UNIVERSITÄT OSNABRÜCK

Using Convolutional Neural Network for Development of an Eye-Tracking Application, Capable of Running on the 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

Acknowledgement

Table of Contents

List of Figures

1. Introduction

2. “Eye Tracking for Everyone”

Khosla et al. (2016) tried to realize a software-based eye tracking method, using the ever-growing potentials of neural networks. The authors of the paper, “Eye Tracking for Everyone,” (Khosla, 2016) calims that their eye tracking model, the iTracker, which they trained on a large-scale specialized dataset for eye tracking, has an equally satisfactory performance as the current hardware-based devices. The Author gathered and augmented the dataset especially for their experiment. However, they also tested their model on other available datasets and achieved state-of-the-art results. And considering the fact that all these devices work with sensitive equipment, and therefore their usage is costly and usually needs expert’s attendance, one can consider the development of a software eye tracker with an equal 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.

2.1. GazeCapture

For an efficient training of a neural network, the high variability of the dataset is essential. To meet this criterion, Khosla et al. decided to build their own dataset instead of using the available ones in the field which are all small, and not sufficiently diverse. One of the undesired, but inherent, characteristics of the lab-produced eye tracking 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 lightning 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 always 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 in building the dataset from their own residence, the diversity of the backgrounds, lightning conditions, and the head poses is tremendous (e.g. some subjects recorded themselves while lying in their beds).

Achieving this goal, the authors wrote an iOS application, called *GazeCapture* [*], and recruit subjects through the Amazon Mechanical Turk [*] which is a global crowdsourcing platform. The application displays 13 fixed locations black dots with pulsing red circles around (to draw attention to the dots) on the device screen for 2 seconds, and takes pictures of the subject, using the front-facing camera, about 0.5 second after appearance of the dots. Furthermore, the authors instructed the subjects to change the orientation of the device, every 60 dots, to add to the variability of the dataset even more.

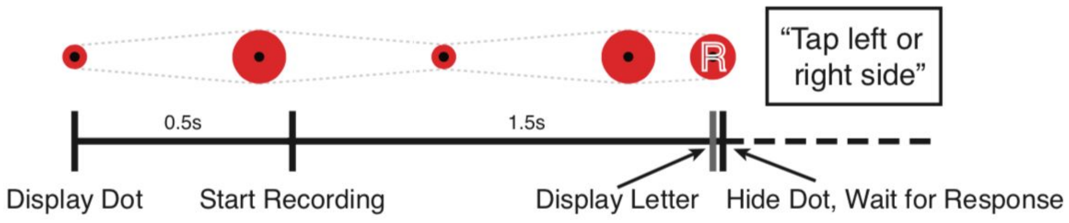

Figure *: "The timeline of the display of an individual dot. Dotted gray lines indicate how the dot changes size over time to keep attention" (Khosla et al., 2016).

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 not looking at the dot, and secondly, the subject's eyes not being in the captured frame. 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 displayed one of the two letters *L* and *R* to the pulsing dots for about 0.05 second, shortly before they disappeared. Based on the displayed letter, subjects had to tap on either the left or the right side of their devices' screen. And failing to do so, the subjects needed to repeat that dot recording.

2.2. Dataset

2.2.1. Images

The GazeCapture dataset consists of 1474 folders, and according to Khosla et al., each contains recorded frames of one individual. The dataset includes 2,445,504 frames, both in portrait and landscape orientations, with the dimensions of 640 x 480 pixels with 3 RGB color channels. That makes an average of 1659 frames per subject, with a range varying from 4 to 3590 frames in a folder. Every folder also contains 9 files of metadata on the recorded frames in *json* format.

Figure *: 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, left eye, and the right eye for each frame. The *IsValid* key takes either 1 or 0 as value, indicating whether the frame properly captured the face, left eye, or the right eye or not, respectively.

