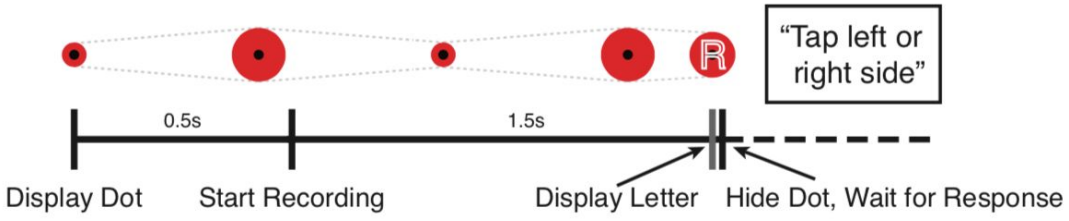


Figure 2.1. "The timeline of the display of an individual dot. Dotted gray lines indicate how the dot changes size over time to keep attention" (Khosla et al., 2016).

The downside of remote crowdsourcing is the questionable reliability of the collected data in the absence of a supervisor. Regarding the specific task of this project, two main issues are imaginable: first, the subject's face and eyes may not be fully captured within the frame, and secondly, the subject may be occasionally distracted and not looking at the dot. To solve the first problem, the authors exploited the real time, built-in face detector of the iOS to make sure the camera captured the face mostly. For the second problem, they designed the app to display one of the two letters *L* and *R* inside the pulsing dots for about 0.05 second, shortly before they

[2] https://apps.apple.com/us/app/gazecapture/id1025021075

disappear. Based on the displayed letter, subjects had to tap on either the left or the right side of their devices' screen. Failing to do so, subjects needed to repeat that dot recording.

2.2. Dataset

The GazeCapture dataset consists of 1474 folders, each of which, according to Khosla et al., contains recorded frames of one individual. The dataset includes 2,445,504 frames in total. That makes an average of 1659 frames per subject, with a range varying from 4 to 3590 frames in a folder. Each frame is a color image with 3 RGB color channels saved in *JPG* format with the dimensions of 640 × 480 pixels, 96 dpi, and a bit depth of 24. The average size of a frame is nearly 50 KB, which led to a total size of 144 GB for the GazeCapture dataset. Every folder, in addition, contains 9 files containing metadata on the recorded frames and saved in *JSON* format. These JSON files include either a dictionary or a list readable in Python.

2.2.1. Images

The images in the GazeCapture dataset are closed-up frames of individuals, divers in race and almost divers in (legal working) age, looking at a fixation point on a digital screen in front of them. The frames are captured with a broad range of various environments, backgrounds, lighting conditions, and head poses. All the images have the same dimensions (640, 480) but they are in two, portrait and landscape, orientations.

Figure 2.2. Sample frames from the GazeCapture dataset (Khosla et al., 2016).

2.2.2. Metadata

The metadata files include: *appleFace.json*, *appleLeftEye.json*, *appleRightEye.json*, *dotInfo.json*, *faceGrid.json*, *frames.json*, *info.json*, *motion.json*, *screen.json*.

The appleFace.json, appleLeftEye.json, and appleRightEye.json are dictionaries with 5 keys: *X*, *Y*, *W*, *H*, and *IsValid*; the values are lists of the X and Y coordinates of the top left corner, and the width and height of a square cut of the image that captures the face, the left, and the right eye for each frame. The X and Y coordinates of the face are with respect to the top left corner of the frame, and the X and Y coordinates of the left and right eyes are with respect to the top left corner of the face crop. The authors extracted these data using the iOS built-in eye and face detectors. The *IsValid* key takes either 1 or 0 as value, indicating whether a frame properly captured the face and both eyes or not.

The faceGrid.json is also a dictionary with the same keys delivering lists of X and Y coordinates of the top left corner, and the width and height of the face box within a canvas, representing the rescaled version of frames to a 25 x 25 square. The *IsValid* value is the disjunction of the *IsValid* values of the same frame in the appleFace.json, appleLeftEye.json, and appleRightEye.json files.

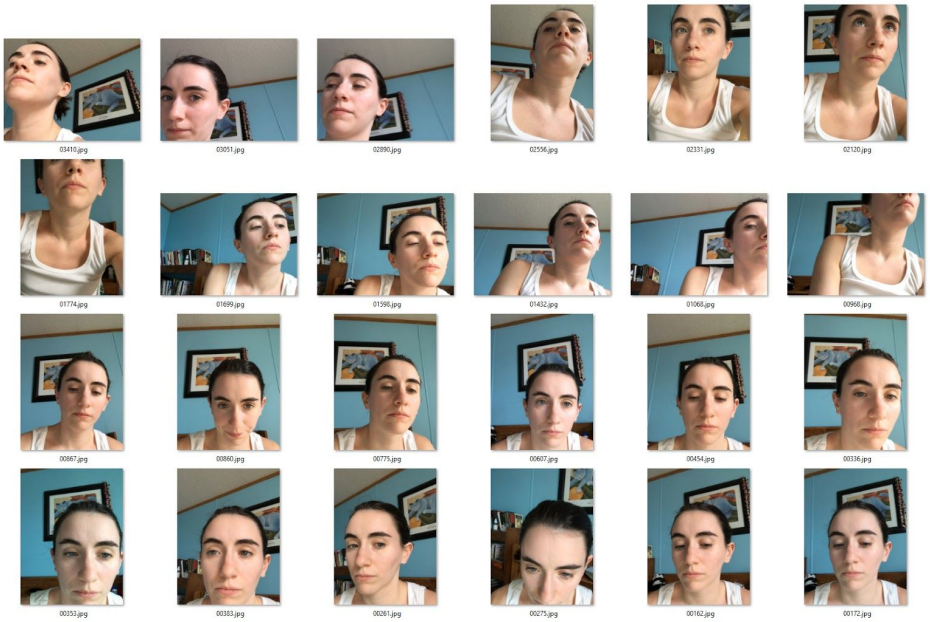


Figure 2.3. Sample frames of one subject.

The dotInfo.json is another dictionary with 6 keys: *DotNum*, *XPts*, *YPts*, *XCam*, *YCam*, and *Time*. According to the authors, the DotNum value is the counting label of the displayed dot for each frame, starting with 0. For example, if the first 14 values in the DotNum list are 0, and the next 15 ones are 1, that means in the first 14 frames, the person was looking at one dot, and in the next 15 frames they were looking at another dot with a different location. The XPts and YPts are the locations of the center of the dots, in points [25], with respect to the top left corner of the screen. And XCam and YCam are presumably giving the same information, except in centimeters and with respect to the center of the device's front facing camera. The *Time* is a list of time spans in second, indicating how long after the display of a dot the camera captured the corresponding frame.

The frames.json file recorded the names of all the frames in the folder.

The motion.json file is a list of the device motion data recorded at 60 Hz frequency. Each element of the list is a dictionary with 10 keys: *GravityX*, *UserAcceleration*, *AttitudeRotationMatrix*, *AttitudePitch*, *AttitudeQuaternion*, *AttitudeRoll*, *RotationRate*, *AttitudeYaw*, *DotNum*, *Time*. The first 8 keys report the measurements of the attitude, rotation rate, and the acceleration of the device, retrieved using Apple's *CMDeviceMotion* class. The DotNum is the same value as in the dotInfo.json file. And the *Time* is the temporal distance between the first appearance of that dot and the exact time of recording the motion parameters.

The info.json file a small dictionary that gives 5 pieces of information: the number of *TotalFrames* in the folder, the *NumFaceDetections* and the *NumEyeyDetections* that indicate how many frames captured the face and the eyes adequately, the *Dataset* that says the subject belongs to which one the train, validation, and test sets (based on the arrangement that the authors used to train their neural network for the reported results in the paper), and finally the *DeviceName* that is the type and the model of the device that took the picture. There are 15 different devices recorded in the GazeCapture dataset: 8 models of the Apple's smartphones (iPhone) summing up to 85% of the dataset and 7 models of the Apple's tablets (iPad) that makes the remaining 15% of the GazeCapture.

The screen.json is another dictionary with 3 keys: *H*, *W*, and *Orientation*. The H and W values recorded the height and width of the device screen. The *Orientation* takes 4 values: 1 for

the portrait position with camera on top of the screen, 2 for the portrait position with camera below the screen, 3 for the landscape position with the camera on the left, and 4 for the landscape position with the camera on the right side of the screen.

