Building Real-Time Unconstrained Eye Tracking with Deep Learning

by

Kyle John Krafka

(Under the Direction of Suchendra Bhandarkar and Walter Potter)

Abstract

Eye tracking is an important tool with applications in many domains. The ability to measure where a person is looking can be used for psychological studies, medical diagnoses, and human–computer interaction techniques. Existing high-accuracy solutions require custom and expensive hardware, limiting their reach. A more accurate low-cost solution is proposed to bring real-time unconstrained eye tracking to mobile devices.

A deep convolutional neural network is used to determine gaze using only an image acquired from the front-facing camera found on modern phones and tablets. As with most deep learning approaches, the model requires a large volume of training data to perform well. Existing gaze datasets suffer from too few subjects or too little variety. This is largely due to the difficulty in conducting such experiments on a large scale. To overcome this data limitation, a crowdsourcing technique is introduced along with an unprecedentedly-large dataset, both in terms of the number of subjects and in variability.

With this dataset, the trained model achieves state-of-the-art accuracy by a significant margin. Furthermore, it is robust to various lighting conditions and different user poses. Through an extensive evaluation, a variety of approaches to further improve the model's accuracy are demonstrated. Finally, to enable real-world mobile application of our model, the computation time and memory usage are optimized while maintaining high accuracy.

This end-to-end design of an eye tracking system represents how modern computer vision and machine learning techniques can be used to make significant progress in appearance-based problems. These novel contributions represent a significant leap forward for eye tracking and should better equip the next generation of researchers and innovators.

INDEX WORDS: Gaze estimation, Eye tracking, Deep learning, Big data, Crowdsourcing, Computer vision, Computer science

Building Real-Time Unconstrained Eye Tracking with Deep Learning

by

Kyle John Krafka

B.S., University of Georgia, 2010

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2015

Building Real-Time Unconstrained Eye Tracking with Deep Learning

by

Kyle John Krafka

Approved:

Major Professors:	Suchendra Bhandarkar Walter Potter
Committee:	Hamid Arabnia Kyle Johnsen

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
December 2015

Building Real-Time Unconstrained Eye Tracking with Deep Learning

Kyle John Krafka

December 2015

For Mom and Dad

&

For Dr. and Mrs. Larry Baker

Acknowledgments

As I reflect on this body of work, my graduate career, and all nine-and-a-half years that I spent at The University of Georgia, I am filled with a deep sense of gratitude. I have grown tremendously during my time here, both academically and personally. While I can only mention a few select groups of people here, I am indebted to so many for the ways they have influenced me.

First, I would like to thank my committee. Dr. Suchendra Bhandarkar: I appreciate you always pushing me to deliver high-quality research, giving me freedom to explore different research topics, and for the advice all along the way. I have you to thank for getting me started in research related to eye tracking and I have thoroughly enjoyed working in your lab! Dr. Walter Potter: Thanks for taking me under your wing as an undergraduate, helping me get my first publication on "hunting snakes," and for sticking with me through my PhD. I've always valued your candid advice along the way. Dr. Hamid Arabnia: I've enjoyed working with you ever since I was your grader for Data Structures. Thanks for giving me my computer vision roots in your excellent image processing class and for always being willing to lend an encouraging hand. Dr. Kyle Johnsen: Thanks for your mentorship ever since joining my committee. I've always appreciated the advice

you bring to the table and your willingness to help.

I would like to offer a huge "thanks" to my collaborators. First and foremost, Aditya Khosla: You have been nothing short of essential to this work and to my abilities as a researcher. I have learned so much from you and your direction has been key to a successful graduate career. Whether cracking a bad joke or sharing from your endless fountain of ideas, I have thoroughly enjoyed working with you.

Petr Kellnhofer: I appreciate your help on this project. You are one of the best programmers I know and I have enjoyed getting to know you through this collaboration. Dr. Raffay Hamid: Although we didn't end up publishing on our project, I learned a great deal from you that I have been able to apply to this dissertation work. I am thankful to have learned to research under your tutelage. Also, thanks to Dr. Tilke Judd for your willingness to help get me connected and to Dr. Antonio Torralba and Dr. Wojciech Matusik for your support of this project, both academically and financially. You were all instrumental in making this a work of which I can be very proud.

Next, I would like to thank the other faculty and staff in the Department of Computer Science and the Institute for Artificial Intelligence. Whether trying to guess the scent of Suzi Sanders' candle, ringing the System Support doorbell, or enjoying some jalapeño pizza at the departmental social hour (organized by Dr. Khaled Rasheed), I felt right at home in Boyd. Everyone made me feel valued as a student and I knew I could always count on the faculty to take time out of their busy schedules if I had a problem. The department really felt like family.

Thank you to the different university entities that made this research possible. I appreciate Kim Fowler and the other OVPR staff that went out of their way to help me with

the IRB paperwork. Also, thanks to the Graduate School for printing posters and always answering my questions.

To my friends who stuck with me through thick and thin: I couldn't have done this without you. To my labmates past and present (Somenath Das, Arun Kumar C.S., Karan Sharma, Anirban Mukhopadhyay, Srijita Chakraburty, and Jagadish Kumar Mahendran): Thanks for all of the laughs and educational moments. Thanks to Dr. Brad Barnes and Michael Scott for keeping me active—without the bike rides, swims, and runs, I might have gone insane! Thanks to Karen Aguar, Michael Cotterell, Ugur Kursuncu, and all of the other wonderful TAs (too many to name) for sharing your passion for teaching with me. Thanks to the fifth floor folks, everyone in the Institute for Artificial Intelligence, classmates, and other graduate students for being a special part of this adventure. Of course, thanks to Jack Walker, Cody Corbett, Andrew Fowler, Jonathan Foss, Kaley Krafka (my sister), and everyone else who has been a great friend throughout graduate school.

Thanks to everyone in the community for making Athens such a great place to live. I have experienced some great traditions and Athens classics during my time here, including AthHalf, Twilight, Jittery Joe's, and Barberitos. Also, thanks to my church family for being some of the most genuinely-loving people I know.

I owe a special "thanks" to Kathleen. From our first date at the Computer Science Christmas party (my first year in graduate school) to a nice dinner in my lab as I stayed late to work on my dissertation, you've been there through it all; patient, understanding, and supportive. I can't imagine a better companion to have by my side.

Finally, this dissertation is dedicated to two couples. First, thank you Mom (Jan

Krafka) and Dad (John Krafka) for teaching me everything I know about computer science. Okay, maybe that's not true, but I do owe it all to you for raising me the way you did—to always shoot high, never give up, and to enjoy the great things this life has to offer. Second, thank you Dr. and Mrs. Larry Baker for giving me my first job, working on computers. I always looked up to Dr. Baker's academic career and the intimate role Mrs. Baker played in it. Thanks for being such supportive and loving people. It is for these reasons that I dedicate my dissertation to you.

Now, as I look forward to the next chapter of my life, one thing is clear: I would not be where I am, or who I am today without each and every person mentioned here. To have you in my life has been nothing short of a blessing. You have shaped me profoundly and I will forever be thankful for that.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Eye tracking is the process that measures where a person is looking. These measurements of *gaze*, or *point of regard* are useful because they provide a window into the mind. Gaze is the externally-observable indicator of human visual attention. In many studies, gaze samples are the building blocks that enable the physiological study of the human visual system.

Eye tracking (or more precisely, *gaze estimation*) has applications in many domains [Duchowski, 2007, 2002], from human–computer interaction techniques [Jacob and Karn, 2003; Majaranta and Bulling, 2014; Morimoto and Mimica, 2005] to medical diagnoses [Holzman et al., 1974] to psychological studies [Rayner, 1998] to computer vision [Borji and Itti, 2013; Karthikeyan et al., 2013]. As we will explore in Section 2.1, researchers have worked to record eye movements since the late eighteenth century [Huey, 1908].

Today, a variety of solutions exist (many of them commercial) but all suffer from one or more of the following: high cost (e.g., Tobii X2-60), custom or invasive hardware

(e.g., Eye Tribe, Tobii EyeX), or inaccuracy under real-world conditions (e.g., Mora et al. [2014]; Sugano et al. [2014]; Zhang et al. [2015]). These factors prevent eye tracking from becoming a pervasive technology that should be available to anyone with a reasonable camera (e.g., a smartphone or a webcam). In this work, our goal is to overcome these challenges to bring eye tracking to everyone.

1.1 Overview of Our Solution

We focus on implementing eye tracking for mobile devices currently on the market. In this way, we inherently minimize the cost, specialty, and invasiveness of the hardware. This makes for a highly accessible and usable solution with one significant problem to overcome: inaccuracy. As we will describe in this section (and validate in Chapter 5), inaccuracy can be overcome by employing a powerful model and using the right data to train it. The big-picture contribution of this dissertation is a novel dataset (*Mobileyes*, collected by our crowdsourcing app, *GazeCapture*) and a model (*iTracker*) which are used together to achieve an unprecedented level of accuracy for unconstrained camera-based eye tracking.

1.1.1 Trends in Computer Vision

The primary sensor that enables eye tracking on mobile devices is the front-facing camera. The field of computer vision offers many techniques for the analysis of these RGB (i.e., red, green, and blue color channels) images. Recent advances in the field suggest that significant improvements can be made to the state of the art in gaze estimation.

In particular, *deep learning* has proven to be a very powerful tool across many domains in computer vision [Krizhevsky et al., 2012; Girshick et al., 2014; Taigman et al., 2014]. Deep learning describes a family of data-driven modeling tools derived from research in machine learning and artificial intelligence. We describe these models in more detail in Section 1.3, but what makes them unique is their ability to handle complex data on a large scale.

Historically, computer vision has relied on the creation of hand-crafted *features* to work with images. Raw image data is too high-dimensional and complex for most systems to work with directly, so the purpose of hand-crafted features is to describe the data in a more compact way. For example, consider the raw pixel values pictured in Figure 1.1. Computer models may miss the "big picture" amidst the many individual values. Furthermore, a seemingly-trivial change in image illumination could change all values. Instead, as an example of a human-crafted feature, the pictured image region could be described by single value describing a horizontal edge. In this way, dimensionality is reduced while still preserving important high-level information about the image.

The choice of representation is essential to being able to work with images. Human-crafted features are inherently limited as they rely on human innovation to be improved upon. With deep learning, features can be discovered automatically from large amounts of data. Furthermore, these features can be tuned to best work for the task at hand. For example, generally speaking, color is less important than the relationships between particular shapes for eye tracking. A significant contribution of this work is a learned feature representation that works well for eye tracking.

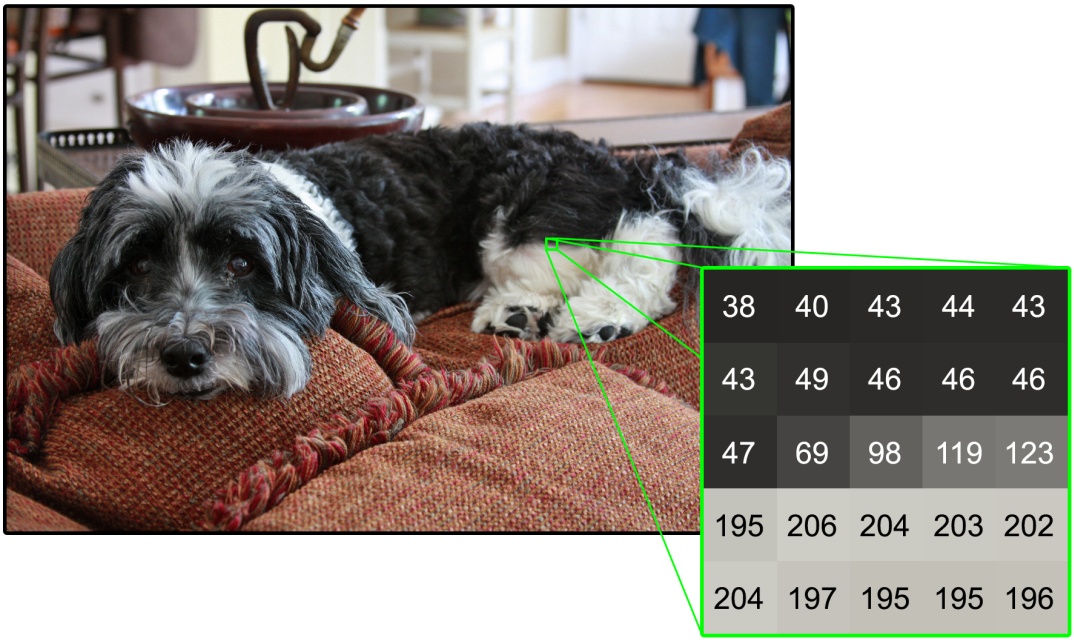


Figure 1.1: Raw pixel values (in the range 0–255) for a single color channel.

1.1.2 Model

We use a deep convolutional neural network (CNN) to process input images and to make a prediction of where the subject is looking. The CNN model is trained and evaluated with example images where the gaze location is known. Unlike similar related models, it does not rely on any preexisting systems for *head pose* (i.e., the position of the head in space) estimation or other manually-engineered features for prediction. We simply train the network with crops of both eyes, the face, and the position of the face.

Due to the hierarchical nature of the model, a gaze-specific feature representation is also learned in addition to the main prediction. We use this feature representation to

show how our model can generalize beyond our training and test data.

We perform a thorough baseline analysis to show how our model outperforms existing eye tracking approaches in this domain by a significant margin. Whereas existing approaches have mean prediction errors (i.e., the average distance between the predicted and the actual gaze point) of over 3 cm at best [Huang et al., 2015], we achieve 2 cm in an unconstrained setting. Furthermore, we show how calibration can significantly reduce this error further.

While our network achieves state-of-the-art performance in terms of accuracy (i.e., minimal mean prediction error), the size of the inputs and number of parameters make it difficult to use in real-time on a mobile device. To address this, we apply ideas from the work on *dark knowledge* by Hinton et al. [2015] to train a smaller and faster network. Through performance tests, we infer that this smaller model can achieves real-time performance on mobile devices with a minimal loss in accuracy.