The faceGrid.json is also a dictionary with the same keys giving the lists of X and Y coordinates of the top left corner, and the width and the height of the face box within a canvas, representing the rescaled version of frames to a 25 × 25 square. The *IsValid* value is either 1, that is the face crop lies mostly within the frame, or 0 meaning that it does not.

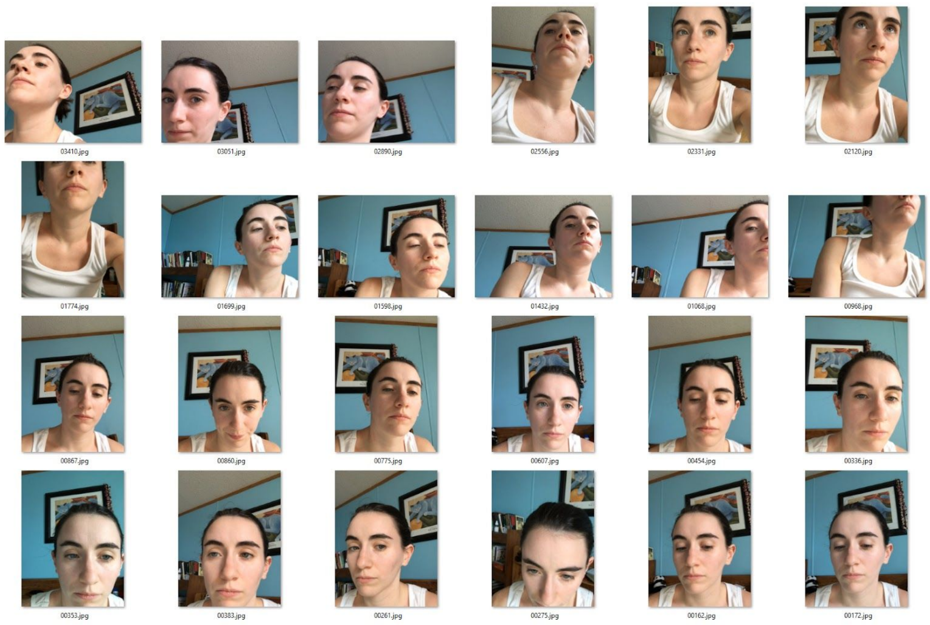

Figure *: Sample frames from one subject.

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. And the XPts and YPts are the location of the center of the dots, in points, with respect to the top left corner of the screen. While XCam and YCam are supposedly giving the same information, except in centimeter (cm) and with respect to the center of the front facing camera of the device. The Time is a list of time spans in second, between the display of a dot and the capture of the corresponding frame.

The frames.json listed the names of all the frames in the folder.

The motion.json is another 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 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, a small dictionary, 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 that the authors used to train their neural network for reporting the results in their paper, and finally the *DeviceName* that includes 7 models of iPad and 8 models of iPhone.

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* that 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.