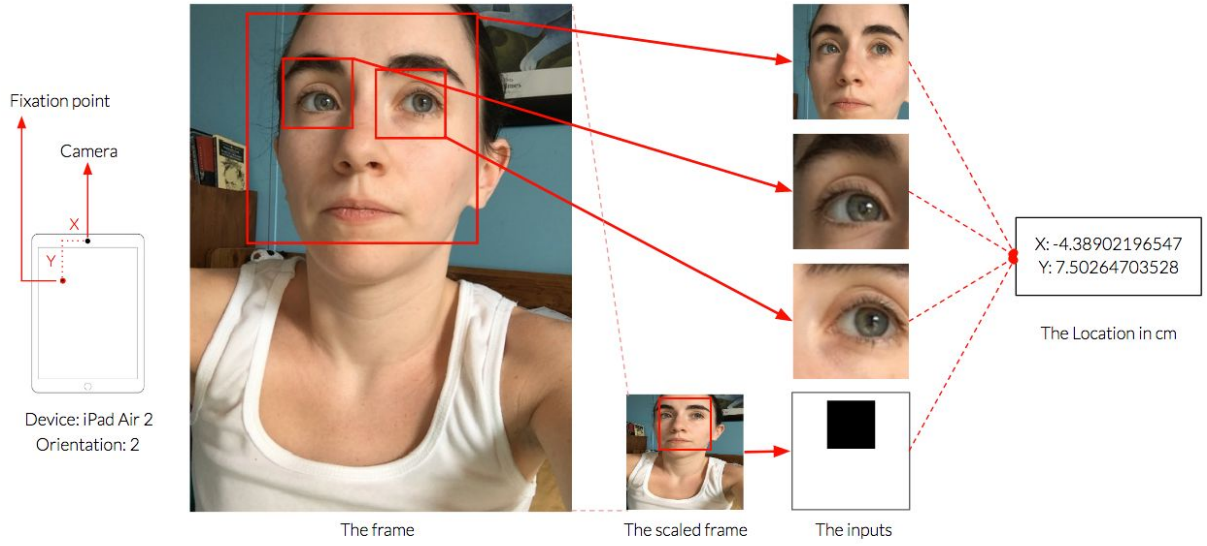


Figure 2.4. Visualization of a sample frame's metadata.

2.2.3. Dataset Analysis

Khosla et al. has created a precious large-scale dataset with high variability which is incomparable with the previously generated gaze tracking datasets. However, a closer inspection of the GazeCapture dataset reveals a few issues and some inconsistencies between the dataset description in the paper and in the iTracker Github repository [14] that could question the reliability of the dataset for gaze tracking training.

Inaccurate count of the subjects. The authors reported that they had collected the GazeCapture dataset from 1474 subjects, each of which has a folder containing their recorded frames in the dataset. While it may not be a significant difference, a review of about 500 of the folders revealed that, at least, 50 pairs of the folders share the images of the same individuals (e.g. both folders *00533* and *00534*). Even though the frames in different folders are not identical, the number of subjects falls short of 1474.

Numerous invalid frames. The authors reported the size of the GazeCapture dataset as large as almost 2.5 million frames. But the number of those frames that they had tagged as invalid, is 954,545. That makes about 40% of the dataset unusable. While even cutting the

GazeCapture dataset in half would leave a dataset 6 times as large as the next largest gaze tracking dataset, yet the difference between the actual and the nominal size of the dataset is significant. And unfortunately, the authors failed to address the issue properly in the paper, especially when comparing it with the other datasets in the table 1 of [15].

Invalid frames that are valid. Checking sample folders of the GazeCapture dataset showed that the reason behind such a large fraction of invalid frames is technical errors, due to Apple's weak technology for the face and eye detection at the time. Many of these invalid frames are clear images of the subjects' full faces without any occlusion of the eyes.

Figure 2.5. Samples of invalid frames in the GazeCapture dataset without any visible defect.

Some of these frames do not have perfect lighting conditions (intense light source behind the subject) or have the subject's face slightly out of the frame, while the eyes are clearly captured. Filtering out frames with these minor problems made the GazeCapture less of the diverse dataset that the authors claim it to be, especially regarding the lighting condition and head pose.

Valid frames that are invalid. Khosla et al. did not give a detailed description of the GazeCapture image recording, however, delving into the metadata, we can see that they recorded about 14 images of a subject while the person was looking at a dot. According to the

authors, the duration of the fixation for each location was 2 seconds and the recording started 0.5 second after the display of the dot. That means they captured about 14 images during a period of 1.5 second, which is a relatively narrow window. And with the *Tap left/Tap right* arrangement in the last 0.05 second of this window, one may expect perfect capture of subjects attention. But reviewing about 1% of the GazeCapture dataset proved otherwise. Many frames in the dataset capture subjects with closed eyes or while they are looking away. Nonetheless these frames are tagged as valid and have coordinates as corresponding labels. Apart from the closed-eyes and looked-away frames, the chance of iOS detecting eyes' locations erroneously, even though insignificant, is not eliminated.

Figure 2.6. Samples of closed-eyes and looked-away frames in the GazeCapture dataset.

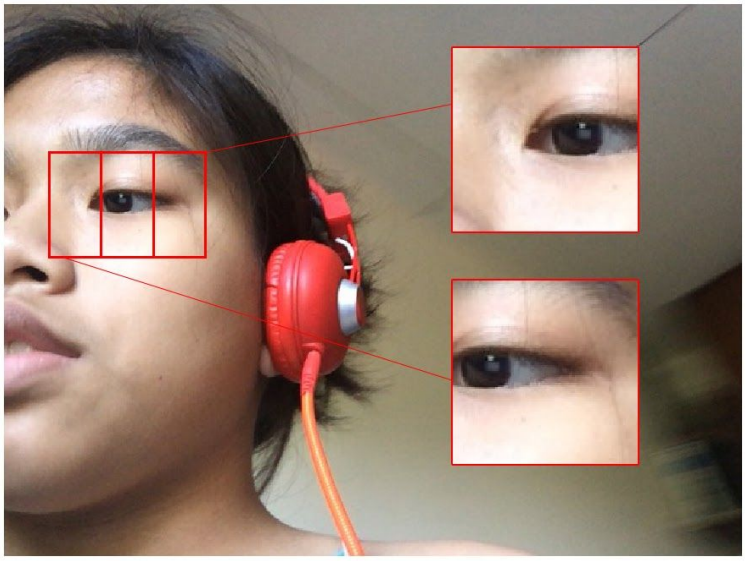


Figure 2.7. For the frame *02728.jpg*, in the folder *02967*, iOS detected the left eye for both left and right eye crops and tagged the frame *valid*.

Fixation location. Unfortunately, Khosla et al. are not clear about the locations, the number per se and per subject, and the distribution of the displayed fixation dots. However, they have mentioned that not all the subjects have images captured of them looking at the full set of the dots. But even the subjects with a full set of fixation locations do not have each location in 4 possible orientations.

The fixation locations are the labels in the GazeCapture dataset. The metadata gives the labels data in two formats: points (XPts, YPts) [25] with the origin being the top left corner of the screen, and also in centimeters (XCam, YCam) with the origin being the center of the front facing camera. The two origins have a constant distance on one unique device. That means the location in points and the location in centimeters, regardless of the orientation of the device, should be a one-to-one or a reversible transformation. Based on the definition of the point unit [25], the only exception to this rule happens when the *Display Zoom* of the device changes (We can estimate the Display Zoom from the change of the screen size values in the screen.json file). Reviewing the authors' codes for converting (XCam, YCam) to (XPts, YPts) and vice versa [17, 18, 19] confirms these assumptions. However, inspecting the dotInfo.json files opens up that only 98 subjects, out of approximately 1474, have the same number of unique coordinates in points and in centimeters. While the majority of the subjects in the GazeCapture dataset either have 76 and 90, or 93 and 120 unique coordinates in points and in centimeters, respectively. That is, one single location in points has more than one corresponding coordinate in centimeters. And that happens when the orientation of the device changes from 1 to 2, or from 3 to 4. But according to [19], it should not be the case. For example, frames *00742.jpg* and *01763.jpg* in the folder *02156* have the same screen size and the same fixation location, while they have two different, and not even symmetric, coordinations in centimeters, only because they differ in orientation (their orientations are 4 and 3, respectively).

2.3. iTracker

Khosla et al. wrote their model, iTracker, in Caffe [12] that is a popular language in computer vision, for coding convolutional neural networks. But in the paper's Github repository, the codes are also available in Pytorch [14].

As a side note, so far, neither a natural nor a legal person developed the iTracker model, in either of experimental or commercial capacity, into an application for mobile devices for using on massive scale; the only exception is that one of the authors of the "Eye Tracking for Everyone" paper, Kannan, H. D., built an application based on the model for limited and experimental usage as a part of her doctoral dissertation [13].