1.1.3 Data

Our model is the product of careful design and experimentation; however, a key element of its success is the data we use to train it. Deep learning's application to eye tracking has been rather limited [Zhang et al., 2015]. We believe that this is due to the lack of availability of large-scale data, with the largest modern datasets having around 50 subjects [Huang et al., 2015; Sugano et al., 2014]. Compare this to the datasets of other deep learning successes. For example, Krizhevsky et al. [2012] used 1.2 million images to train the well-regarded and high-performing CNN used for image classification.

We overcome this problem by building our own dataset. As with many other re-

cent large-scale datasets requiring human interaction, we employ crowdsourcing. Our dataset, Mobileyes, was created by asking subjects to use an app that displays dots on the screen while video frames from the front-facing camera are saved.

With over 1000 subjects, our dataset is significantly larger than existing datasets (roughly, by a factor of 20). Furthermore, due to the collection methodology, it has more in-the-wild variety than existing datasets. We believe Mobileyes represents a significant contribution to the community and will have a lasting impact on the future of eye tracking.

1.1.4 Summary

We propose an end-to-end eye tracking solution using the front-facing camera on existing mobile devices. Inherently, this removes the barrier-to-entry present in many existing eye tracking systems, so we focus on reducing the inaccuracy, especially in real-world conditions.

As depicted in Figure 1.2, our end-to-end solution involves collecting an unprecedentedly-large dataset which we use to train a deep learning model. By combining the power of big data with the learning capability of deep learning, we achieve state-of-the-art performance by a significant margin. Overall, we take a significant step towards putting the power of eye tracking in everyone's palm.

1.2 Visual Attention

Gaze is only the "tip of the iceberg"—it is the externally-observable portion of the complex *human visual system*. This system has intrigued researchers for well over a century.

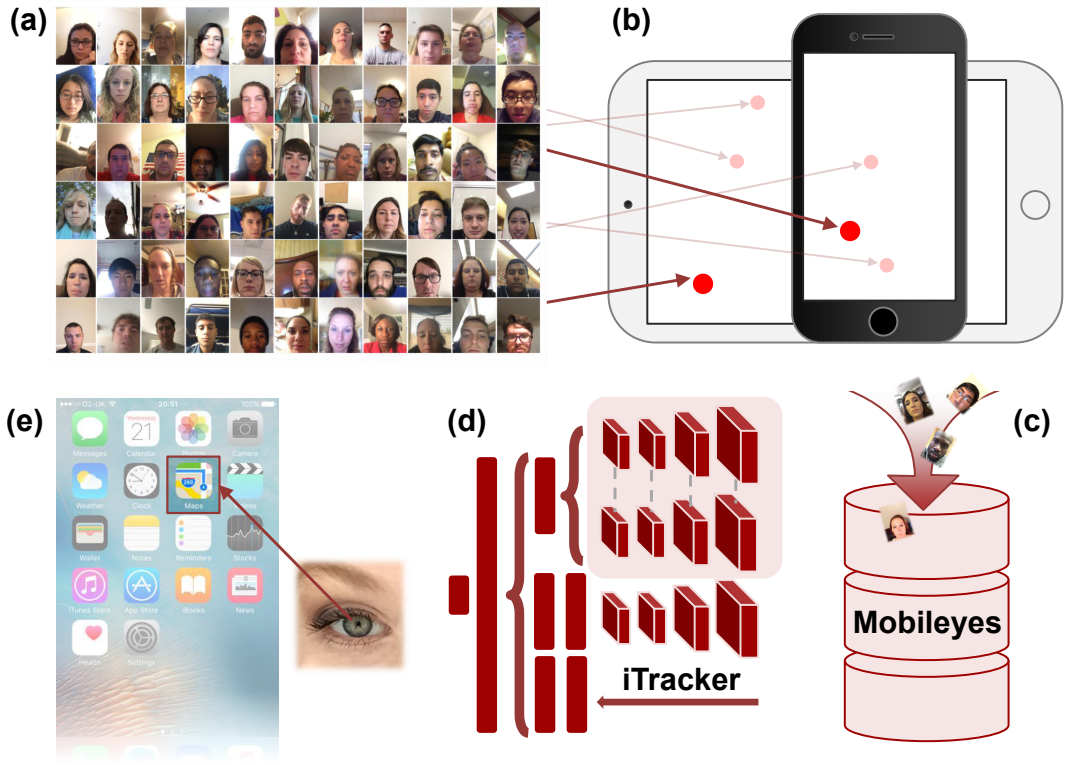


Figure 1.2: Utilizing a massive user pool (a) and our data gathering application (b) we have acquired a large dataset of face images with associated ground truth gaze locations (c) which enabled training of our eye tracking convolutional neural network model (d) and predicting gaze location using only the RGB image of a user's face (e).

By examining the physiological underpinnings of gaze, we will be able to better design and evaluate our experiments.

Imagine being able to read a page of a book without moving your eyes. Our eyes are certainly capable of "seeing" an entire page at once, but at any given time, we can only *attend to* a few words.

The reason for this is two-fold. First, the eye only receives high-resolution input at the center of the field of view; the farther from the center, the more blurry the image. This is known as *foveated imaging*. Second, on a related note, the human brain is not capable of processing much input in the first place.

The way the brain handles these limitations is through a process known as *selective attention*.

> Every one knows what attention is. It is the taking possession by the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought. Focalization, concentration, of consciousness are of its essence. It implies withdrawal from some things in order to deal effectively with others…
>
> — William James [1890]

The selective attention process is typically divided up into two parts. First, during the *preattentive process*, the brain considers the entire scene [Neisser, 1967]. This process is very quick and happens subconsciously. It determines the most *salient* points in the scene which guide the second part: the *attentive process*.

During the attentive process, the eyes move to *fixate* on a particular point in the scene for more detailed analysis. After a short period of time, the eye will *saccade* (French for

"jerk") to a new fixation point [Yarbus, 1967]. In this way, gaze is indicative of visual attention.

There are some caveats, though. For example, *fixational eye movements* are miniature eye movements around a particular fixation point [Engbert and Kliegl, 2003]. These movements happen naturally even if attention is fixed on a particular spot. Furthermore, attentional shifts may even happen without eye movement [Posner, 1980]. This is known as *covert attention*, while *overt attention* involves an associated eye fixation. Thus, gaze is inherently linked to the study of visual attention, but is not always a pure indicator of visual attention.

1.3 Introduction to Deep Learning

In machine learning and artificial intelligence, any learning or intelligence must take place inside of a well-defined model. For example, one such model (albeit naïve) could involve averaging daily rainfall from previous years and using that to predict the chance in the future. Indeed, this approach is unlikely to work well, but the predictions are likely to be better than random guesses, so we can say the model learned something from the training data.

1.3.1 Artificial Neural Networks

In this work, our model is based on *artificial neural networks* (or simply *neural networks*). They have their foundations in the 1940s [McCulloch and Pitts, 1943] and 1950s [Rosenblatt, 1958] and are loosely inspired by the human brain. In their most basic form, neural

networks are made up of "neurons" which take one or more numerical inputs and produce a single numerical output. The output is produced by taking a weighted sum of the inputs, then passing the result through some non-linear function. See Figure 1.3 for a common visual representation of a neuron.

More formally, we can specify the neuron's task mathematically:

$$y = f\left(\sum_i w_i x_i + b\right),$$

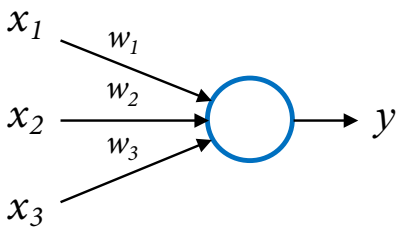


Figure 1.3: A single neuron in a neural network.

where $f(\cdot)$ is the *activation function*. This non-linear function is often the sigmoid function or the hyperbolic tangent function. These "squash" the outputs to be in the range between 0 and 1 or -1 and 1, respectively. This operation is biologically inspired and makes the model more powerful than a simple linear model. The b term above is not shown in the figure, but is a model parameter along with the weights. It is called the *bias term* or the *intercept term* and it influences the prediction independent of all inputs. For example, without the bias term, all 0 inputs would always produce a 0 output, which may not be desirable.

To understand the value of such a model, we can consider an example. Imagine we want to predict whether I will ride my bicycle today or not. Valuable inputs to use might be the probability of a friend joining (x_1), the likelihood of rain (x_2), and how much of my dissertation is complete (x_3). I might set w_1 to some positive value so that a friend joining positively affects the decision. w_2 should probably be negative so that a high chance of

rain has the inverse effect. Then, of course, w_3 should be very high to ensure that my dissertation gets finished.

In practice, many of these neurons are chained together to make predictions. See Figure 1.4 for an example. The outputs from some layers are the inputs to other layers. Note that neurons do not produce multiple outputs, but rather the same output may be copied to multiple inputs. Also, note that the weights and bias terms (i.e., parameters) are not displayed.

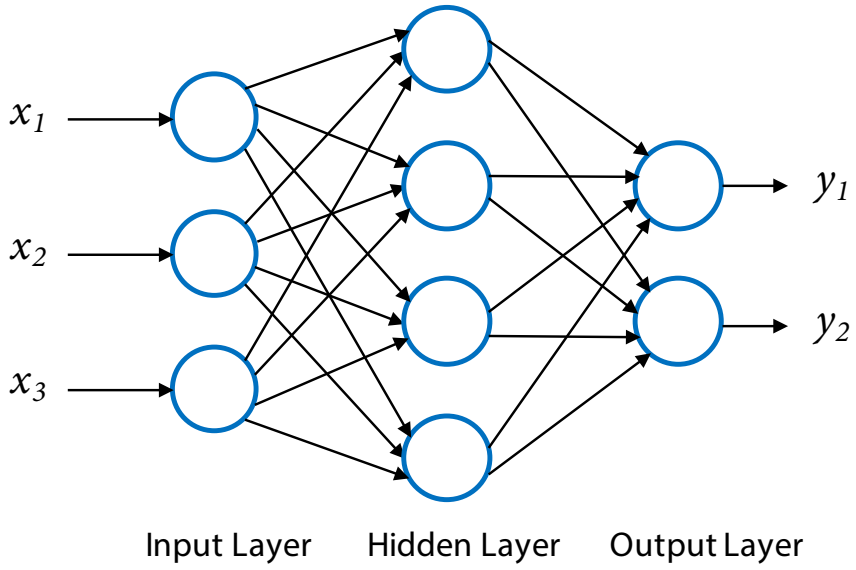


Figure 1.4: An example neural network with two final outputs.

Also shown in Figure 1.4, neurons are often grouped together in layers. The first layer is the "input layer," the last layer is the "output layer," and the others are called "hidden layers." Every neuron in one layer is connected to every neuron in the next layer. We say that these layers are "fully connected." This property allows the network the flexibility to determine which connections are most important on its own; the weight value for each

connection can strengthen or weaken the connection as needed.

1.3.2 Backpropagation

In the above bike-riding example, I selected the weights. Practically though, the weights are *learned* during the *training* phase, where sample inputs with known *ground truth* outputs are given to the network. During a *forward pass* through the network, each input gets an associated output (i.e., a prediction). A *loss function* compares the prediction with the ground truth to compute an error value (or a *loss*). This error value is then *backpropagated* through the network (in a *backward pass*) so that each weight is attributed some amount of responsibility for the overall error.

Then, typically through a process known as *stochastic gradient descent*, the weights are updated according to the partial derivative of the error function (relative to each particular weight). The partial derivatives are useful to determine how much error is changing respective to any given parameter—this information is useful in deciding exactly *how much* the weight should be adjusted and in which direction.

1.3.3 Deep Learning

Computers have become exponentially more powerful since the inception of neural networks. *Deep learning* is, in essence, the latest iteration of neural network research made possible by these computational improvements and some other essential theoretical breakthroughs. One key defining characteristic of *deep neural networks* is their size. In terms of layers, neurons, and parameters, deep neural networks are very large. For example, a typical deep model may have tens of millions of parameters! To deal with this size, graphical

processing units (GPUs) are often used because of their ability to perform many floating point mathematical operations in parallel.

1.3.4 Convolutional Neural Networks

In this work, we use a specialized deep neural network known as the *convolutional neural network* (CNN). It is one of the most frequently used deep learning techniques in use today and has proven to be a very effective model in the computer vision community. Specifically, it is well-suited to take images as inputs.

Convolutional neural networks are still very closely related to simple "shallow" neural networks; however, to effectively deal with 2D multi-channel (i.e., red, green, and blue) images, the formulation is more complex. By classifying and describing different types of layers (beyond just "input," "hidden," and "output"), these complex networks can be modularized.

Convolutional Layers

As the CNN's namesake, convolutional layers lie at the heart of what makes CNNs uniquely suited to work with images. Image convolution is a process based on a signal processing operation. Given an input image and a *convolutional kernel* (or *mask*, or *filter*), a modified output image is created.

Kernels are typically small, square 2D matrices. They typically have an odd number of values in both dimensions such that there is a single center value. The values in the kernel determine the effect when applied to the input image. To "convolve" a kernel with an image, a window the same size as the kernel is moved across the image to every pos-

sible position. The values from the window are multiplied by the associated values in the kernel, then summed to produce a single output value, which is placed in the output image at coordinates relative to the position of the window in the image. See Figure 1.5 for a visual example.