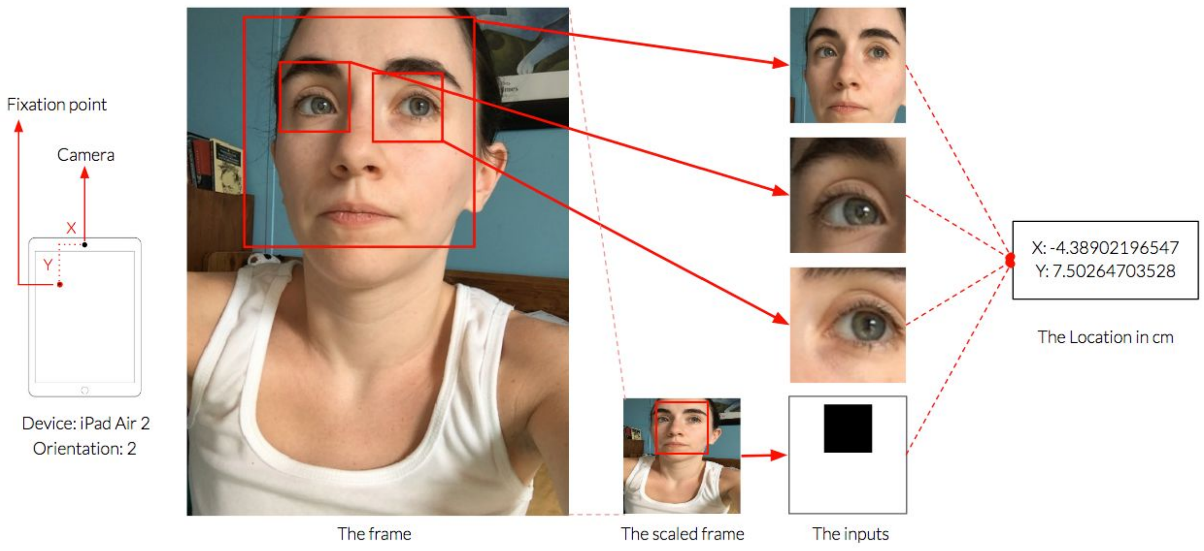

Figure *: A sample frame metadata.

2.2.3. Issues

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

Inaccurate count of the subjects. The authors reported that they have 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 5,000 of these folder revealed that the frames taken from

2.3. iTracker Model

3. Improvements

3.1. Model

3.2. Dataset

3.3. Experiment 1: Grayscale Images

In this scenario, each frame was converted to a grayscale image. The motivation behind this scenario was to reduce the computational workload, since dropping the color channels will downsize the dataset to one third. The conversion was done using a method of the *OpenCV* Python library: `cv2.COLOR_BGR2GRAY`.

3.4. Experiment 2: Eyes' Networks

3.5. Experiment 3: MTCNN

3.5.1. Grid

Multi-Task Cascaded Convolutional Network (MTCNN) is a lightweight CNN for face detection and alignment in real time which is claimed to have superior accuracy over the state-of-the-arts methods and algorithms. The core idea of MTCNN is to run a CNN on a cascade of scaled versions of the original image, and it was introduced in a paper by Zhang,

Zhang, Li, and Qiao (2016). In 2017, the Github user, *ipacz* [*], wrote a Python (pip) library based on the MTCNN idea which is now available to install and to use 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's value is a list of four values of the X and Y coordinates of the top left corner, and the width and the height of the face box (in pixels). The *confidence* is the reliability of the detection (in percent). And the *keypoints* is a nested dictionary of 5 facial landmarks coordinates, including the approximate center of the left and the right eyes, the nasal tip, and the left and the right corners of the mouth.

```
!pip install mtcnn
from mtcnn.mtcnn import MTCNN
faces = MTCNN().detect_faces(img)
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 *. Install, import, and running MTCNN

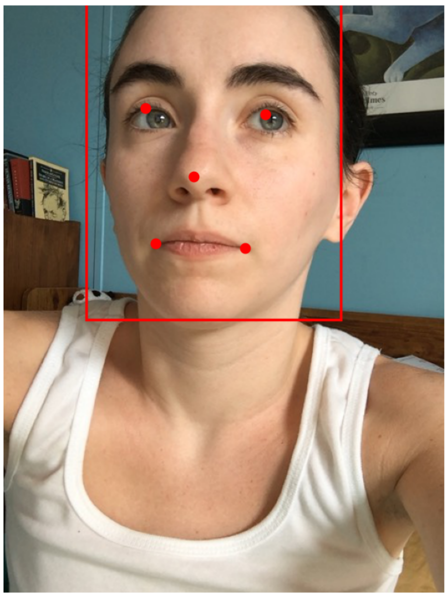

Figure *: MTCNN results plotted with red box and dots.

In MTCNN experiment ideand 5 facial detection which was released in 2017. Unlike the regular CNNs, an MTCNN recognizes face in an image

3.5.2. Vector

3.6. Experiment 4: Nose crop

4. Results

5. Conclusion

Bibliography

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.

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.

https://www.mturk.com/

https://apps.apple.com/us/app/gazecapture/id1025021075

https://github.com/CSAILVision/GazeCapture