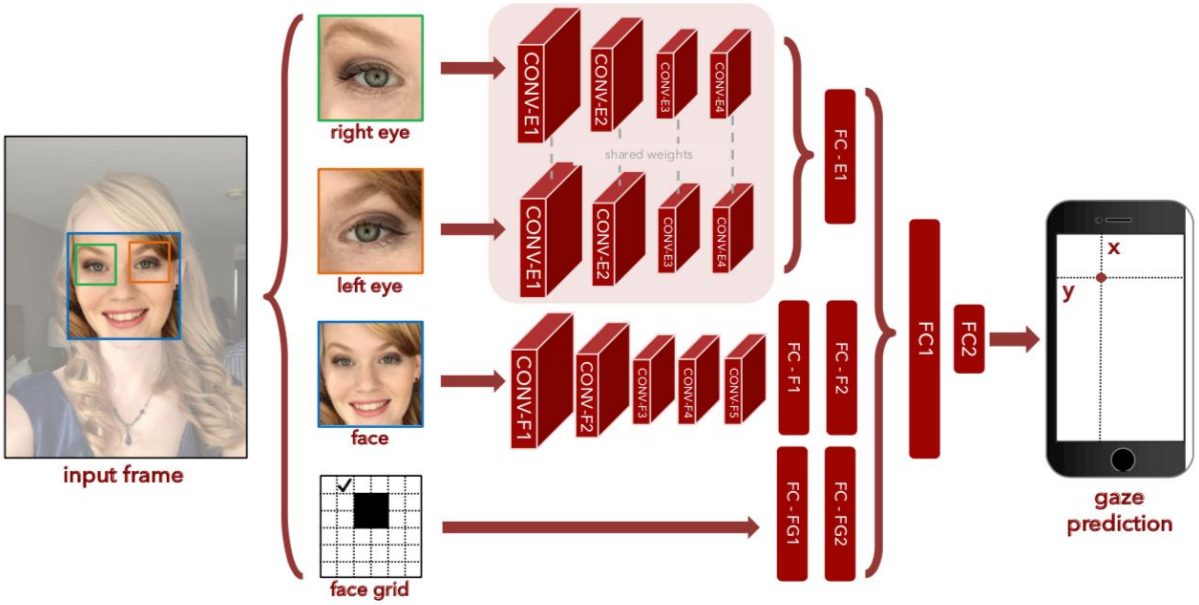


Figure 2.8. The architecture of the iTracker model (Khosla et al. 2016).

2.3.1. Architecture

The iTracker model has a complex, parallel architecture. The model takes 4 inputs: the left eye crop, the right eye crop, the face crop, and the face grid. The eyes and the face crops are cut out of the original frame based on the appleLeftEye.json, appleRightEye.json, and the appleFace.json instructions, and then rescaled to a square patch of dimensions 224 × 224 pixels. The face grid crop is a 25 × 25 pixels square patch, created with the faceGrid.json metadata. Each input is fed to a subnet whose output is concatenated with the others', on different levels.

The left and right eye crops are fed into two identical CNNs that share the weights. In practice, that means feeding both eyes into one single CNN, but one at a time. The eyes' CNN has 4 convolutional units with 96, 256, 384, and 64 filters, and 11, 5, 3, and 1 kernel sizes, respectively. All the strides sizes are 1, except for the first unit in which the kernels jump over 4 pixels in every move. Following the first two convolutional layers, first comes a max pooling

layer with a pool size of 3 and stride size of 2, and then a *local response normalization* layer. All the activation functions are ReLU. The output of the eyes' last convolutional layer is flattened, concatenated, and fed into a single dense layer with 128 neurons and ReLU activation function. The output of the dense layer is the output of the processing of both eye crops.

The face crop is fed into an identical CNN with the eyes' CNN. Except that the flattened output of the final convolutional layer is fed into two dense layers, sequentially; first one with 128 and the next one with 64 neurons, and both with ReLU activation function. The output of the second dense layer is the face processing output.

The face grid is simply flattened and then fed into two sequential dense layers, with 256 and 128 neurons, and ReLU activation functions. The output of the second dense layer is the face grid processing output.

And finally, the eyes, the face, and the grid outputs are concatenated to a 1 × 320 vector and fed into a dense layer with 128 neurons and ReLU activation function. The final and output layer is another dense layer with only 2 neurons (for the coordinates regression) with ReLU as its activation function.

Table 2.1. The summary of the iTracker model.

Layer (type)	Output Shape	Param #	Connected to
input_6 (InputLayer)	[(None, 224, 224, 3)	0	
conv2d_8 (Conv2D)	(None, 54, 54, 96)	34944	input_6[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 26, 26, 96)	0	conv2d_8[0][0]
lr_n2d_4 (LRN2D)	(None, 26, 26, 96)	0	max_pooling2d_4[0][0]
conv2d_9 (Conv2D)	(None, 26, 26, 256)	614656	lr_n2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 12, 12, 256)	0	conv2d_9[0][0]
lr_n2d_5 (LRN2D)	(None, 12, 12, 256)	0	max_pooling2d_5[0][0]
conv2d_10 (Conv2D)	(None, 12, 12, 384)	885120	lr_n2d_5[0][0]
input_8 (InputLayer)	[(None, 224, 224, 3)	0	
input_9 (InputLayer)	[(None, 224, 224, 3)	0	
conv2d_11 (Conv2D)	(None, 12, 12, 64)	24640	conv2d_10[0][0]
input_7 (InputLayer)	[(None, 25, 25)]	0	
model (Model)	(None, 9216)	1559360	input_8[0][0] input_9[0][0]
flatten_3 (Flatten)	(None, 9216)	0	conv2d_11[0][0]

flatten_4 (Flatten)	(None, 652)	0	input_7[0][0]
concatenate_2 (Concatenate)	(None, 18432)	0	model[3][0] model[4][0]
dense_7 (Dense)	(None, 128)	1179776	flatten_3[0][0]
dense_9 (Dense)	(None, 256)	160256	flatten_4[0][0]
dense_11 (Dense)	(None, 128)	2359424	concatenate_2[0][0]
dense_8 (Dense)	(None, 64)	8256	dense_7[0][0]
dense_10 (Dense)	(None, 128)	32896	dense_9[0][0]
concatenate_3 (Concatenate)	(None, 320)	0	dense_11[0][0] dense_8[0][0] dense_10[0][0]
dense_12 (Dense)	(None, 128)	41088	concatenate_3[0][0]
dense_13 (Dense)	(None, 2)	258	dense_12[0][0]
Total params: 6,900,674 Trainable params: 6,900,674 Non-trainable params: 0			

2.3.2. Training

Dataset splits. Khosla et al. divided the GazeCapture dataset into three subsets of train, validation, and test splits. Instead of randomly distributing the frames, they picked 1271, 50, and 150 subjects for the train, the validation, and the test subsets, respectively. Studying the metadata shows that the train, validation, and test data subsets contain 1251983, 59480, and 179496 frames. They made sure that all the subjects in the validation and the test sets would have frames for the full set of fixation locations. In addition, for the trial that led to their best reported results in the paper, they also augmented both train and test datasets 25-fold, by shifting the eyes and the face and the face grid accordingly.

Parameters. In the paper "Eye Tracking for Everyone" [15], the authors did not provide the reader with a detailed outline of the training. And available information in the GazeCapture Github repository [14] conflicts the paper's, which could be due to the different rounds of training. In the paper, Khosla et al. mentioned that they had achieved their best results by training the network "for 150,000 iterations with a batch size of 256" [15]. But they have never discussed the number of epochs. In the neural networks literature, the end of an *iteration* is punctuated with the update of the trainable parameters of the model (usually, the weights). That is, the *iteration* ends after training the network over one full batch. On the other hand,

training the network over each and all data samples for one round, is called an epoch [26]. Therefore, knowing the data size, one can calculate the number of iterations (per epoch) by having the batch size and vice versa. On this account and considering that the neural networks do not have a set-in-stone terminology, it seems that by *iteration*, the authors meant *epoch*, which is a common mistake. With this assumption, the authors trained their best model with a learning rate of size 0.001 for 75,000 epochs and then continued training it for 75,000 more epochs with a learning rate of 0.0001. They also used a momentum of size 0.9 and weight decay rate of size 0.0005 over all the 150,000 epochs of training. Khosla et al. did not discuss the optimizer that they have used for training, but according to their published codes [14], they applied stochastic gradient descent (SGD).

Accuracy metrics. The authors reported the iTracker's error as the average over all the Euclidean distances between the fixation points and the model predictions, and in centimeters.

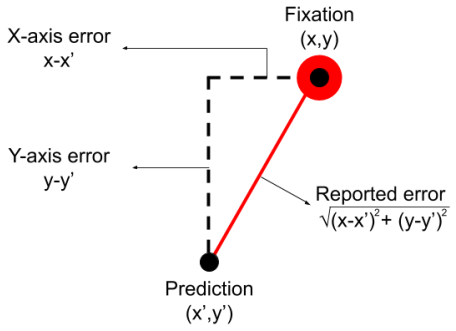


Figure 2.9. The authors calculated the error as the Euclidean distance between the fixation and the predicted point.

However, they also calculated the average errors for the smartphones and the tablets separately. According to the authors, the reason for this decision was to cancel out the screen size, and also the user-to-screen distance differences among different types of devices. In addition, for the best results that they have achieved, Khosla et al. also "fine-tuned" [15] the iTracker model for *each device* and *each orientation*. But they did not give any details about the quality of these fine-tunings. Taking these arrangements into account, they achieved a prediction error of size 1.71 and 2.53 centimeters for smartphones and tablets, respectively. The authors also found out that they can reduce the error size to 1.34 for smartphones and 2.12 for tablets, in case of running a 13-fixed-point calibration based on the works of Xu et al. [30].