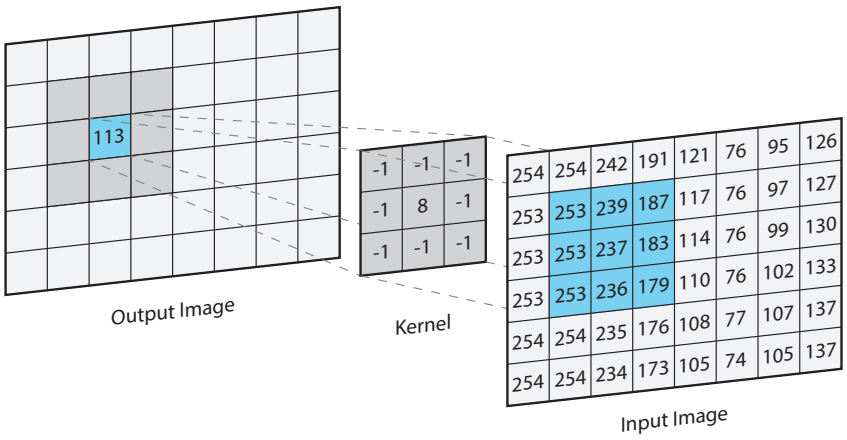


Figure 1.5: An example of image convolution. Each of the highlighted input pixel values are multiplied with their associated kernel values before being summed to a single pixel value in the output image. In this example, the output pixel value is computed as follows: $(253 * -1) + (239 * -1) + (187 * -1) + (253 * -1) + (237 * 8) + (183 * -1) + (253 * -1) + (236 * -1) + (179 * -1) = 113$ The operation pictured is repeated for every possible position of the window in the input image.

Depending on the size of kernel, some number of border pixels are not produced in the output image. For example, in Figure 1.5, the window in the input image will reference non-existent neighbors for border pixels in the output image. In these cases, a number of trivial solutions exist to preserve the resolution of the image, such as repeating border pixels or simply padding the image with 0 values.

Convolution is a dynamic and powerful tool for applying a wide variety of image operations. These operations can help expose key elements in an image useful to the task

at hand. For example, a crosswalk detector may make use of a horizontal edge detector. In a CNN, hundreds or thousands of different kernels may be used. Some common mathematically-derived, hand-crafted kernels are shown in Table 1.6.

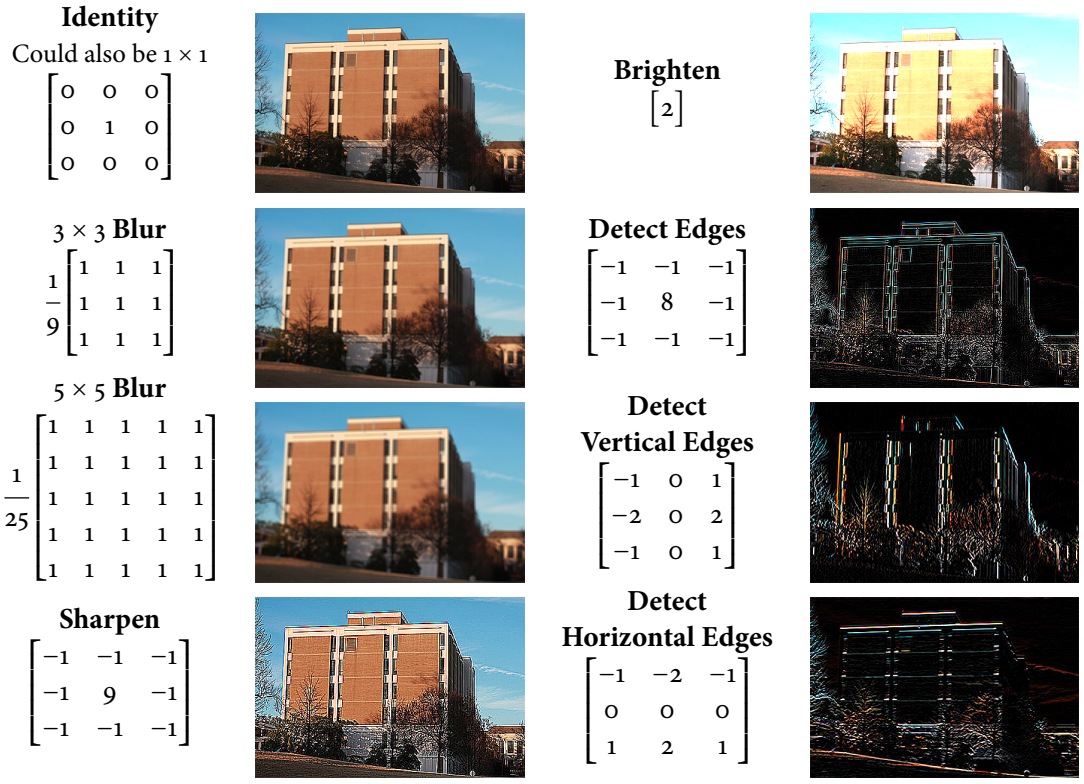


Figure 1.6: Sample convolutional kernels and their effects. The image for the "Identity" kernel is the input image. In a CNN, kernels are learned. Note that a scalar multiple is used to improve the readability of the blur kernels.

The use of convolution in a neural network is natural. Convolution involves computing the weighted sum, as does a neuron. Thus, the values of the convolution kernel are represented as neuron weights in a CNN. To account for the moving window, there exists a neuron for each pairing of a window to an output pixel. The weights are shared between

these neurons, though, as the convolution kernel should not change across space. [1]

When designing the CNN architecture, the size and number of kernels is set, but the weights are learned during training. The configurable number of kernels is equivalent to the number of output images, which brings up an important (though subtle) variation between standard image convolution and CNNs.

For a three-channel (i.e., RGB) image, a 2D kernel is typically applied to each color channel independently. This produces three outputs, which can again be displayed as a three-channel color image. This is how the images in Figure 1.6 were produced. In CNNs, however, we may wish to produce only a single output channel or many output channels. Furthermore, the model may be able to produce more useful output if multiple input channels can be considered simultaneously. For this reason, convolution kernels are 3D; two dimensions are configurable and the third is equal to the number of input channels. Thus, there are more weights to be learned to allow different channels to contribute differently to the output. The input window still moves in the 2D spatial domain and output from this 3D convolution is still the weighted sum of all input pixels, producing a single output value.

Thus, to summarize, in a convolutional layer, there exists a single neuron for each pixel in each output channel (excluding edge padding). The inputs to these neurons are the pixels in the spatially-associated window across all input channels. The weights on the input of the neuron are learned during the training of the network and weights are shared amongst all neurons for each output channel.

[1] When weights are shared, error is averaged across all instances during training with backpropagation.

Pooling Layers

Pooling layers operate much like convolutional layers. They move a certain-sized window across an input image and they produce an output image. Instead of learning parameters though, the operation is fixed: typically either taking the average of all values or the maximum of all values (which can't be done with a weighted average). Another difference is that instead of using a configurable number of output channels, there is exactly one output channel for each input channel. Furthermore, each channel is processed independently, unlike the 3D kernel in convolution layers.

In practice, this layer is used with an increased *stride* setting. Stride is configurable for both convolutional and pooling layers; it defines how many pixels should be skipped between each window location. A stride of 1 visits every possible location (as was described for convolutional layers), while a stride of 2 visits every other pixel, reducing the number of neurons and the resolution of the output channels by a factor of 2.

This is typically the primary use of pooling: to reduce the resolution of the image channels while maintaining important information from the larger resolution. Reduced resolution is desirable for both performance reasons and to improve spatial invariance of the model. For example, we may use "max pooling" with a stride of 2 to remember how much of an edge was detected by the previous convolutional layer, but by halving the resolution, we care only half as much about *where* it appears.

Fully Connected Layers

Fully connected (or "inner product") layers are the simple standard neural network layers introduced in Section 1.3.1 and in Figure 1.4. Whereas convolutional and pooling layers

are designed to extract important features from images, fully connected layers perform the final classification or regression task. Any spatial aspect of the data is lost at this point as well. For these reasons, fully connected layers are typically used as output layers, after all convolutional and pooling layers.

1.4 Overview of Dissertation

Our end-to-end solution is described and analyzed in this dissertation. Chapters are organized as follows:

- Chapter 2 explores the rich history of eye tracking research up through modern-day approaches. Approaches related to ours are reviewed in more detail. A survey of related datasets and crowdsourcing approaches are also covered.
- Chapter 3 covers our methodology for crowdsourcing gaze data.
- Chapter 4 describes the detailed setup of our eye tracking model.
- Chapter 5 describes evaluation techniques for both the dataset and the model. Results are presented and analyzed.
- Chapter 6 reflects on the implications of our work, highlighting key applications and future work.

This dissertation was written for an audience with a background in computer science and a rudimentary understanding of machine learning. Advanced topics are covered, but for the most part, our contributions should be made clear after reading the introduction.

Though this dissertation was written by a single author and the intent is to highlight the individual contributions to the field, this work was done in collaboration with others. As such, "we" and "our" are used throughout the work. The collaborators are graduate students studying at Massachusetts Institute of Technology in Boston, Massachusetts: Aditya Khosla (advised by Dr. Antonio Torralba) and Petr Kellnhofer (working with Wojciech Matusik, visiting from Max-Planck-Institut für Informatik in Saarbrücken, Germany).

Chapter 2

Review of the Literature

Eye tracking has a long history, largely due to its value in a variety of different fields. In this chapter, we will survey the history of eye tracking before exploring modern approaches. Then, we will introduce specific works related to the methodologies proposed in this work. The contributions of this work are underscored in light of the existing literature.

2.1 History of Eye Tracking

The idea of recording and studying eye movements has been around for centuries. Improvements to eye tracking methodologies have followed trends in research related to human visual attention. In this regard, the progress has been joint: as eye tracking methodologies have improved, so has our understanding of various aspects of the human visual system.

Before special devices were made to assist in the observation of eye movements, direct

observation was employed. In the late eighteenth century, Erasmus Darwin with his son, Robert Waring Darwin [1786] demonstrated the user of *afterimages*: the phenomenon where an image will remain burned into the retina for a period of time after seeing it.

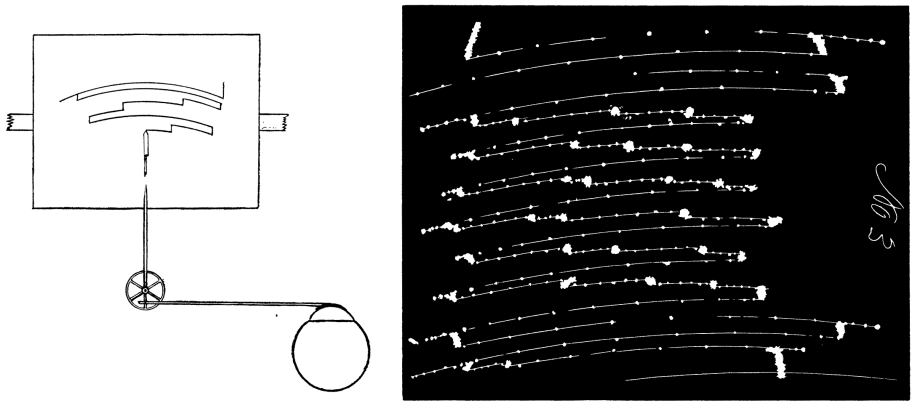
Darwin discovered fixational eye movements by trying to stare at a red dot on white paper without moving his eyes. The edges of the circle would "librate" on one side—a part of the afterimage would stand out against the white paper as his eyes moved quickly [Darwin and Darwin, 1786].

In a similar manner, William Charles Wells [1792] studied visual vertigo. First, he stared at a candle to temporarily burn an afterimage into his retina. It appeared stationary at first, but after spinning in circles until he became dizzy, the afterimage and the real image would drift apart for a while. He validated the relationship between afterimages and eye movements by watching someone else's eyes move during the same experiment.

Visual vertigo was one of a few medical reasons for studying eye movements in early eye tracking history. In the late nineteenth century, there was elevated interest in reading studies. In this context, an important discovery was made: Though perceived as being smooth, eyes jerk from point to point. Louis Émile Javal [1879] is widely credited with using the term "saccade" (French for "jerk") to describe this motion, though his individual contributions on the matter are debated [Wade and Tatler, 2009].

Instead, Lamare [1892] and Hering [1879] (working in Javal's laboratory [Wade, 2015]) should be credited with discovering saccade motions [Wade, 2010]. They were both able to record *when* eyes saccade by mounting rubber tubes on subjects' eyelids. They heard "clapping" sounds whenever the eye moved. They used this technique to validate older afterimage techniques.

By the end of the nineteenth century, the first attempts at building a mechanical eye tracker were introduced [Delabarre, 1898; Huey, 1898, 1900]. These early devices were used to record lateral eye movement during reading. Huey's method used a contact lens to physically attach the cornea to a device that recorded motions onto a smoked drum kymograph. See Figure 2.1 for an overview of this setup and Figure 2.1 for a sample kymograph recording. Subjects bit into a wax mold custom-fit to their teeth and attached to the machine to prevent head motion. Subjects' eyelids were held open as well. This early technique was invasive and was limited by the friction and inertia of the machine's parts [Wade, 2010].



(a) Setup of the eye tracker used in Huey [1898].

(b) Recording from Huey [1900]. Only lateral eye movements are recorded; vertical movement encodes time.

Figure 2.1: Early eye trackers were invasive and only recorded lateral movements.

Photographic approaches emerged around the turn of the twentieth century. Light reflected from the subject's eye was recorded onto a photographic plate, leaving a light trail for analysis. Orschansky [1899] used mirrors attached to the eye to reflect light while [Dodge and Cline, 1901; Dodge, 1903, 1904] used *corneal reflections* (a point source of

light reflected off the cornea; also called a *glint* [Hansen and Ji, 2010]). Light was passed through a horizontal slit while the recording surface slowly dropped, recording lateral movement over time (as with the kymograph recordings).

Dodge's use of corneal reflections was groundbreaking, as it was the first non-invasive recording technique. For this reason, it inspired many follow-up studies [Taylor, 1937; Wade, 2010] and corneal reflections are still in use today (see Section 2.2).

Corneal reflections have the property of moving only slightly as the eye moves. As a spherical surface rotates about its center, light is reflected in the same location (relative to the outside world); however, this slight motion occurs because the eye is neither entirely spherical nor does it rotate about its center [Richardson and Spivey, 2004].