Furthermore, the authors also reported the model *dot error*, which is the average Euclidean distance over all frames with dots at the same location. The advantage of calculating the dot error, according to the authors, is that it sheds a more realistic light on the model accuracy. Because in practical applications of models like iTracker, the accuracy over a series of successive frames is in question, not a single frame. Khosla et al. reported the lowest dot error of the iTracker model 1.83 and 2.38 for smartphones and tablets, respectively.

Chapter 3

Experiments

We tried to test three ideas for improving the performance of the iTracker model in the gaze tracking task. Each of these ideas falls under one aspect of neural networks in practical applications:

Preprocessing. Does the color information contribute to the purpose of gaze tracking?

Inputs. Does the nose localization improve the iTracker model performance?

Architecture. Do both eyes convey identical information in the gaze tracking context?

3.1. Dataset

For testing the thesis's ideas, we decided to work on a smaller subset of the GazeCapture dataset. First, we randomly sampled 10,000 valid-tagged frames, regardless of the device type or the orientation. Then we dropped 9 frames, due to the MTCNN failure in face detection, and one more frame because of the erroneous metadata. Therefore, the miniature dataset of the thesis consists of 9,900 frames. Additionally, we adjusted the eyes and face crops sizes by rescaling them to 64 × 64 pixels (instead of 224 × 224 as in the paper description).

The immediate necessity of reducing the size of the dataset and decreasing the images size was the shortage of the computational resources for processing tera-scale data size with a model with about 7 millions of trainable parameters (more than 9 millions in the nose crop test), for 150,000 epochs. The system that we had at our free disposal was a gaming machine with a GPU of NVIDIA GeForce® GTX 1070 with 8 GB GDDR5 VRAM, that is, 1920 CUDA cores. Using Google Colab was another option with a higher GPU power. Nevertheless, the disk space for uploading the data to the cloud was not enough. In addition, even with a Pro Google account, which is not accessible for non-US-based users, the maximum possible runtime on the Colab is 24 hours; while replicating the original experiment needs weeks of training.

The miniature dataset is considerably smaller than the GazeDataset, both in number and size of the samples. Regardless, that could not affect the reliability of the results of this thesis, as our primary purpose was to investigate the outcome of some adjustments for improving the *performance*, rather than to raise the benchmarks for improving the *outcome*. To that purpose, we also trained the original iTracker model on the miniature dataset to make the comparison the possible.

3.2. Model

For the thesis's experiments, we worked with the same iTracker model as the authors outlined in the paper [15]. But for finer details, which are missing in the paper, our work heavily relies on the Pytorch codes available in the GazeCapture Github repository [14]. However, we coded the model in TensorFlow 2.1.0, and *tf.keras* submodule with version 2.2.4-tf. And for replicating the hybrid architecture of the model, we used Keras *Model()* class with the *functional* API. The code is accessible on the thesis's Github repository [23].

3.2.1. Architecture

During the translation from Caffe to Keras, two problems came up. First, implementing the Pytorch *torch.nn.CrossMapLRN2d* layer [28] in Keras, and then adjusting the *groups* argument of the *tf.keras.layers.Conv2D* layer [27].

The CrossMapLRN2d layer is a local response normalization layer that normalizes activation function output in a local neighborhood by looking over all the feature maps of the previous convolutional layer. This type of normalization is helpful, particularly when using ReLU as the activation function. Because ReLU is a linear function and therefore its responses are unbounded. Normalizing ReLU response over a local area keeps the model sensitivity sharp for the most important features captured in the data. The Keras API had an equivalent LRN2D layer for the CrossMapLRN2d, but the developers decided to remove the layer in 2015. However, the LRN2D class's code is still accessible through the remove commit on the Keras Github

repository [6]. Therefore, we employed the same code for implementing an equivalent local normalization layer as a child of the *tf.keras.layers* class.

The argument *groups* in the *tf.keras.layers.Conv2D* layer is an option for forming groups out of the input channels / feature maps, and run the filters on each group separately. Basically, every filter convoles each and every channel. In this manner, one can consider *grouping* as one more step towards feature localization. In both Pytorch and Keras, *groups* is an argument defined for the convolutional layer, nonetheless, our code threw error after setting the *groups* to 2 (as in the Pytorch code [14]). Our efforts to solve the problem came to a dead end, especially because changing the *groups* value (from its default that is 1) is not a popular practice in Keras. Therefore, we could not find enough resources addressing the issue, except a brief mention to version conflicts. Consequently, we commented the line out.

Other than that, our model is a loyal replication of the iTracker. However, since we feed smaller size crops, the total number of trainable parameters of our model reduced from the original 6,900,674 to 3,460,034 with the miniature dataset.

3.2.2. Training

For training, we made all the decisions mainly in line with an effort to compensate for the small size of our dataset. For all the experiments, we uploaded the corresponding weights of the reported results to the thesis's Github repository [23].

Dataset splits. To overcome the small size of the dataset, we decided to exercise the K-fold cross validation method to use up the entire set of samples. Therefore, instead of splitting the dataset into three sets of train, validation, and test, we activated the *validation_split* argument in the *fit* method of the *model* and set it to 20%. That is, in every epoch the code trained the model on 1592 random samples and held 398 ones out as the test set.

Parameters. Regarding the dataset size, we also trained the network for a proportionally smaller number of epochs to avoid overfitting. Therefore, we first trained the network for 100 epochs with a batch size of 128, and then 20 more epochs with a batch size of 32. The learning rate was 0.1 for all 120 epochs. We also worked with Adam optimizer with all its default

parameters, except for the learning rate. For the loss we implemented the *Mean Squared Error* (MSE).

Accuracy metrics. For comparing the accuracy of the experiments, we measured the error with the *Mean Absolute Error* (MAE) metric (in centimeters). In the original experiment, Khosla et al. used Euclidean distance which works with squared error (L2 distance), whereas MAE measures the absolute difference (L1 distance). We decided to train the model with observing the L1 distance for the network to focus on minimizing its misjudgments over X- and Y-axis independently. We consider that as an optimal treatment when the data distribution along the X-axis conveys different information compared to the Y-axis distribution. In addition, as Kannan, H. D., one of the authors of the "Eye Tracking for Everyone" paper [15] discusses in her doctoral dissertation [13], the Euclidean distance metric is sensitive to outliers. This metric, because of its squared terms, wastes training efforts on including the extreme cases, instead of fine-tuning the weights on the average ones.

In measuring our experiments' prediction errors, unlike in the original experiment, we did not calculate and report MAE specifically for each device, orientation, or dot, nor did we fine-tune the model for training over any of these variables.

3.3. Experiment 1: Grayscale Images

In this experiment, we converted all the color inputs (eyes and face crops) to grayscale images. For the conversion, we used a method of the *OpenCV* library (`cv2.COLOR_BGR2GRAY`). The inputs in this experiment are the same as in the paper: eyes and face crops and face grid; except that the crops are in grayscale and have dimensions of 64 × 64 × 1. The model, in this case, has 3,413,570 trainable parameters.

The motivation behind this experiment is to reduce the data size and thereby the computational workload. By dropping the color channels, the dataset will shrink to one-third of its original size, and it cuts off about 47,000 trainable parameters.

3.4. Experiment 2: The Nose

In the nose experiment, either explicitly or implicitly, we try to investigate whether the nose localization can help improve the accuracy of the iTracker model. The idea is based on the fact that, unlike simply tracking the eyes which needs detecting iris/pupil, gaze tracking success heavily depends on the head pose detection. The head pose in turn, geometrically depends on the location of major facial landmarks, including the nose tip [22]. To reflect the nose data and convey its data to the network, we tested three methods: with a vector, with a grid, and with a crop. But in all these three cases, to get the nose data we used the MTCNN library.

3.4.1. MTCNN

Multi-Task Cascaded Convolutional Network (MTCNN) is a lightweight CNN for face detection and alignment in real time which its developers claim to have superior accuracy over the state-of-the-art methods and algorithms. The core idea of MTCNN is to run a CNN on a cascade of scaled versions of the original image. Zhang, Zhang, Li, and Qiao introduced the idea in their paper "Joint face detection and alignment using multitask cascaded convolutional networks" in 2016 [32]. In 2017, the Github user, *ipacz* [24], wrote a Python (pip) library based on the MTCNN idea, which is now available to users for free.

MTCNN takes an image and returns a list of detected faces in the image, each of which is a dictionary with 3 keys: *box*, *confidence*, and *keypoints*. The *box* key returns a list of four values of the X and Y coordinates of the top left corner, and the width and height of the face box (in pixels). The *confidence* is the reliability of the detection (in percent) and it may vary in each run. And the *keypoints* is a nested dictionary of 5 facial landmarks coordinates, including the approximate center of the left and the right eyes, the nose tip, and the left and right corners of the mouth.

```
1 !pip install mtcnn
2 from mtcnn.mtcnn import MTCNN
3 faces = MTCNN().detect_faces(img)
4 faces
```

```
>>> [{'box': [90, -39, 277, 381],
  'confidence': 0.9999997615814209,
  'keypoints': {'left_eye': (155, 111),
   'mouth_left': (166, 259),
   'mouth_right': (263, 264),
   'nose': (207, 186),
   'right eye': (285, 118)}}]
```

Listing 3.1. Install, import, and run MTCNN.

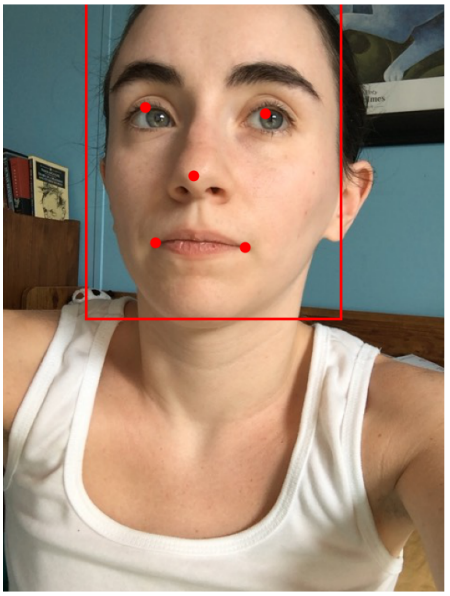
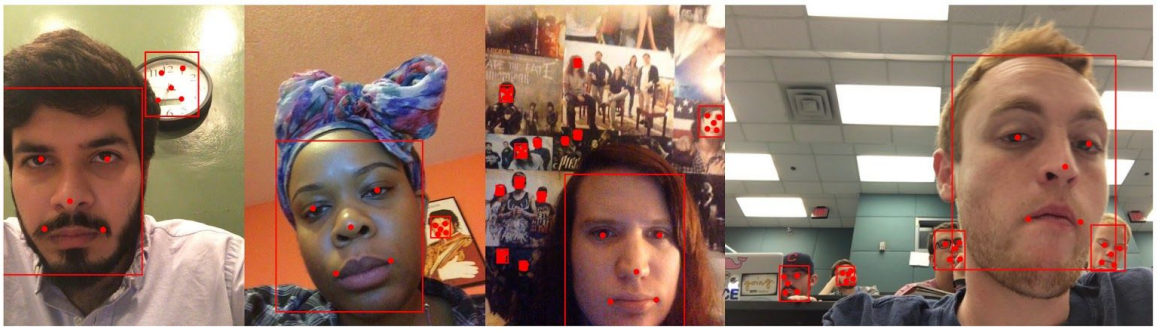


Figure 3.1. The MTCNN results are plotted with a red box and red dots.

In this thesis we took advantage of the MTCNN library for two primary purposes. First for elaborating the face grid concept, numerically and visually, and secondly and more specifically for getting the nose tip coordinates to cut out the nose crop. To do so, we ran MTCNN on the 10,000 frames that we had sampled randomly and saved the results as a Python list. However, as noted earlier, MTCNN returns a list of detections. That is, if there are more than one face in the image, MTCNN will return the facial landmarks of all the detections in the frame and not only the face of the subject of interest. The possible irrelevant detections include people in the background, pictures of people, and in some infrequent cases, objects that resemble a human face (Fig. 3.2a). To locate the subjects faces among an indefinite number of detections, we validly assumed that the subject's face box would be the largest detected box in the frame. Based on this assumption, before getting the coordinates of the subject face detection, we iteratively compared the first two elements in the list and removed the detection with the smaller face box width, until only one detection was left. In addition, we also had frames in which MTCNN could not find any face. We spotted them out simply by checking whether a list is empty. Although it is noteworthy that, over three trials, MTCNN had an average of 8 failures out of 10,000 frames that is barely 0.1% failure rate.



(a)

(b)

Figure 3.2. (a) MTCNN recognizes inanimate/irrelevant faces. (b) MTCNN recognizes no face.

3.4.2. Vector

For the *vector* test, we got the MTCNN generated coordinates of both eyes and the nose tip for all frames and created a *numpy* array of float32 values with a shape of [9990, 6]. Then we added an extra subnet to the original iTracker model for processing the vector. The vector subnet consists of two dense layers: the first one with 128 and the output with 64 neurons, and both layers have ReLU as their activation functions. In the final concatenation phase of the model, we concatenated the vector subnet output along with the output of other subnets. The trainable parameters of this model sum up to 3,267,842.

The motivation for running this test was to investigate if the model could, in addition to the visual inputs, extract an auxiliary pattern out of merely numerical values of the eyes and nose locations and map them to the fixation point coordinates. That is, the vector test stresses the absolute coordinates of the eyes and nose tip without having the face box as a reference.

3.4.3. Grid

In the *grid* test, we replaced the paper's face grid with one made with the MTCNN-derived coordinates. We got the coordinates of the face box, both eyes, and the nose tip from MTCNN, and created a white canvas with the same dimensions as the frame. Then we drew a black box with the MTCNN's face box coordinates, and inside the box, plotted a white triangle whose vertices are the MTCNN's detections for the two eyes and the nose. And as the final step, we scaled down the canvas to 50 × 50 dimensions, that is 4 times larger than the paper's face grids. We decided to keep the face grids up to that size since downsizing them to 25 × 25 pixels would have distorted and debunked the extra, fine details of the eyes-nose coordinates.

The grid test model architecture is the same as the original iTracker. We changed only the content of one input, and because of that the number of trainable parameters of the model raised to 3,940,034.

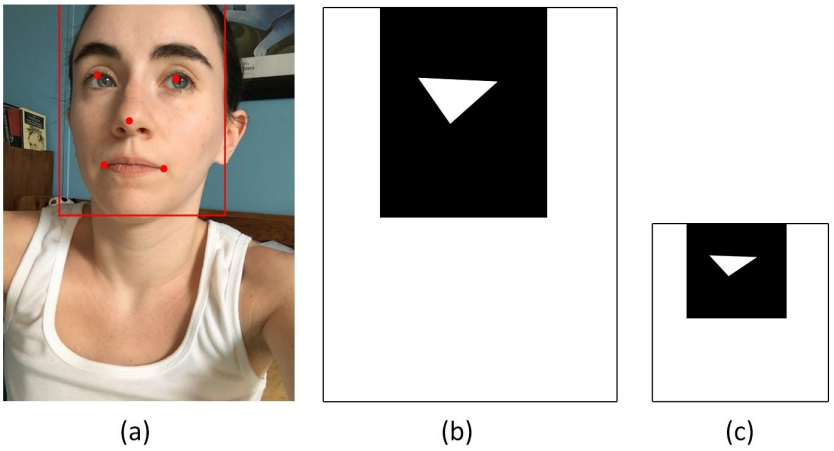


Figure 3.3. (a) The frame augmented with the MTCNN detections in red box and dots. (b) The grid with the face box and the eyes-nose triangle on a white canvas with the same size as the frame. (c) Rescaled grid to 50 × 50 pixels.

Multiple improvements motivated the grid test. The first one was to keep the face ratio in the grids. The face grids, as the GazeCapture described them, have black *squares* as the face box. In more than 99% of the frames, the face boxes are accurate squares. And almost in all the frames of the left 0.5% of the dataset, the difference between the face grid height and width is only 1 pixel, which is due to the noise of down scaling to such small dimensions. But according to Kannan, H. D. [13], preserving the ratios improves the accuracy of the model. Secondly, for

generating the grids in this test, we used the MTCNN detections which, considering Apple's technology for the face and eye detection at the time of GazeCapture development [7], outperform the original dataset metadata. And as the final motivation, we augmented the face grid with the eyes-nose triangle that contributes to the head pose delineation and consequently to the gaze tracking. However, compared to the crop test, the grid test downplays the high-level visual features of the nose and works with the raw geometry.

3.4.3. Crop

In the *crop* test, we took the tip of the nose coordinates from the MTCNN and the width of the left eye[3] crop from the GazeCapture metadata. Then we cut nose squares out of the frames with the MTCNN's nose tip coordinates at the center and with the same size of their corresponding eye crops. Finally, to make the nose crops consistent with the other crops, we rescaled them to 64 × 64 pixels. We added a subnet to the iTracker model for processing the nose crops, with the exact same architecture as the face subnet. In the concatenation phase, we slided the output of the nose subnet in, along with the other subnets. Except for one extra subnet, the model in the crop test is identical to the original iTracker. The extra subnet increased the model's trainable parameters to a total of 5,068,738.