> The movement of the corneal reflection lags behind that of the eye, and so presents a diminished copy of the original; and moreover this diminution is greater in some instances than in others, according to the direction of the eye's movement, the position of the object reflected in the cornea, and the direction from which the reflected image is observed.
>
> — George Malcolm Stratton [1902]

In some cases (as with Dodge and Stratton), this slight movement is used to measure eye movement, while in other cases (described in Section 2.2), the relative lack of movement is used as a reference point for eye tracking. Despite this dichotomy, corneal reflections have proven to be an important facet of eye tracking regardless of usage.

In the context of studying how people view geometric shapes, Stratton [1902] and Judd [1905] recorded both horizontal and vertical movements. Stratton continuously recorded corneal reflections in an otherwise dark room, but faced inaccuracy (due to the

nature of corneal reflections; see the above quote). Judd used a white speck placed on the eye along with motion picture photography. Their approach did not require a dark room, but they were limited by temporal resolution, recording only around 8–9 frames per second [Wade, 2015].

Similar photographic approaches were used for the better part of the twentieth century [Richardson and Spivey, 2004]. Buswell [1935] and Yarbus [1967] both recorded continuous gaze in two dimensions to explore a problem still widely studied today: How do people look at pictures? In these studies, viewing paths from multiple subjects were overlaid and a critical observation was made: The task given to the subject before viewing the image had a large impact on the viewing pattern. Buswell's recording device is pictured in Figure 2.2.

Over the years, eye trackers have been improved in various areas, including recording resolution in different dimensions (i.e., horizontal, vertical, and temporal), invasiveness, and accuracy. Even today, trade-offs must be made in the same areas, but technological advances have greatly reduced limitations. For a more in-depth survey of historical eye movement research, see Wade [2007, 2010, 2015]; Richardson and Spivey [2004]; Jacob and Karn [2003]; Young and Sheena [1975].

2.2 Mainstream Eye Tracking

Today, through advances in technology, many more hassle-free eye tracking solutions exist. Commercial implementations of modern research have made conducting eye tracking experiments relatively simple. They still typically require expensive hardware and a

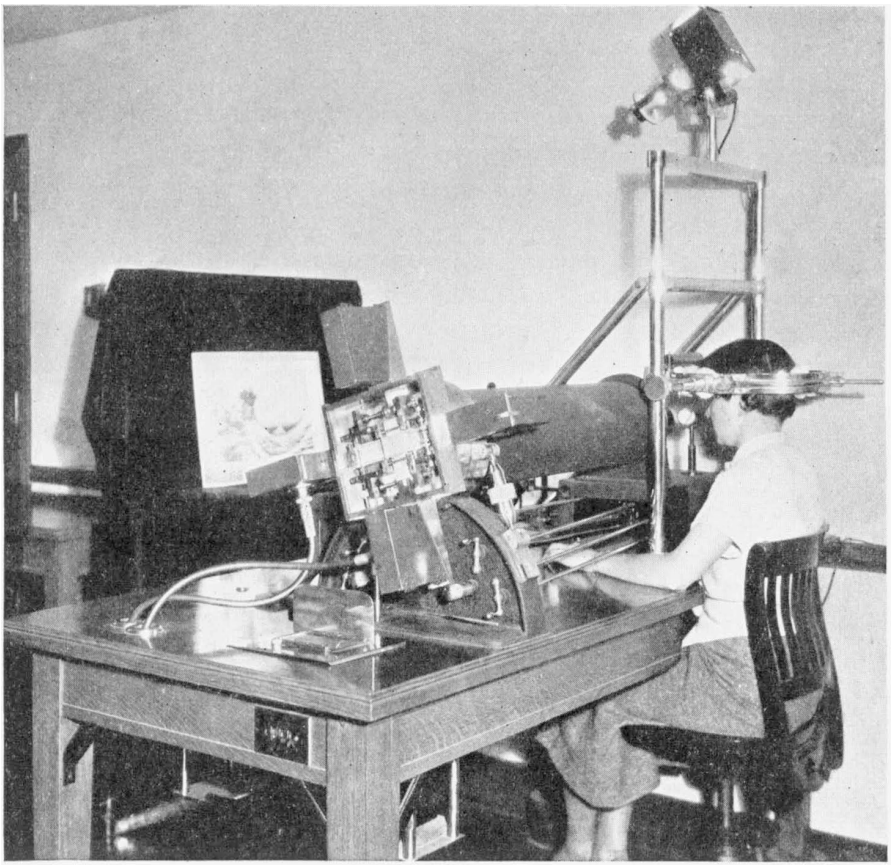


Figure 2.2: Eye tracker used by Buswell [1935].

relatively controlled environment, but high accuracy eye tracking is available.

The most widely-used scientific eye tracking solution in use today is *video-oculography* (VOG) [Duchowski, 2007]. Using frames from a video camera as input, certain features of the eye are detected and used to compute the point of regard. One such feature is the pupil center, often detected by a geometric model.

This technique works well when integrated into a headset, where the camera moves

with the head, but using the pupil center alone can be problematic otherwise. "Remote" solutions (where the tracking is done in a noninvasive way) must account for head motion. For a fixed gaze point, the pupil center relative to the corners of the eye can vary greatly depending on head pose.

To overcome this, corneal reflections (or "Purkinje images") are used as a mostly-fixed reference point on the eye. Light reflected off the cornea from a fixed light source will appear in nearly the same position on the eye for a fixed gaze point, regardless of head position (since the eye's absolute position will not change much relative to the light). On the other hand, if the eye gaze changes, the position of the pupil center relative to the reflection will change. Thus, the corneal reflection is used as a reference point to track eye movement relative to the outside world (as is the goal with eye tracking) rather than to the head. This use of corneal reflections differs from historical uses of corneal reflections for eye tracking (see Section 2.1).

To create a corneal reflection, most modern eye trackers integrate one or more light sources somewhere near the camera. Infrared light is typically used to avoid shining visible light into the subject's eyes. This approach produces some of the most accurate results today, but the use of a special light source requires specialized hardware and a relatively controlled environment.

While VOG is most prevalent today because it produces accurate results and can be used in a noninvasive way, there are other approaches in use. *Electro-oculography* (EOG) uses sensors placed around the subject's eyes to measure changes in potential, which can be mapped to eye movements. This approach is robust, even if the subject's eyes are closed, as is the case when monitoring eye movements during REM sleep. Also, contact

lenses are still used in some cases, where a metal ring's movement in a magnetic field can be sensed.

These modern approaches offer many practical solutions for academics looking to conduct eye tracking studies. However, there exists a significant barrier to entry due to the price and special hardware required. A robust approach using commodity cameras could greatly expand the audience for practical eye tracking.

2.3 Research Trends in Eye Tracking

In most cases, mainstream commercial eye trackers are the direct result of successful research plus time required to refine the technique. Current eye tracking research is primarily geared towards improving video-oculography approaches. Many of these approaches focus on removing limitations—most notably, the requirement for specialized hardware. Here, we give a brief overview of gaze estimation methods in current research. We suggest the survey, Hansen and Ji [2010], for a more complete picture.

Gaze estimation methods can be divided into two broad categories: model-based and appearance-based [Hansen and Ji, 2010]. Model-based approaches use a geometric model of the eye. These approaches can be further classified as either corneal-reflection-based approaches or shape-based approaches. Corneal-reflection-based methods [Yoo and Chung, 2005; Zhu and Ji, 2005; Zhu et al., 2006; Hennessey et al., 2006] rely on external light sources to detect eye features. On the other hand, shape based methods [Ishikawa, 2004; Chen and Ji, 2008; Valenti et al., 2012; Hansen and Pece, 2005] infer gaze direction from observed eye shapes, such as pupil centers and iris edges. These ap-

proaches tend to perform poorly when presented with low-quality images, as is possible in our unconstrained scenario.

Appearance-based methods [Tan et al., 2002; Sewell and Komogortsev, 2010; Lu et al., 2014b,a; Torricelli et al., 2008; Baluja and Pomerleau, 1994] directly use eye images as input (not requiring conformity to a model) and can potentially work on low-resolution images. Appearance-based methods are believed [Zhang et al., 2015] to require larger amounts of user-specific training data as compared to model-based methods.

Our proposed model is an appearance-based method; however, we show that our model is able to generalize well to novel faces without needing user-specific data. While calibration is helpful, its impact is not as significant as it is in other approaches, given our model's inherent generalization ability achieved through the use of deep learning and large-scale data. Thus, our model does not have to rely on visual saliency maps [Chen and Ji, 2011; Sugano et al., 2013] or key presses [Sugano et al., 2008] to achieve accurate calibration-free gaze estimation.

2.4 Gaze Datasets

Used for training and evaluating gaze models, gaze datasets are important tools for eye tracking research. There are a number of publicly available gaze datasets in the community. We survey many of them in this section, and we summarize various distinctions between them in Table5.1 (see Chapter 5).

Many of the earlier datasets [McMurrough et al., 2012; Weidenbacher et al., 2007; Smith et al., 2013] either lack significant variation in head pose or have a coarse gaze point

sampling density. We overcome this by encouraging participants to move their head while recording and generating a random distribution of gaze points for each participant.

More modern datasets follow an approach similar to ours. An example of this is TabletGaze [Huang et al., 2015], which we use extensively for evaluation in Chapter 5. In this dataset, subjects are asked to look at dots on a tablet screen while a video is recorded on the device's front-facing camera. Dots are randomly selected from a set of 35 dots arranged evenly across the screen in a 5×7 grid. Each dot appears for three seconds.

Participants are asked to complete the task in each of four loosely-described poses: standing, sitting, slouching, and lying. They are also asked to hold the tablet in the landscape orientation only. Otherwise, they are free to hold the device as desired. Recording is done in a lab setting with overhead lights turned off to prevent backlighting.

Before using the data, the authors prune away problematic frames and recordings. The authors perform a manual inspection of the data to determine loss of attention resulting in mislabeled data. Also, the first 1.5–2.5 seconds of each dot are removed to allow the participant sufficient time to saccade to the dot. Blinks and blur are also detected and removed.

We also perform evaluation on MPIIGaze [Zhang et al., 2015], which only involves 15 subjects and uses laptops, but conducts experiments outside of the lab over the course of several months. Subjects use special software that interrupts them during normal laptop use for true in-the-wild data. No instructions are given regarding pose.

In the task, subjects are asked to look at a sequence of 20 dots that shrink. To ensure attentiveness while viewing dots, they are asked to push a key on the keyboard right before the dot disappears.

While some of the modern datasets follow a similar approach to ours [Sugano et al., 2014; Mora et al., 2014; Zhang et al., 2015; Huang et al., 2015], their scale—especially in the number of participants—is rather limited. We overcome this through the use of crowdsourcing, allowing us to build a dataset with roughly 20 times as many participants as the current largest dataset. Further, unlike Zhang et al. [2015], given our recording permissions, we will release the complete images without post-processing. We believe that our dataset will serve as an invaluable resource for future work in this domain.

Chapter 3

GazeCapture: Crowdsourcing Gaze Data

Data lies at the heart of many scientific inquiries. More data points often yield better models and more compelling results. For example, imagine being trained for a new job as flight attendant—the more examples you are shown of what's expected of you, the better your chance for success and the better you'll be able to evaluate your own performance.

Furthermore, data quality is just as important as data quantity. Returning to our example, imagine your extensive training deals with customer service, flight safety, and other cabin operations. If, during a flight, you are then expected to fly the airplane, you may not perform so well. In practice, issues with data quality may be more subtle, but finding the right data can have a significant impact on the real-world applicability of a model.

The end goal of eye tracking studies is to be able to predict where a person is looking.

Thus, the needed data should include some measurement of the subjects' eye(s) along with known ground truth annotations of where they are looking. As described in Section 2.4 and validated in Section 5.1, existing datasets of this type are limited both in terms of size and in variability. This is particularly true regarding their applicability to deep learning approaches, which have proven to be very powerful solutions where large datasets exist.

In this chapter, we describe a crowdsourcing approach, "GazeCapture," for collecting gaze data on a large scale. Crowdsourcing has not yet been applied to collecting ground truth gaze data, so we describe our methodologies for maximizing the size and quality of our dataset. We use this approach to collect a novel dataset, "Mobileyes," which we analyze to validate our methods in Section 5.1.

3.1 Experimental Design

As with other recent papers, we pose the gaze estimation problem as a mapping from an input image to an (x, y) pair of screen coordinates. Although our model actually predicts the gaze location in physical space relative to the camera (see Section 4.2), the predictions are directly mapped to screen coordinates. The use of screen coordinates for ground truth labels (rather than, e.g., 3D gaze vectors) is well-suited to our ultimate goal of predicting where a subject is looking on a screen. Also, in a crowdsourcing setting, this simple approach is pragmatic, as it avoids special constraints or tools.

To collect such ground truth data, on-screen dots are displayed to participants and they are asked to look at the dots while video frames are recorded using the front-facing

camera. We chose to target smartphones and tablets because of their many applications in an increasingly-mobile world. Also, by focusing on mobile devices, where camera motion is expected, we anticipate a more robust solution, such that the model may be applied in a wider variety of environments.

3.1.1 Dot Display

The task is designed to be completed by subjects in one sitting of approximately ten to fifteen minutes. In the main part of the task, a single red dot appears on a blank white background. This strong color contrast is chosen to maximize the saliency of the point on the screen. To further increase the saliency of the dot, we make the dot continually oscillate between a larger and a smaller size without moving the center of the dot (which is precisely identified by a small, fixed-size black point). This motion is also intended to help keep attention after the initial fixation.