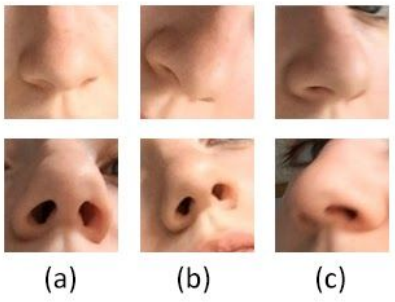


Figure 3.4. The nose crop indications of the head rotation around: (a) X- (b) Y-, and (c) Z-axis.

The crop test is focused on the nose, not only the coordinates and localization, but also the high-level visual characteristics of the nose. That is, by feeding the nose crop into a CNN,

[3] In the GazeCapture dataset, both left and right eye crops of a frame are squares and have the same size.

we were hoping the network would extract task-relevant features that address the nose alignment and the head pose.

3.5. Experiment 3: The Eyes

In the original iTracker model, eyes subnets share the weights. Theoretically, that means with each input moving through the left eye subnet, the right eye subnet updates its weights, as well. But practically speaking, the idea is as simple as feeding both eye crops into one subnet. That is, with feeding one sample, unlike the other subnets which update their weights for one round, the eyes subnet gets two inputs and updates its weights for two rounds. In the eyes experiment, we have changed the core architecture of the iTracker model by setting *two* but *identical* CNNs for processing the eye crops. We made the change by replicating the eyes subnet, removing the concatenation phase of the two eyes, and inserting both eye subnets output directly into the list of the outputs in the final concatenation phase. Despite that, the eyes experiment takes the same inputs as the original experiment. However, for splitting the eyes subnets, the number of trainable parameters increased to 5,002,882.

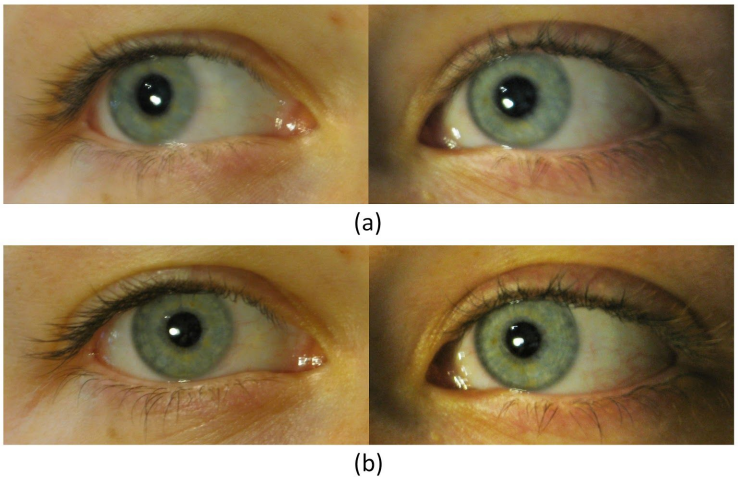


Figure 3.5. The irises' positions in the sclera are more consistent when the fixation point is at (a) a far distance, compared to (b) a close distance.

The idea of the eyes subnets sharing weights would make sense only if we assume that both eye crops, regarding the gaze tracking task, are conveying the same information and that the right eye is only emphasizing the features already extracted from the left eye. However, we believe that while in general, both irises keep the same position in the sclera, this will be less and

less accurate as the gaze location gets closer to the eyes. And that is the case when looking at the screen of a mobile device, as small as smartphones and tablets. Therefore, we decided to test the iTracker performance with two disconnected eye subnets.

Chapter 4

Results

We present the results of the thesis's experiments and tests in terms of loss (MSE) in centimeters squared and error (MAE) in centimeters. For reporting the error, we did not consider calculating and averaging over devices, orientations, or dot locations. We did not fine-tune the networks for neither of these variables. We ran all the experiments and the tests under similar conditions and with similar parameters, and only changed the factors that we wanted to investigate their effects on the training process. We also integrated the successful experiments and tests into one single model and trained it on the same dataset, as well. And for being able to compare our results with the iTracker, we also ran the original model on the miniature dataset.

4.1. Experiments Schematic

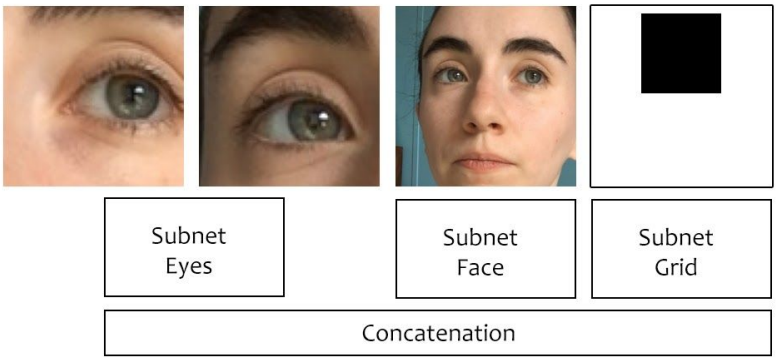


Figure 4.1. The paper model (miniature) schematic.

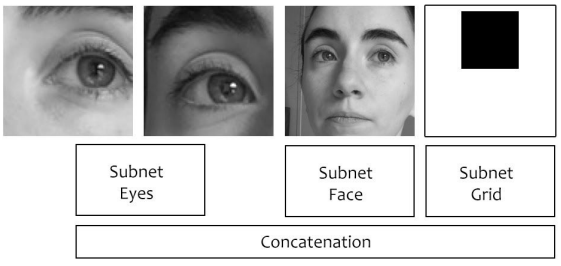


Figure 4.2. The grayscale model schematic.

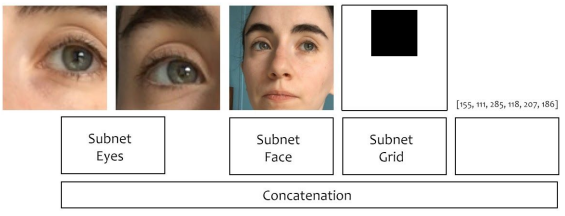


Figure 4.3. The nose-vector model schematic.

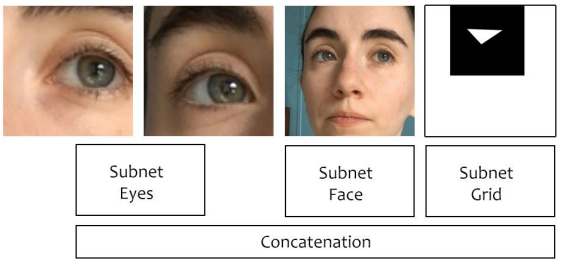


Figure 4.4. The nose-grid model schematic.

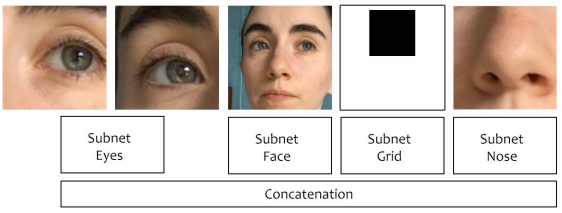


Figure 4.5. The nose-crop model schematic.

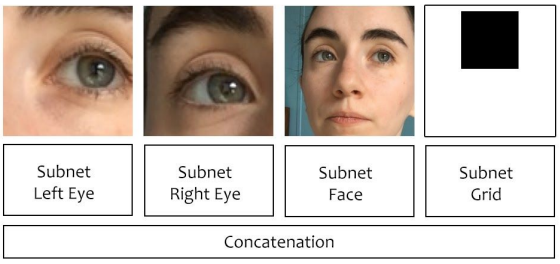


Figure 4.6. The eyes model schematic.

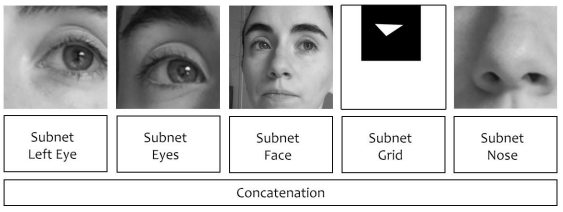


Figure 4.7. The integrated model schematic.

4.2. Results Table

Table 4.1. The results of the thesis's experiments and tests.

	Paper (Miniature)	Grayscale Model	Nose-Vector Model	Nose-Grid Model	Nose-Crop Model	Eyes Model	Integrated Model
MSE (cm^2)	22.1859	22.5243	22.9354	21.4305	22.0264	22.1384	**21.1914**
MAE (cm)	3.4152	3.4419	3.4973	3.3473	3.3991	3.3904	**3.3222**

Chapter 5

Discussion

5.1. Our Work

As we see in the results table (Table 4.1), except for the nose-vector test and the grayscale experiment, all investigated cases showed improvement in terms of accuracy. While the integrated model achieved the best result, the nose-grid test had the best accuracy among the thesis's individual experiments and tests.

Nose-vector test. The relative failure of the nose-vector test tells us that the model could not find a mapping from the mere numerical values of the eyes-nose coordinates to the gaze location, and still needs to have them with a reference to the face box rather than the frame. Therefore, from another perspective, it indicates that for gaze tracking the location of the face within the frame is less expressive than the location of the nose-eyes within the face box.

Grayscale experiment. The grayscale experiment also did not improve the original iTracker settings. However, we have considered it as a success and added the idea to the integrated model. The reason is that the goal of the grayscale experiment was to reduce the data size, not to improve the accuracy. And the fact that it has a competitive performance while we shrunk the data content to one-third of its original size *is* a success on its own. Especially considering the size of the parent dataset, GazeCapture, converting the images to grayscale would be an efficient method for saving disk space. Because usually training models on such huge datasets needs the researchers to upload the data to either a cloud or a grid computing space.

Small scale improvements. Even our best result, which is for the integrated model, improved the error of the original iTracker model only by 1 millimeter. From a gaze tracking perspective, that is insignificant. But the thesis's goal was rather to improve the neural network performance in the field of gaze tracking than to raise the benchmarks. With this perspective, having a stable reduction in error terms, even as small as 1 millimeter, is an improvement. For example, in the "Eye Tracking for Everyone" paper [15], in table 2, the authors reported the

iTracker error for the smartphones without data augmentation 2.04 centimeters. The same metric for the iTracker with a 25-fold augmentation of the train and the test sets, and after fine-tuning for each device and each orientation is 1.71. That is, the authors augmented 96% of about 1.5 million data samples and also adjusted the parameters for 60 different device-orientations only to achieve 3.3 millimeters of improvement. On the other hand, we trained our models on not only a smaller number of samples (less than 1% of the GazeCapture), but also on smaller samples (crops in the miniature dataset cover only about 8% of the pixels in the GazeCapture crops). We believe training the models on the complete dataset, with the original crop size, will improve the accuracy and stress the error differences. According to an article printed in the *MIT News* [10], after publishing their paper, Khosla et al. acquired data from 700 more subjects which adding them to the GazeCapture dataset decreased the error to about 1 centimeter[4]. The article also mentions an experiment by Khosla et al. on the effect of the data size, in which they have found out adding each 10,000 frames increases the model accuracy by 0.5 centimeter (that means 0.35 centimeter improvements in the thesis's error metric). We believe training the integrated model on the complete GazeCapture dataset with the original sample sizes and data augmentation will demonstrate the advantages of our modifications in full.

5.2. Khosla et al.'s Work

iTracker accuracy report. Khosla et al. failed to report a single decisive error value that would reflect the success of the model and the training parameters described in the paper. The best results that they have published are for "fine-tuned" settings for each of 15 devices (8 models of iPhone and 7 models of iPad) and each of 4 orientations, without any practical details about the adjustments. Yet they have calculated the error for smartphones and tablets separately, even though the training set covers samples from all the 15 available devices[5]. Conditional reporting of the error, along with the ambiguity of the detailed settings of specialized training, not only made a loyal replication of the results impossible, but it also will overshadow any efforts for comparing fine improvements.

[4] The article did not mention what error.

[5] The validation and test sets combinations are 8-3 and 7-6 for the iPhones and the iPads, respectively.

iTracker generalization. Khosla et al. seem to be shifting from a generalization disposition at first and while defining the project, towards a specification one at the end, when they are reporting their results. They have started by collecting a huge dataset, which is diverse in many aspects, including environment, background, lighting condition, head pose, age, and race. They also defined the iTracker goal to be capable of running on any mobile devices. But after training the iTracker, they decided to report its performance in a case-specific manner. Mainly because of the fact that they have found out that the error rate and uncertainty in tablets, or devices with larger screens and longer user-screen distances, is considerably higher than smartphones with smaller screens and shorter user-screen distances. Therefore, they first chose to report the error for smartphones and tablets separately, and then they went on to calculate the error for each single device alone to get better results. They also realized the different behavior of the images with different orientations so they narrowed their accuracy report by specific training for each orientation. Furthermore, by appealing to the practical application of a gaze tracking device, they reported one even more specific version of the iTracker accuracy by calculating and averaging the error over dots with same locations. Thereby, they gradually lost the validity of their initial claim to generality of the iTracker. That causes two problems. First, walking down this way, we can achieve even better accuracy by calculating the error for specific lighting conditions, for specific range of user-screen distance, and also specifically for age ranges or races. It is not an efficient practice to draw an arbitrary line somewhere in the middle and set it as the border of generality against specificity. And secondly, setting all these specifications and conditions reduces the iTracker chances for getting developed as a practical application available to "everyone."

5.3. Future Work

Dataset size. Predicting the gaze location, which on a 2 dimensional screen is a set of 2 real values, is a regression task. Employing CNNs, which usually play the role of a classifier, for a regression task is an almost recent trend. Consequently, there is not enough, if any, literature and research available on the appropriate size of the dataset for a regression task with CNNs. However, the speculations all depend on the size of the range of possible predictions. Coordinates, per se, can take any real values and even in a limited space the set of possible

coordinates is infinite. Nonetheless, since the gaze coordinate, supposedly, reflects a point on a screen, we can consider the prediction space to be discrete, as it is a finite set of pixels. Especially in our case where the set of predictions is finite and as small as about 71,000 coordinates. Based on that, we may be able to apply the dataset size rules of the classification.

Having an estimation for a fitting size of the dataset is important for the network to learn, and for the network not to overfit. But when it comes to a regression task, there is another necessity for the estimation. In classic regression methods, the only concern about the oversized dataset is overfitting to noise. Thus, assuming there is no noise in the data, the bigger the dataset the more accurate the prediction. But we should not apply the same formula to the regression tasks in machine learning. With a noise-free dataset assumption, having more than enough samples in the data does make the model predictions more accurate, but at the expense of making it less intelligent. The magic of deep learning lies in its capacity for *generalizing* the task-relevant patterns in the seen samples to those unseen. That is also what makes it *efficient*. In a gaze tracking context, in the pictures of people with different facial morphology, with different head poses, under different lighting conditions, still the location of the irises (for tracking the eyes) and the location of the nose tip (for estimation of the head pose) within the face box, are the most expressive and relevant features of the data samples. And these features could not have a wildly broad range of variety within a mobile device picture frame. Therefore, continuously adding more frames to the dataset may risk transforming an artificially intelligent model to a simple look-up table.

MTCNN augmentation. As we mentioned earlier, the Apple face and eye detectors tagged almost 40% of the GazeCapture dataset (954,545 out of 2,445,504 frames) as invalid. That is, iOS built-in detectors could not recognize a whole face or the eyes in the picture. For example, the folder *00358* of the GazeCapture dataset includes 3,487 frames from which almost half of them (1508 frames) are invalid. As we discussed the iOS poor technology for face and eye detection, many of these frames captured the full face with clear eyes. However, there are also invalid frames with worse occlusions and critical lighting conditions. We tested MTCNN on some of these frames, and it succeeded in detecting the face box with all the five facial landmarks in it. That opens an opportunity for improving the GazeCapture dataset by recycling the extravagantly invalid-labeled frames. But that goes well beyond the iTracker project. The GazeCapture dataset has great potential, and combining this precious dataset with the MTCNN

detection metadata will create a great opportunity for researchers in the field of face detection and computer vision in general.