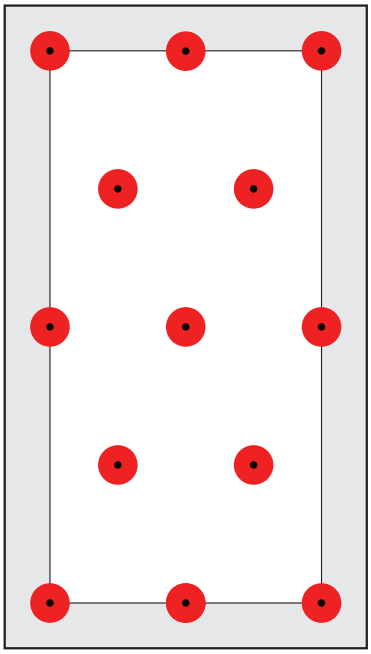


Figure 3.1: 13 calibration dots, as used in Xu et al. [2015]. The gray border indicates the margin where no dots are displayed to prevent them from going off-screen. Evaluation using these dots is described in Section 5.4.2.

Dot locations are selected both randomly and from a set of 13 fixed locations (see Figure 3.1). The fixed locations represent a typical eye tracking calibration task, as is used by Xu et al. [2015], designed to provide good coverage of the screen. See Section 5.4.2 for our application of these fixed

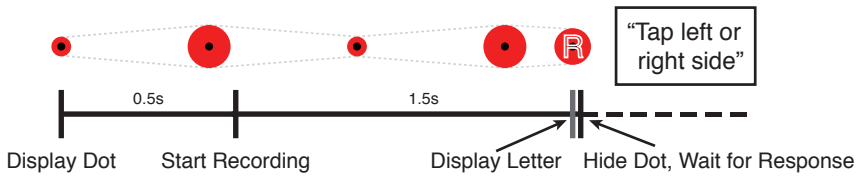


Figure 3.2: The timeline of a single dot. The recording begins 0.5 seconds after displaying the dot to allow the subject time to saccade to the dot. The dotted lines around the dots indicate the way a dot pulses as it is being displayed (to attract and maintain attention). A letter is flashed inside the dot for 50 milliseconds at the end for quality assurance (see Section 3.1.2).

locations in a calibration scenario. All dot centers appear within a fixed border to prevent the dot from going off of the screen.

We select the duration of each dot's display qualitatively, by examining data collected during both development and an early round of crowdsourcing. There exist a few trade-offs between shorter and longer dot durations. For shorter durations, more dots can be displayed in the same time period, but fewer samples will be collected at each point. If the duration is too short, the subject may not have time to fixate on the dot at all. For longer dot durations, we notice that subjects may look away from the dot. We find that 1.5 seconds for each dot is a good balance between these trade-offs. We also find that omitting the first 0.5 seconds of each recording is sufficient to allow most subjects time to saccade to the dot. See Figure 3.2 for a timeline of our dot display process.

3.1.2 Attentiveness Task

Perhaps one of the most important aspects of the process was finding a way to ensure subjects were paying attention. We reduce the likelihood of on-device interruptions (see Section 3.2.4), but in a crowdsourcing scenario, we do not expect intrinsic motivation to provide quality data. We suggest an *attentiveness task* that is essential to quality control.

Immediately before the end of each dot's display, we flash the outline of a letter inside the dot for 50 milliseconds (see Figure 3.2). Afterward, the subject is asked to report which letter they saw—they must tap the left side of the screen for "L" and the right side for "R." There is also an "I Don't Know" button for users to self-report inattentiveness.

The user is warned after each incorrect answer and the dot is displayed again. The number of incorrect answers is logged to help identify bad data. With only one correct and one incorrect answer, some bad data (i.e., of an inattentive subject) is likely to pass this filter. This decision is intentional to maximize speed. Distinguishing left from right is simple and can be performed quickly. The input method is fast and reliable as well, requiring only two buttons that fill the screen. In this way, we anticipate a high signal-to-noise ratio.

The user's response time for each answer is also logged and—although we do not currently make use of the information—the average is reported to the user when they have completed the task. We discovered through post-study feedback (see Section 3.3.1) that some participants find the the task enjoyable like a game:

- "Fun. Like a game."
- "That was awesome!"

- "It was a fun game. It probably wasn't a game, but I still thought it was fun"

Others find it less enjoyable:

- "Mild headache from the task."
- "I got a little dizzy!"
- "That was way harder than it seemed."

Many of these comments reinforce our confidence in the duration of the letter display—it is short enough to be challenging and long enough for attentive subjects to see. Furthermore, this encourages us to consider gamification for future versions of this task. Such an approach could be valuable to attracting and engaging users (as was done in Xu et al. [2015]), perhaps even without incentive. Simply adjusting dot length over time—perhaps in response to the user's attentiveness task accuracy—could create a more engaging *flow* through the duration of the task [Deci and Ryan, 1985].

3.1.3 Head Motion

We encourage continuous head motion (or device motion) during initial instructions (see Section 3.3.3) and during the task in the app (see Figure 3.4). We do this to increase variety in pose and visual appearance. Even for frames recorded during a single dot (which are typically highly correlated, and thus, less useful for machine learning) we anticipate the continuous motion will provide more frame-to-frame variety. A side effect is an increased likelihood of motion blur, especially in low-light scenarios, but even this may be useful variety to have for designing a robust model for use in the real world.

3.1.4 Device Orientation

During the task, the user is asked to rotate their device on occasion as to record dots for each supported orientation. This is important because the appearance of the user's eyes will differ greatly for any given dot at different orientations. We account for the orientation in the way we map between screen coordinates and our prediction space (see Section 4.2).

All four possible orientations are supported on iPads, but we do not support the "portrait upside down" orientation on iPhone since few apps support it in practice. An even number of dots is shown for each supported orientation. We also display all 13 calibration points at each orientation.

3.2 The GazeCapture App

Typically, crowdsourcing tasks are implemented as a web-based application. Xu et al. [2015] even performed live processing of a camera feed in a web browser for their crowdsourced eye tracking study (utilizing a preexisting gaze prediction model). However, browser-based camera access is still a nascent technology and is faced with many limitations, especially on mobile devices. Thus, we decided that a native app would be the best solution.

Figure 3.3: The GazeCapture app icon.

3.2.1 Platform

We designed the experiment as an iOS app, supporting iPhone and iPad devices. The majority of the code was written in Swift, with some Objective-C and C++ as necessary.

The decision to develop for iOS rather than Android was primarily due to the smaller set of hardware configurations to consider. This decision was particularly advantageous when testing on-device performance of the app and when gathering physical device specifications (as described in Section 4.2).

A server application was written (using PHP and MySQL) to accept uploaded recordings, serve app configurations (e.g., dot duration) and manage different aspects of crowdsourcing.

3.2.2 Consent Form

After launching the app, the user is prompted for their date of birth. This is used to confirm that they are at least 18 years of age. After verifying this, a consent form is presented. The consent form clearly describes the purpose of the study, procedures, privacy considerations, and contact information, amongst other details. This form was approved by the IRB, as described in Section 3.3.4. The user must press an "Accept" button to continue.

3.2.3 Participation Code

The user is presented with a text field and asked to enter a code if they have one. In Section 3.3.2, we show the need to uniquely identify subjects via a code. This request is intentionally vague and generic, as to support a variety of contexts without requiring an

update to the app. The code is validated with the server before continuing. Alternatively, there exists a "Skip" button for anyone without a code.

3.2.4 Device Setup

So as to reduce the likelihood of distracting on-device notifications, we require the user to put their device into "Airplane Mode," disabling communication devices for the duration of the task. If the setting is disabled at any point during the task, we pause the recording session. At this point, we also prompt the user to grant access for our app to use their camera.

3.2.5 Interactive Instructions

We use an interactive instruction stage to inform, engage, and train users before starting the experiment. First, a camera preview is shown to the user with bounding boxes around the face and eyes. The user is asked to ensure good lighting conditions and to make sure their face is visible at all times. Since the camera feed is not displayed to the user during the experiment, we find this screen useful for putting the user in a feasible pose.

Next, the user is shown sample dots and asked to notice the flashing letter before pressing "Next." The attentiveness task is then described with a visualization of the full-screen buttons. To verify that the user is ready, we display two consecutive practice dots. Finally, we tell the user they're ready and we summarize the instructions.

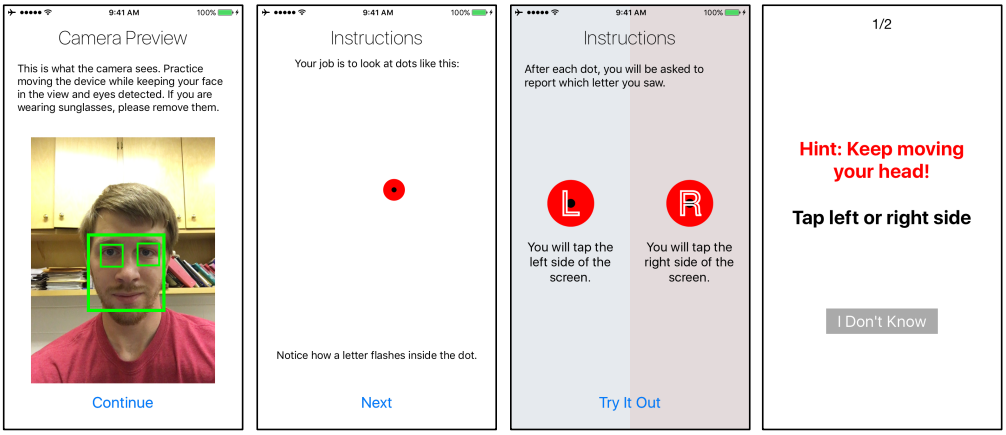


Figure 3.4: Select screenshots from GazeCapture's interactive instructions. After verifying that the camera can see the subject (a) the dots with blinking letters are introduced (b). The attentiveness task is described (c) then two demo dots are displayed (d) for practice.

3.2.6 Main Task

During the experiment, the user has a set of three controls: "Start"/"Reset," "Pause"/"Resume," and "Cancel." These controls were designed to stay out of the way and not to attract attention. In some cases, dots may overlap the buttons. After starting the experiment, the user is shown a brief countdown before displaying dots. The countdown is shown again whenever flow is broken by an incorrect answer or resuming from a paused state.

After a predetermined number of completed dots, the user is prompted to rotate their device to a certain orientation. This is accomplished by presenting an arrow and asking the user to make it point upward.

We repeat a dot if the user fails the attentiveness task (or if the user taps "I Don't

Know"). Also, if no faces are detected in any of the frames, the dot is repeated with some suggestions to the user. For example, after rotating to a landscape orientation, we find that many users may inadvertently cover up the camera with their fingers.

Also, on occasion, the user is reminded to continue moving their head. We find that this greatly helped increase pose variety, as users frequently forgot to move due to their focus on the task. As one user said in post-study feedback (see Section 3.3.1), "The reminder to keep moving helped a lot. I frequently forgot!"

The user can track their progress through the task by an indicator at the top of the screen. This is only displayed during the attentiveness task.

3.2.7 Data Recording

During the display of a dot, video frames are captured from the camera at a fixed resolution of 640 × 480 (or 480 × 640, depending on the orientation). For each frame, a fast and robust GPU face detector and a CPU eye detector is used (both using built-in Apple frameworks). The eye detector was configured to provide fast detections at the cost of accuracy—we expect our model to account for inaccurate detections.

Depending on the device and lighting conditions, anywhere from 1–19 frames may be recorded for a dot. In this way, the same pipeline used for data collection could also be used in a realistic detection environment where face and eye detections are required.

We also record data from the device's motion sensors at a rate of 60 samples per second. This includes accelerometer, gyroscope, and magnetometer data. We do not currently make use of this information, but we record it in case it could be used to better estimate device pose in the future.

3.2.8 Data Upload

After the experiment is complete, we present average reaction time, total number of errors, and size of the data to the user. After asking the user to disable Airplane Mode, a button is presented to upload the data. The upload is typically a few hundred megabytes. The user is also given the option to erase their data and start over.

3.3 Crowdsourcing Procedure

Though the app is designed to be scalable, finding and managing participants requires a considerable amount of overhead. In this section, we describe our procedures regarding human subjects in our study.

3.3.1 Recruitment

Most participants are recruited via *Amazon Mechanical Turk* (AMT), a service that allows people to earn money (to spend on the Amazon.com store) for completing short-term tasks. Registered as an AMT "Requester," we create a *Human Intelligence Task* (HIT) for AMT "Workers" to use our app. Workers discover our HIT through either keyword searches or lists sorted by the pay or number of available tasks. The description of our HIT makes clear that the Worker must have a supported Apple device.

After accepting, Workers are given 24 hours to download the app and complete the HIT while incentive is reserved for them upon their completion. If the Worker does not complete the task within 24 hours, the HIT (and the associated money) is returned to the pool of available tasks, allowing another AMT Worker to accept the work. We release

HITs in batches during development to refine our methodology, instructions, and app. Feedback is gathered through a short, optional demographic survey presented after the HIT is complete. The survey includes fields for age, gender, race, and a comment.

We also recruit students in large undergraduate computer science classes (with between a 60% and 70% participation rate) at The University of Georgia (UGA). A brief presentation and demo of the app was given to motivate the topic. For students without a supported Apple device, we provide devices during office hours. No survey is given to UGA participants, though feedback has typically been positive.

3.3.2 Incentive

AMT Workers are offered an incentive of 1.00 USD for completing the HIT. Many other HITs pay significantly less for much shorter tasks, so our task appears high when Workers sort HITs by pay. We also have to pay a premium to Amazon for using their service. In cases where Workers experience a problem with our app (e.g., a crash), we still give the incentive.

To associate a HIT with a particular upload from our app (independently downloaded from the Apple App Store), we generate a code for each accepted HIT. We tell the user that they must use this code in the app (see Section 3.2.3) to receive their incentive.

UGA students are offered extra credit for their participation. They are asked to enter their official school ID with a special prefix as their code to identify their participation. We emphasized that the decision to participate or not participate in this research would have no impact on their grade. For students preferring a non-research alternative, we

offered the same amount of extra credit for writing a couple of paragraphs brainstorming ideas on uses for eye tracking.