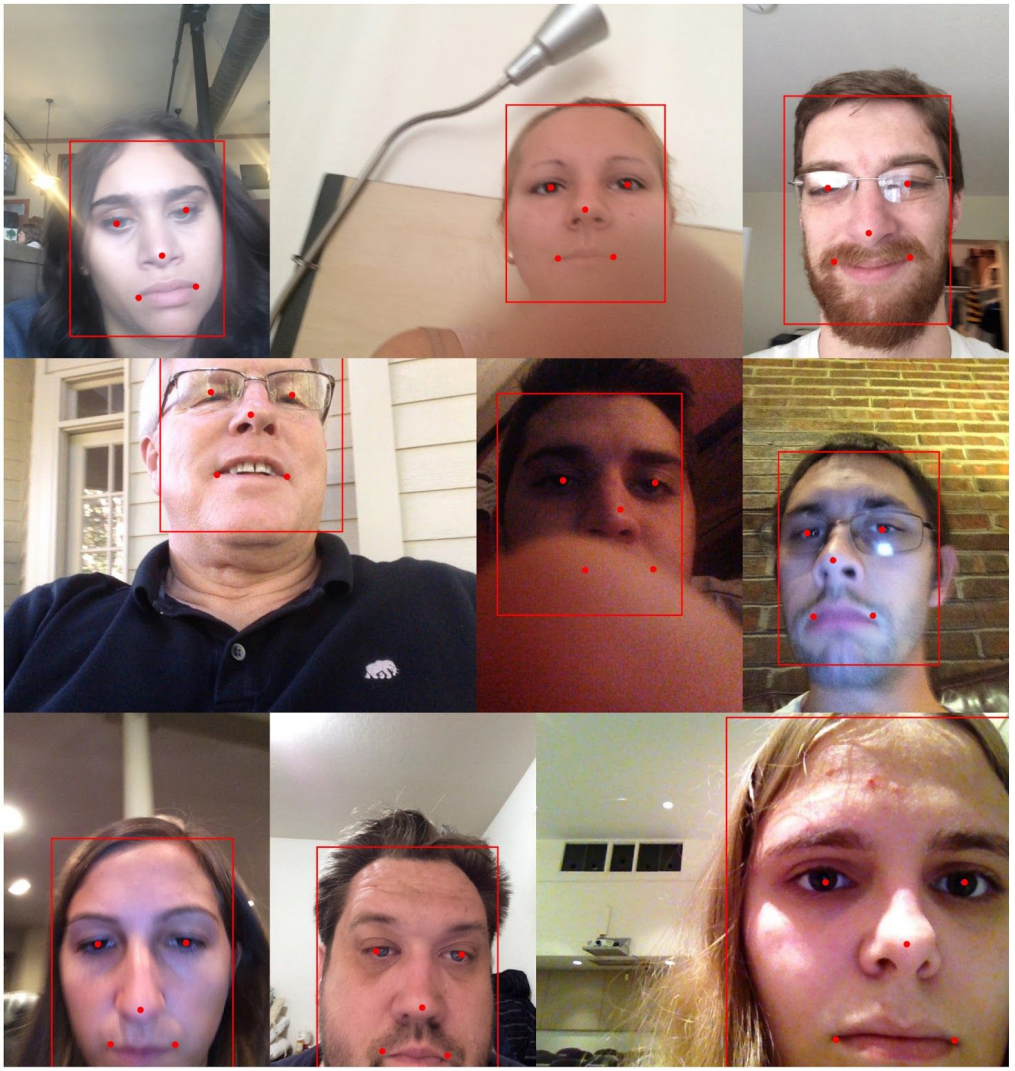


Figure 5.1. Samples of invalid frames in the GazeCapture with successful MTCNN detections, marked with red lines and dots.

Preserving the ratios. As it is evident in the Fig. 5.1, the dimensions of the MTCNN's face box is proportional to the subject's face ratio. According to Kannan, H. D. [13], keeping the original frames ratios will improve the accuracy of the model (although on a 0.1 millimeter scale). We believe with the MTCNN preserved face box ratios, the improvement will be substantial, especially if we combine the face grid with the face crop, while keeping both frame and the face ratios. One possible setting could be zero padding the face crop, so to make sure the non-face pixels will not affect the weights update.

Figure 5.2. Keeping both individuals' face and the frame ratios while combining the face grid and crop by zero-padding.

Uniform orientations. Khosla et al. reported their best results by fine-tuning the iTracker for each orientation. The coordinates of the fixation points are with respect to the center of the front facing camera, which is also the image perspective *vanishing point*. Therefore, the fixation point coordinates (in magnitude), the head pose, and the pupils locations are rotation invariance, when the center of rotation is the camera. With this assumption, we can rotate all frames into one orientation to prevent the necessity of different treatments of the frames with the different orientation. It may sound counterintuitive to estimating a person's gaze location by looking at an upside-down image of the face, however, with the absolute coordinates value, the network does not care much about the reference ground in the images.

Bibliography

[1] Amazon Mechanical Turk. (2015). *Amazon Mechanical Turk*. Retrieved from https://www.mturk.com.

[2] Atienza, R., & Zelinsky, A. (2002). Active gaze tracking for human-robot interaction. In Proceedings. Fourth IEEE International Conference on Multimodal Interfaces (pp. 261-266). IEEE.

[3] Baluja, S., & Pomerleau, D. (1994). Non-intrusive gaze tracking using artificial neural networks. In Advances in Neural Information Processing Systems (pp. 753-760).

[4] Boraston, Z., & Blakemore, S. J. (2007). The application of eye-tracking technology in the study of autism. The Journal of physiology, 581(3), 893-898.

[5] Chennamma, H. R., & Yuan, X. (2013). A survey on eye-gaze tracking techniques. arXiv preprint arXiv:1312.6410.

[6] Chollet, F. (2015, December 9). Remove LRN2D layer. *Keras Github Repository*. Retrieved from https://github.com/keras-team/keras/commit/82353da4dc66bc702a74c6c233f3e16b7682f9e6 .

[7] Computer Vision Machine Learning Team. (2017, November). An On-device Deep Neural Network for Face Detection. *Machine Learning Research*. Retrieved from https://machinelearning.apple.com/research/face-detection.

[8] Deans, P., O'Laughlin, L., Brubaker, B., Gay, N., & Krug, D. (2010). Use of eye movement tracking in the differential diagnosis of attention deficit hyperactivity disorder (ADHD) and reading disability. Psychology, 1(04), 238.

[9] Duchowski, A. T., & Duchowski, A. T. (2017). Eye tracking methodology: Theory and practice. Springer.

[10] Hardesty, L. (2016, June 15). Eye-tracking system uses ordinary cellphone camera. *MIT News*. Retrieved from https://news.mit.edu/2016/eye-tracking-system-uses-ordinary-cellphone-camera-0616.

[11] Holzman, P. S., Proctor, L. R., Levy, D. L., Yasillo, N. J., Meltzer, H. Y., & Hurt, S. W. (1974). Eye-tracking dysfunctions in schizophrenic patients and their relatives. Archives of general psychiatry, 31(2), 143-151.

[12] Jia Y., Shelhamer E., Donahue J., Karayev S., Long J., Girshick R., Guadarrama S., & Darrell, T. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

[13] Kannan, H. D. (2017). Eye tracking for the iPhone using deep learning (Doctoral dissertation, Massachusetts Institute of Technology).

[14] Khosla, A., Krafka K., Kellnhofer P., Beavers J. (2020, July 9). *GazeCapture Github Repository*. Retrieved from https://github.com/CSAILVision/GazeCapture.

[15] Khosla, A., Krafka, K., Kellnhofer, P., Kannan, H., Bhandarkar, S., Matusik, W., & Torralba, A. (2016). Eye tracking for everyone. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2176-2184).

[16] Kim, K. N., & Ramakrishna, R. S. (1999). Vision-based eye-gaze tracking for human computer interface. In IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028) (Vol. 2, pp. 324-329). IEEE.

[17] Krafka, K. (2017, March 5). cm2pts.m. *GazeCapture Github Repository*. Retrieved from https://github.com/CSAILVision/GazeCapture/blob/master/code/cm2pts.m.

[18] Krafka, K. (2017, March 5). pts2cm.m. *GazeCapture Github Repository*. Retrieved from https://github.com/CSAILVision/GazeCapture/blob/master/code/pts2cm.m.

[19] Krafka, K. (2017, March 5). screen2cam.m. *GazeCapture Github Repository*. Retrieved from https://github.com/CSAILVision/GazeCapture/blob/master/code/screen2cam.m.

[20] Mishra, A., & Bhattacharyya, P. (2018). Cognitively Inspired Natural Language Processing: An Investigation Based on Eye-tracking. Springer.

[21] Molitor, R. J., Ko, P. C., & Ally, B. A. (2015). Eye movements in Alzheimer's disease. Journal of Alzheimer's disease, 44(1), 1-12.

[22] Murphy-Chutorian, E., & Trivedi, M. M. (2008). Head pose estimation in computer vision: A survey. IEEE transactions on pattern analysis and machine intelligence, 31(4), 607-626.

[23] Niknam, S. (2020, September 22). *ETE Github Repository*. Retrieved from https://github.com/shrnkm/ETE.

[24] de Paz Centeno, I. (2020, June 8). *mtcnn Github Repository*. Retrieved from https://github.com/ipazc/mtcnn.

[25] PixelCut. (2020). The Ultimate Guide To iPhone Resolutions. *PaintCode*. Retrieved from https://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions.

[26] Sharma, S. (2017, September 23). Epoch vs batch size vs iterations. *Towards Data Science*. Retrieved from https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[27] TensorFlow Developer Team. (2020, September 25). tf.keras.layers.Conv2D. *TensorFlow*. Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.

[28] Torch Contributors. (2019). Class CrossmapLRN2D. *PyTorch*. Retrieved from https://pytorch.org/cppdocs/api/classtorch_1_1nn_1_1functions_1_1_cross_map_l_r_n2d.html.

[29] Wedel, M., & Pieters, R. (2008). Eye tracking for visual marketing. Now Publishers Inc.

[30] Xu, P., Ehinger, K. A., Zhang, Y., Finkelstein, A., Kulkarni, S. R., & Xiao, J. (2015). Turkergaze: Crowdsourcing saliency with webcam based eye tracking. arXiv preprint arXiv:1504.06755.

[31] Yang, F., Jiang, Z., Wang, C., Dai, Y., Jia, Z., & Hirota, K. (2018). Student Eye Gaze Tracking During MOOC Teaching. In 2018 Joint 10th International Conference on Soft

Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS) (pp. 875-880). IEEE.

[32] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10), 1499-1503.