Though we do not reject multiple uploads from the same subject, we do not offer incentive more than once. As we show later in Figure 5.3, preferring additional subjects to additional samples produces better results.

3.3.3 Instructions

For both AMT and UGA populations, we present a specialized web page for prospective participants. On this page, we provide a description of the task, app installation and usage instructions, and other general advice. For example, we recommend that users complete this task over Wi-Fi due to the heavy data usage. On this page, we make use of animated images to describe more complex subjects, such as the expected head movement and how to enable Airplane Mode. We find that we can improve collected data quality by fine-tuning the way we present our expectations.

3.3.4 Ethical Considerations

As this study makes use of human subjects, we obtained an *exempt approval* from The University of Georgia Institutional Review Board (IRB). The approval can be referenced using UGA IRB IDs `STUDY00002656` and `MOD00002239`. Through this process, we reported and refined key ethical aspects of our research, including recruitment procedures, incentives, informed consent, and data collection methods.

An important outcome of this process is our ability to release full camera images to the research community as a part of our dataset. Other datasets have been limited in this

regard (e.g., Zhang et al. [2015], which only includes eye images). We believe models have the capacity to perform better given complete camera input.

Chapter 4

iTracker: Deep Model

Given the large scale of the data, we can hope to learn eye tracking end-to-end using deep learning without having to include hand-crafted features such as the head pose [Zhang et al., 2015]. Inspired by well-performing naïve approaches, we build a convolutional neural network customized to our problem. We select the inputs and prediction space to best suit our purposes, then fine-tune the design and learning parameters for optimal results. We call our model "iTracker."

This initial model is computationally expensive and incorporates all of the required information to make a highly accurate prediction of gaze. However, we find that this model is slow and cannot be incorporated into modern mobile devices for real-time eye tracking. Thus, we use techniques such as *dark knowledge* [Hinton et al., 2015] to help learn a smaller model that achieves a similar performance while using only a fraction of the parameters and being able to run at a reasonable frame rate on a modern mobile device.

4.1 Inputs

Rather than feeding our model only eye crops or entire image frames, we select inputs that we intuitively surmise should be most useful to the network. Also, by using only information built into the GazeCapture pipeline (i.e., face and eye detections), we are operating in a practically-implementable domain.

First, we use both left and right eye images cropped from the frame. We use eye

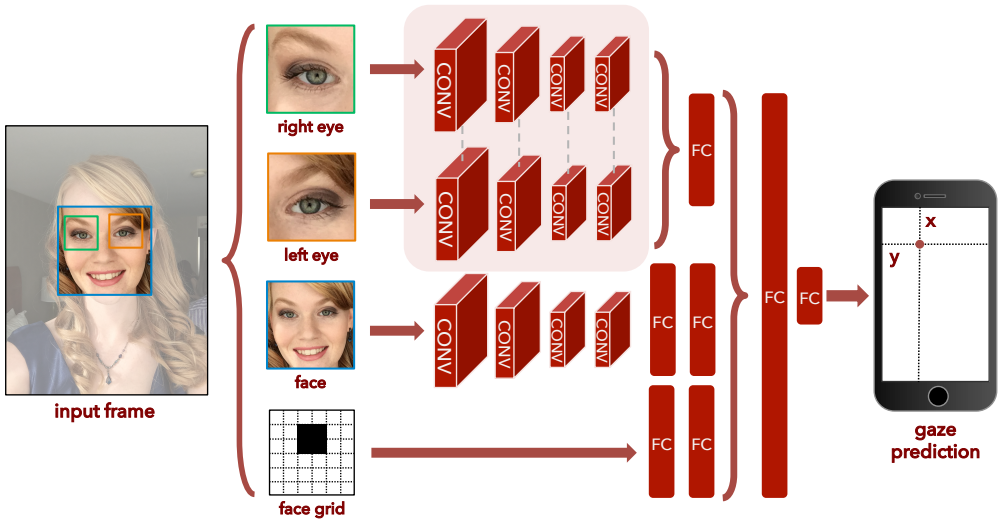


Figure 4.1: Overview of our model, "iTracker." Inputs include left eye, right eye, and face images detected and cropped from the original frame. The face grid input is used to indicate the location and size of the head within the frame. The convolutional layers are same for all image pipelines. Dotted lines between eye convolutional layers indicate that the weights are shared. The pooling layers are not shown, but the relative size of the convolutional layers is indicative of reduced resolution from pooling (and in the first case, convolutional stride). See Table 4.1 for layer details. The output is the distance from the camera which is then directly mapped to on-screen coordinates.

detections saved at the time of capture. The eye detector used in GazeCapture (an Apple library) was selected with performance in mind; it is fast at the expense of some accuracy (see Figure 4.2). Thus, we do not require tight crops. As such, we find that a relatively large image size (224×224) performs best over lower resolutions.

We also include the face image at a resolution of 224×224. This is designed to take the place of the head pose model used in Zhang et al. [2015]. If the recorded face detection is outside of the bounds of the image, edge pixels are repeated for padding.

Finally, we include face location as an input to help the model determine the subject's pose. Rather than directly using the coordinates and the size of the face bounding box, we use a binary mask to give the network greater representational power, as is done in Recasens* et al. [2015]. We create a 25×25 grid of bits to represent the image frame. We place 1's in the relative position of the face bounding box within the image and 0's elsewhere. We call this input *face grid*. We expect this input to be particularly useful in distinguishing between device orientations; if the camera is to the left of the subject's field of view, the face bounding box will likely appear towards the left of the frame.

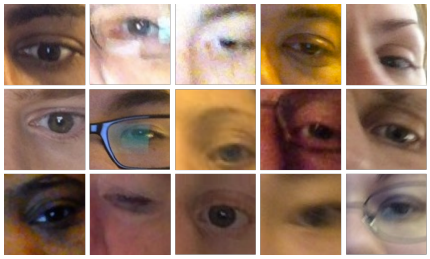


Figure 4.2: Sample left eye crops. We expect our model to compensate for inaccurate eye detections.

We assume that the perspective and field of view is similar across the front-facing cameras on Apple devices. As such, we assume our model should naturally be robust to any minor differences.

4.2 Prediction Space

We want our solution to naturally adapt to different devices and orientations. Directly predicting screen coordinates would not be meaningful beyond a single device in a single orientation since the input could vary significantly between devices. Instead, we leverage the fact that the front-facing camera is typically on the same plane as, and facing the same direction as, the screen. As shown in Figure 4.3, we predict the dot location relative to the camera (in centimeters in the x and y directions).

This requires precise measurements of device screen sizes and camera placement, which we acquire from an Apple document for case designers. Specifically, we use the distance between the camera and the top-left corner of the screen (in both the horizontal and vertical directions) as well as the width and height of the screen.

4.3 Architecture

We use deep convolutional neural networks (CNNs; see Section 1.3.4) to make use of our large dataset. Table 4.1 describes the details of each layer in our CNN, which we will motivate in this section.

The `conv1_*` through `conv3_*` layers (where the $*$ indicates a different copy for each image input) are unmodified from the ImageNet [Krizhevsky et al., 2012] architecture. This includes `pool1_*` and `pool2_*`, which are used to reduce resolution. We choose these ImageNet layers because our extensive baseline evaluation indicates that they are useful for representing eye images (see Section 5.2).

The `conv3_*` layer is very large, producing 384 different image channels. In `conv4_*`,

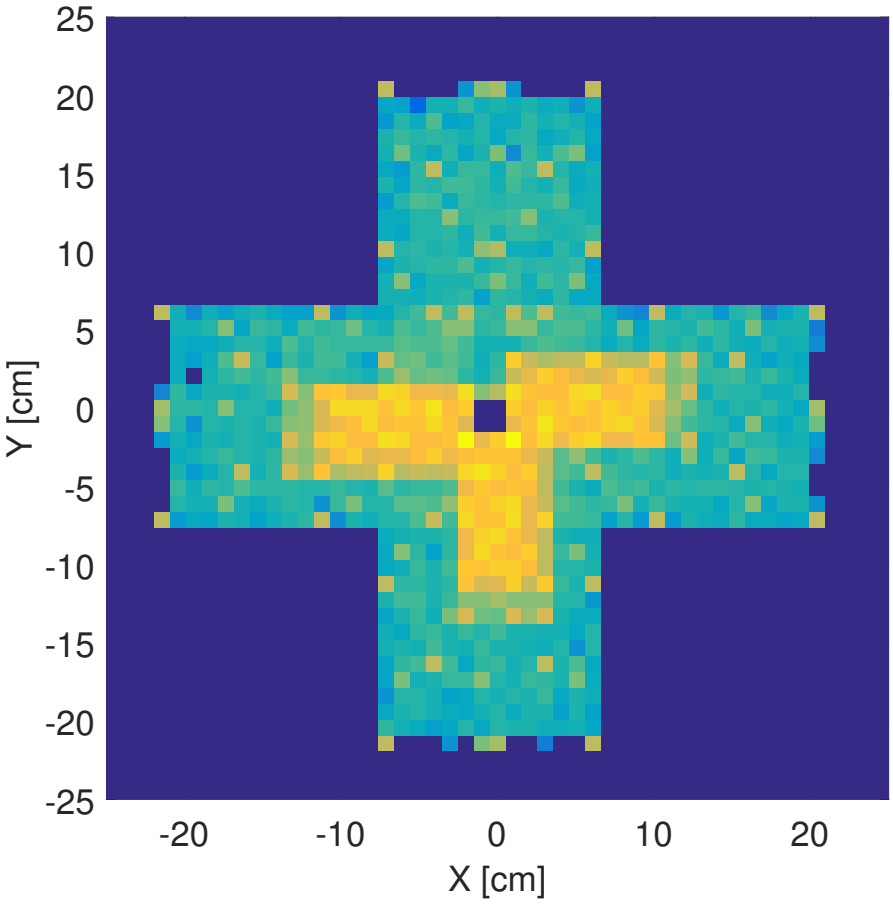


Figure 4.3: Locations of dots shown to participants, mapped to our prediction space. Axes measure centimeters from the camera. Brighter areas represent a higher concentration of dots in our dataset. iPhone screens in three supported orientations and iPad screens in four supported orientations can be easily distinguished. The empty area in the center is indicative of the camera position, since we cannot display dots on the camera.

we use a 1×1 kernel to avoid spatial convolution while still weighting across all channels. This allows the network to selectively combine image channels (while still maintaining the 2D structure), suppressing those that are less useful for making gaze predictions.

The left and right eye convolutional layers are configured to share weights due to the similarity between the inputs. Furthermore, we concatenate the output from both eyes immediately after the convolution layers (`conv4_l` and `conv4_r`).

At this point, we apply fully connected layers to each input pipeline separately. This includes the face grid, which is not processed by convolutional layers at all; the input is simple enough to not require convolution, even though it is 2D. Input pipelines are concatenated as shown in Figure 4.1. This arrangement of fully connected layers being concatenated helps combine and compact information from the different inputs. Ultimately, the final fully connected layer produces two output values in our prediction space (see Section 4.2).

All layers (except for the output layer and non-weighted pooling layers) make use of *rectified linear units* (ReLUs) to apply an activation function on each neuron's output. Also called *rectifier*, or *hinge* activation, the function is simple: $max(0, x)$. This activation function has been popular in deep learning due to its fast computation, sparse output, efficient backpropagation, and biological plausibility [Glorot et al., 2011]. Furthermore, in keeping with the ImageNet architecture for our early layers, *local response normalization* (LRN) is used to help the network generalize from training data [Krizhevsky et al., 2012].

4.4 Data Augmentation

In the case of certain experiments, we artificially increase our dataset size by a factor of 25. We find that this helps prevent overfitting, improve performance, and promote

Layer Name	Type	Kernel Size	Stride	Padding	Output Size
`conv1_*`	Convolutional	11×11	4	0	$96 @ 54 \times 54$
`pool1_*`	Max Pooling	3×3	2	0	$96 @ 27 \times 27$
`conv2_*`	Convolutional	5×5	1	2	$256 @ 27 \times 27$
`pool2_*`	Max Pooling	3×3	2	0	$256 @ 13 \times 13$
`conv3_*`	Convolutional	3×3	1	1	$384 @ 13 \times 13$
`conv4_*`	Convolutional	1×1	1	0	$64 @ 13 \times 13$
`fc1_e`	Fully Connected	—	—	—	128
`fc1_f`	Fully Connected	—	—	—	128
`fc1_fg`	Fully Connected	—	—	—	256
`fc2_f`	Fully Connected	—	—	—	64
`fc2_fg`	Fully Connected	—	—	—	128
`fc1`	Fully Connected	—	—	—	128
`fc2`	Fully Connected	—	—	—	2

Table 4.1: Layer-by-layer details for the iTracker architecture. * indicates that all convolutional and pooling layers are duplicated for the each input image: left eye (`l`), right eye (`r`), and the face (`f`). Likewise, `e` indicates both eyes, concatenated and `fg` indicates face grid. "Output Size" for convolutional and pooling layers is the number of image channels produced along with their resolution. For fully connected layers, this represents the number of neurons. Input images are 224×224.

spatial invariance for inaccurate eye detections. We achieve this by pairing five different face detection variants with five different eye detection variants. The five variants include the original detection plus detections shifted up, down, left, and right. The shifted face detections are moved enough such that the face grid is also shifted one column or row. In this way, the augmentation affects all inputs.

In Section 5.4.1, we demonstrate that augmentation is effective both for training and testing. Training on the augmented data improves accuracy on the original test set. Accuracy can be improved further by augmenting test samples, making a prediction from each variant, then averaging all predictions together. In Section 5.4.1, we show how this single averaged prediction tends to be better than the prediction from the original sample alone. Augmented test results require 25 forward passes through the network instead of just one, but it is a feasible approach for improving real-world accuracy.

4.5 Training

Our primary network is trained on our augmented dataset to maximize accuracy. We initialize eye ImageNet [Krizhevsky et al., 2012] layers with ImageNet weights, as they are known to work well (see Section 5.2). We use two different *learning rates*. First, we use a learning rate of 0.001 for 0.65 epochs over our augmented data (16.25 epochs relative to our original data). Then, we step to a learning rate of 0.0001 and continue training for half the duration of the first learning rate (i.e., 0.33 epochs over our augmented data and 8.13 epochs relative to our original data).

We use the open source deep learning framework, Caffe [Jia et al., 2014], accelerated

by cuDNN [Chetlur et al., 2014]. Using a single GeForce GTX TITAN X GPU with 12 GB of memory, training takes about 15 hours.

4.6 Truncation

In our setting, we assume that the system is aware of the current device's orientation and screen size. This information is not used to make predictions, but it is required to map from our prediction space to screen coordinates. While our prediction space should naturally account for off-screen gaze predictions, we truncate any such predictions. This is done by clamping to the boundary of the screen, both in the x and the y direction.

4.7 Real-Time Inference

As our goal is to build an eye tracker that is practically useful, we provide evidence that our model can be applied on resource-constrained mobile devices. Our primary model produces accurate results, but it is too large to run in real time (and with limited memory) on existing mobile devices. Thus, we reduce the size of the network significantly for a more feasible solution.

While we design the iTracker network to be robust to poor-quality eye detections, we find that tighter crops make the biggest difference in network size. Tighter crops reduce the expectations put on the model to localize the eye and allow lower-resolution images (non-square crops scaled disproportionately to 80×80) to be used for better performance (due to significantly fewer neurons needed across all convolutional and pooling layers). We use facial landmark eye detections [Baltrusaitis et al., 2013] to get these crops, as

Layer Name	Type	Kernel Size	Stride	Padding	Output Size
conv1_*	Convolutional	5 × 5	2	2	64 @ 40 × 40
pool1_*	Max Pooling	3 × 3	2	0	64 @ 20 × 20
conv2_*	Convolutional	5 × 5	2	2	32 @ 10 × 10
pool2_*	Average Pooling	3 × 3	2	0	32 @5 × 5
conv3_*	Convolutional	5 × 5	2	2	64 @ 3 × 3
pool3_*	Average Pooling	3 × 3	2	0	64 @ 1 × 1
fc1_e	Fully Connected	—	—	—	128
fc1_f	Fully Connected	—	—	—	128
fc1_fg	Fully Connected	—	—	—	256
fc2_f	Fully Connected	—	—	—	64
fc2_fg	Fully Connected	—	—	—	128
fc1	Fully Connected	—	—	—	128
fc2	Fully Connected	—	—	—	2

Table 4.2: Layer-by-layer details for the real-time iTracker architecture. This table is set up in the same way as Table 4.1. The fully-connected layers are exactly the same. Input images are 80 × 80.

used in Zhang et al. [2015]. This detector takes longer to run than the detector used in the GazeCapture pipeline, but we find the trade-off worthwhile considering the relative speedup and accuracy of the smaller network.

We further reduce complexity of the network by decreasing the kernel size and increasing the stride of the convolutional layers. We leave the fully connected layers untouched as they are very fast compared to the convolutional layers. Specifics of the network architecture can be found in Table 4.2.

Finally, encouraged by the work of Hinton et al. [2015], we apply ideas related to dark knowledge to train our smaller model. Typically, training is done by optimizing a loss function which compares the last-layer predictions to the ground truth. We find that our

smaller model struggles to make the same end-to-end connections as our larger model, resulting in reduced accuracy.

To overcome this, we compute loss by comparing the final two layers of our smaller network to the final two layers of our original network (i.e., `fc1` and `fc2`). In this way, we provide additional supervision to help our smaller network learn the same way our larger network did. This process uses an additional training hyperparameter, which controls the relative loss between the last layer and the penultimate layer. We find that weighting the penultimate layer to 60% of the final layer produced the best results.

Chapter 5

Experiments

In this chapter, we evaluate the methodologies proposed in Chapter 3 and Chapter 4. First, we report the makeup of our dataset and analyze the variety. Then, we evaluate the accuracy of our model in a variety of scenarios. Through this, we are able to better understand the importance of data, strengths and weaknesses of our model, and ways to maximize performance. Overall, we show our key hypotheses to be valid, representing significant advances to the study of gaze modeling.

5.1 Data Composition and Analysis

We gather GazeCapture recordings from **1025 subjects**. See Figure 5.1 for sample frames (in portrait orientation only). This represents a significant improvement over existing datasets, as shown in Table 5.1.

Using a subset of subjects (see Section 5.2), we examine the composition of our dataset. A demographic survey was given to crowdsourced participants, but it was optional (see

	Participants	Poses	Targets	Illumination	Images
McMurrough et al. [2012]	20	1	16	1	videos
Weidenbacher et al. [2007]	20	19	2–9	1	1,236
Smith et al. [2013]	56	5	21	1	5,880
Mora et al. [2014]	16	cont.	cont.	2	videos
Sugano et al. [2014]	50	8 + synth.	160	1	64,000
Zhang et al. [2015]	15	cont.	cont.	cont.	213,659
Huang et al. [2015]	51	cont.	35	cont.	videos
Ours	**1025**	**cont.**	**13 + cont.**	**cont.**	**1,610,401**

Table 5.1: Comparison of our Mobileyes dataset with other publicly available datasets. Mobileyes has approximately 20 times as many participants as the largest existing datasets. Furthermore, it contains a significant amount of variation in pose and illumination, as it is recorded via crowdsourcing without human supervision. We use the following abbreviations: *cont.* for continuous and *synth.* for synthesized.

Section 3.3.1); thus, we use human annotations on this subset. Of the 787 subjects—393 male and 394 female—142 were wearing glasses. There were 664 iPhone users and 123 iPad users. We examine the impact of this iPhone–iPad imbalance in Section 5.4.3.

We wish to show that our dataset offers more variation than other datasets. Since we ask subjects to move their heads around during the recording, we expect to see more variety in head pose. As is done in MPIIGaze [Zhang et al., 2015], we estimate head pose ($\mathbf{h}$) to compare the range of variation of our dataset with other datasets. We also compare the range of gaze direction relative to head pose ($\mathbf{g}$). Results, shown in Figure 5.2, indicate more variety over both $\mathbf{h}$ and $\mathbf{g}$. We also observe this variation qualitatively with an animated version of Figure 5.1.

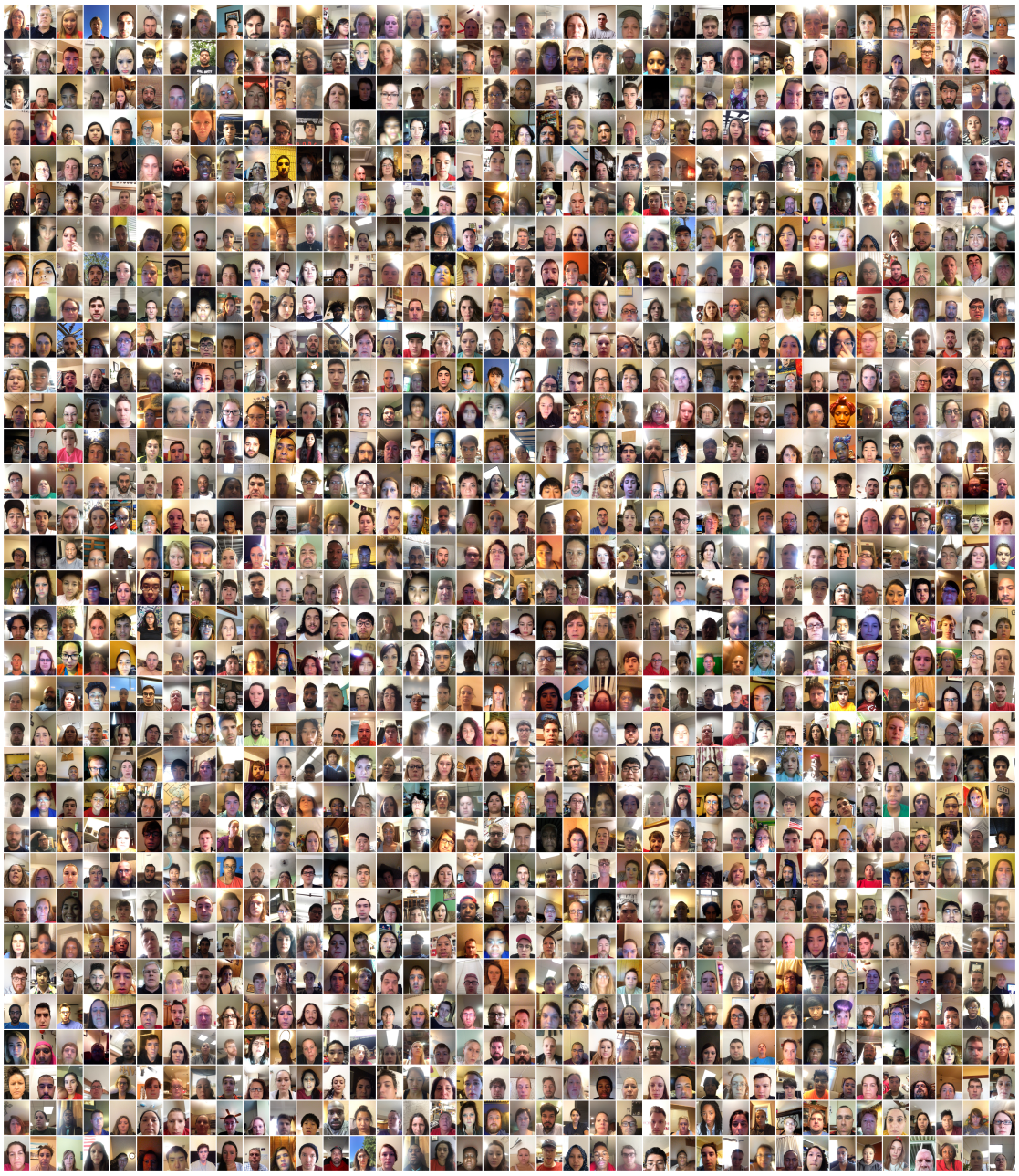
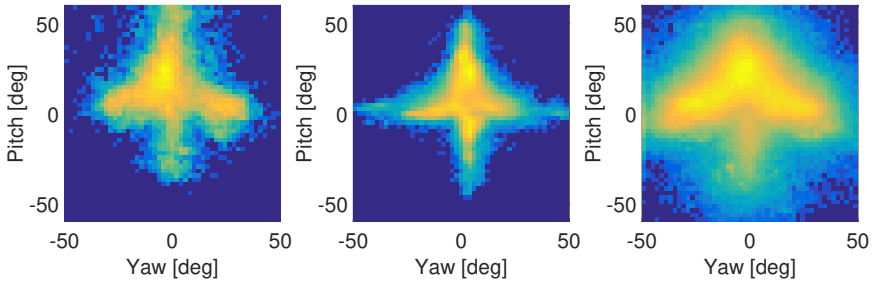
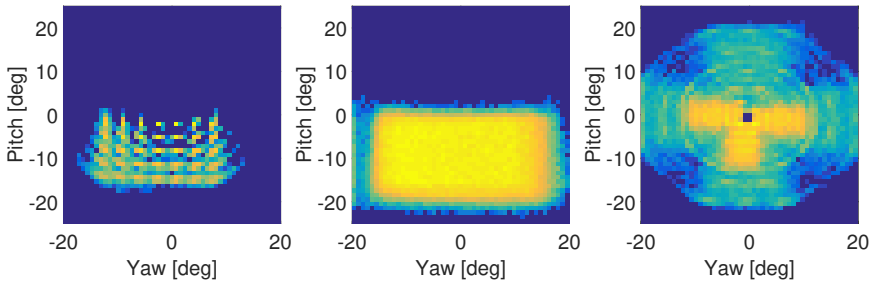


Figure 5.1: A preview of random frames (the first recorded in portrait orientation) from our Mobileyes dataset. Note the significant variation in illumination, head pose, appearance, and background. This variation allows us to learn robust models that generalize well to novel faces.



(a) **h** (TabletGaze) (b) **h** (MPIIGaze) (c) **h** (**Mobileyes**)



(d) **g** (TabletGaze) (e) **g** (MPIIGaze) (f) **g** (**Mobileyes**)

Figure 5.2: Distribution of head pose **h** (*1st row*) and gaze direction **g** relative to the head pose (*2nd row*) for datasets TabletGaze [Huang et al., 2015], MPIIGaze [Zhang et al., 2015] and Mobileyes (Ours). All intensities are logarithmic.

5.2 Setup and Baselines

In this, and following sections, we explore in-depth the performance of our model in terms of mean error. Traditionally, eye tracking solutions measure gaze prediction error in an eye-centric manner using angular deviation, measured in degrees. In such a scenario, gaze is thought of as a 3D vector indicating the direction in which the eye is pointing. These 3D predictions can be made for both eyes, then intersected with the screen and averaged to infer the point of regard.

In our end-to-end model, we expect predictions to be made directly on the screen

plane, so like other similar approaches (e.g. Huang et al. [2015]), we primarily evaluate error in the 2D domain. Specifically, we measure the error in centimeters to compare across different screen coordinate systems. We also predict error in degrees for comparability with other angular gaze measurements. This is done using estimates of eye position relative to the device in 3D space.

For our experimental results, we make use of the TabletGaze dataset [Huang et al., 2015] and 787 subjects with complete uploads from our Mobileyes dataset.

On the TabletGaze data, we use the same 41 subjects used for the evaluation in Huang et al. [2015] (with the exception of one subject recording which was unavailable at the time). We randomly select eight subjects for testing and use the remaining 32 for training. Eye crops are obtained using the same method as Huang et al. [2015] and head detections are inferred from the geometry of the eyes.

For our Mobileyes dataset, we select 789 subjects, but two of the subjects are in such dark conditions that the eye detector provides no detections. The recording is stopped if no face is detected during a dot recording, but the face detector used is very robust to low illumination. Thus, of the remaining 787 subjects, 80 subjects are held out for testing and the other 707 are used for training. We eliminate frames lacking either face detections (7.8% of selected frames) or eye detections (39.6% of selected frames) leaving 884,216 frames.

We do not discard any subjects with high failure rates on the attentiveness task (see 3.1.2). Dots with an incorrect response are recorded again in the app and we do not notice significant correlation between a subject's failure count and their mean error.

We evaluate recent work and some naïve approaches using the TabletGaze dataset.

Insights from this comparison are used to guide the design of our model. Some key baseline results are highlighted in Table 5.2.

Notice the state-of-the-art performance using only features from ImageNet [Krizhevsky et al., 2012], a deep CNN that was trained to perform a completely different task (object classification)! In this way, features from the ImageNet `conv3` layer can be considered to be good generic feature descriptors. This approach is unwieldy to train due to the large size (64,896 dimensions) of the `conv3` layer (even on a machine with 220 GB of main memory). To train the SVR model with two eyes from `conv3` plus face features, we modify the liblinear [Fan et al., 2008] source code to work with a smaller data type.

Baseline Method	Error (cm)
TabletGaze (as reported in Huang et al. [2015], using leave-one-out cross validation)	3.1
TabletGaze [Huang et al., 2015]	4.04
MPIIGaze [Zhang et al., 2015]	3.63
TurkerGaze [Xu et al., 2015] raw pixel features + SVR	4.77
Center	7.54
ImageNet [Krizhevsky et al., 2012] features (eyes (`conv3`), face (`fc6`), and face grid) + SVR	3.09

Table 5.2: Baseline approaches using our train/test splits on the TabletGaze [Huang et al., 2015] dataset (unless otherwise specified). In Section 5.6, we show how we use our model to achieve an error of **2.80 cm** on TabletGaze data. See Table A.1 for an extended version of this table.

5.3 Error Metrics

Our standard evaluation metric is the mean Euclidean distance between the predicted location and the ground truth location (in 2D). We also evaluate x and y error separately.

Regardless of device orientation, x describes horizontal error and y describes vertical error in physical space. Unless otherwise noted, all measurements are in centimeters.

While our model works on each sample independently, real-world applications may use a video stream, which is likely to have a strong correlation between temporally-local frames. In such scenarios, solutions such as temporal smoothing (e.g., the Kalman filter) can help improve both error and the perceived stability of the system. As a rough estimate of the improvement potential, we incorporate a metric we call *dot error*. For this metric, we use ground truth labels to average all predictions for a single dot (and a single participant) together before computing error. The use of ground truth data limits the practicality of this metric in a real-world environment, but it provides valuable insight into the stability of our system.

5.4 Model Results

We evaluate our model using our held-out Mobileyes test data. By comparing different variants of our model, we show the relative merit of the design decisions outlined in Chapter 4. Although we design for a fully unconstrained solution, we also explore ways to improve our model in different settings.

5.4.1 Data Augmentation

iTracker was trained on the augmented Mobileyes dataset (see Chapter 4). In Table 5.3, we show the benefit of augmentation during training (compare "no augmentation" with "augmentation on train"). We also show how error can be improved by augmenting test

Model	Error	x Error	y Error	Error (deg.)	Dot Error
iTracker, no augmentation	2.46	1.41	1.69	2.53	2.21
iTracker, augmentation on train	2.22	1.21	1.58	2.37	2.07
iTracker, augmentation on train/test	2.16	1.17	1.54	2.30	2.06
iTracker, augmentation on train/test with device fine-tuning	2.05	1.13	1.44	1.61	1.89

Table 5.3: Results for our uncalibrated iTracker predictions. Fine-tuning results are described in Section 5.4.3; other results are described in Section 5.4.1. All error measurements are in centimeters unless otherwise specified.

samples, then averaging multiple network outputs to produce a single prediction (see Section 4.4; referred to as "augmentation on test" in Table 5.3).

5.4.2 Calibration

Eye tracking error can be improved on a per-user basis through calibration. By collecting ground truth samples of the user looking at a known location, the system can be tuned to an individual. Typically, calibration is a formal procedure, although incognito calibration may be applied in certain contexts.

We apply calibration for all subjects in our test set using 13 fixed dot locations (depicted in Figure 3.1). Along with random dot locations, the GazeCapture app displays these 13 dots in each recorded orientation. We train a support vector regression model [Fan et al., 2008] using features extracted from the penultimate layer of our network (`fc1`). All samples (augmented by a factor of 25 as described in Section 4.4) at calibration points are used for training. All samples at non-calibration points are used for evaluation (not augmented).

Calibration Points	Error (cm)	Dot Error (cm)
4	2.30	2.30
5	2.10	1.97
9	1.92	1.72
13	1.82	1.64

Table 5.4: Effect of calibration on error using our model. The error reduces as more points are used for calibration.

In Table 5.4, we show improved performance after each user goes through a 13-point calibration. We also evaluate subsets of the 13 points: border dots and the center dot (nine dots), corner dots and the center dot (five), and only corner dots (four). The evaluation set remains fixed throughout (i.e., all points not at one of the 13 calibration locations).

5.4.3 Device Fine-tuning

Training a single model to make predictions for different device types is convenient and hopefully more generalizable to new domains; however, in many cases, optimizing a model to one specific device would be a feasible way to reduce error. We fine-tune seven copies of our model to optimize performance for each device type in each supported orientation. We start with our model that was trained on multiple device types, and resume training again using only samples for the specific device and orientation.

Results are shown in Table 5.3 and Table 5.5, with averaged augmented predictions in the test set. iPhone error improves to 1.88 cm and iPad improves even more, to 3.53 cm. We analyze these results further in Section 5.5.2.

Data Split	Error	Error (deg.)	Dot Error
iPhone Portrait	1.99	2.15	1.84
iPhone Landscape (Camera on Left)	1.86	2.06	1.67
iPhone Landscape (Camera on Right)	1.78	1.97	1.62
iPad Portrait	3.08	2.95	3.26
iPad Portrait Upside Down	4.02	3.73	3.60
iPad Landscape (Camera on Left)	3.55	3.65	3.48
iPad Landscape (Camera on Right)	3.45	3.29	3.58
Weighted average:	**2.05**	**1.61**	**1.89**

Table 5.5: Results for our iTracker model fine-tuned to specific devices/orientations.

5.5 Model Analysis

In this section, we analyze various aspects of our model and its relationship to our data. By exploring the results in detail, we gain a better understanding of how our model works. We also explain how we can leverage our insights to improve performance.

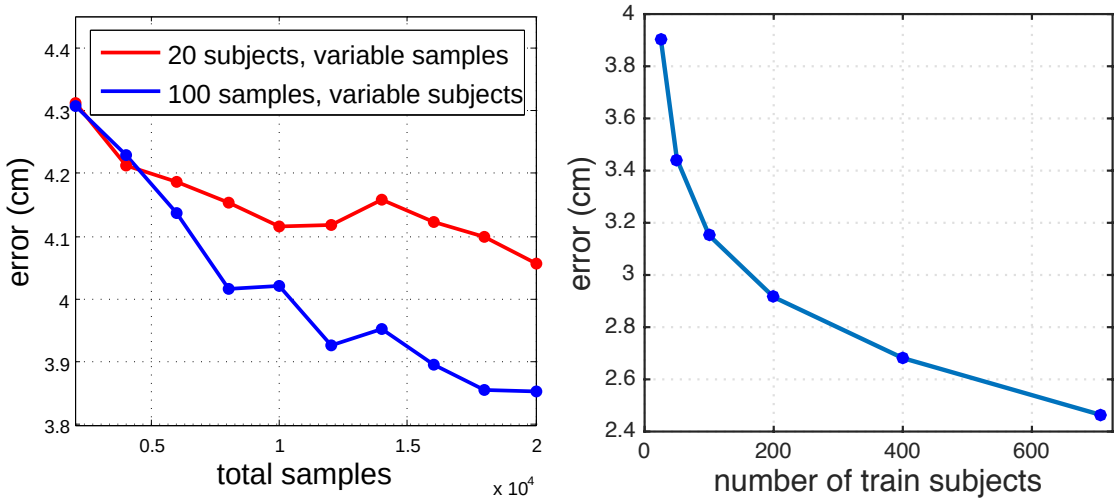
5.5.1 Dataset Size

We use crowdsourcing to collect data because of its potential to reach a large number of subjects. We distinguish this from the goal of simply collecting a large number of samples (e.g., Zhang et al. [2015]). Indeed, as shown in Table 5.3, augmenting the number of samples (even artificially) is valuable in terms of reducing error, but we argue that the number of subjects is even more important.

To validate this claim, we train our model from scratch on different subsets of data. In Figure 5.3, we grow one subset by the number of samples per subject (keeping the number of subjects fixed) while we grow the other subset by the number of subjects (keeping the

number of samples per subject fixed). In both cases, our fixed test set is used to compute error. Figure 5.3 demonstrates a clear advantage to collecting data from many subjects.

In the same figure, we also show the overall trend in adding participants to our dataset. The initial impact is largest, but we are encouraged that continued crowdsourcing of data could continue to have a positive effect on our model.



(a) Subjects vs. samples

(b) Number of subjects vs. error

Figure 5.3: Dataset size is important for achieving low error. Specifically, growing the number of subjects in a dataset is more important than the number of samples, which further motivates the use of crowdsourcing.

5.5.2 Error Maps

To better understand the source of error in our model, we plot a heat map (see Figure 5.4) of error at different spots in our prediction space. From this figure, we observe that the error is highest where the dot is farthest from the camera (which is in the white area in the middle of the plot). While there are theoretical explanations for inaccuracies at the

extremities [Hansen and Ji, 2010], we suspect insufficient training data to be the primary cause. These areas are only covered by iPad samples, which account for only 15.6% of our entire dataset. This lack of iPad data, particularly in the test split, is visible in the patchy nature of the plot.

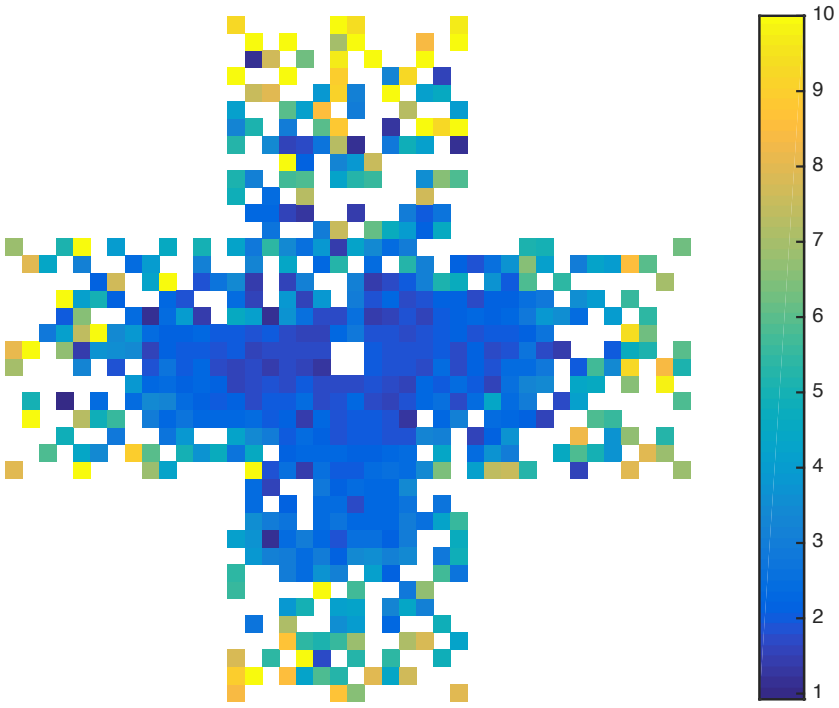


Figure 5.4: Error across the prediction space, plotted at ground truth location. We observe that the error near the camera tends to be lower as the darker regions correspond to the iPhones, for which we have significantly more data. The outer parts with greater error correspond to the iPads, where our data is rather limited. Error is clamped to 10 (i.e., $error = min(error, 10)$) to show more color variation on the low end of the scale.

We confirm a disparity between iPhone and iPad prediction quality by evaluating

error separately. Whereas, for the entire dataset, the mean error (averaging augmented test predictions) is 2.16 cm (see Table 5.3), iPhone error is 1.94 cm, and iPad error is 4.14 cm. This can be explained by the network learning to prefer predictions close to the camera due to the infrequency and high prediction error of points farther away.

In Section 5.4.3, we show how fine-tuning to specific devices and orientations can be helpful; however, iPad error is still significantly worse. The most improvement is likely to come from collecting additional iPad data.

5.5.3 Best and Worst Samples

In Figure 5.5, we visualize samples which have the lowest and highest prediction error. We only select one sample for a given dot due to the high correlation between frames; however, subjects may appear multiple times. Best-scoring samples are on the top row and the worst-scoring samples are on the bottom row. We observe that the worst-scoring samples tend to have severe variation in lighting conditions, heavy blur, and reflections from glasses. Furthermore, they tend to have head poses which occur less frequently in our dataset for a given gaze point.

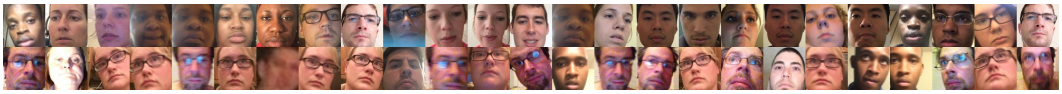


Figure 5.5: Samples with the lowest (top row) and highest (bottom row) prediction error in our dataset. Only one sample is shown for a given dot, as we assume strong correlation between frames. Both rows are sorted from lowest error to highest error. For the worst-scoring samples (bottom row), notice the extreme examples of poor illumination, blur, reflection from glasses, and head pose.

5.5.4 Error Visualizations

We also visualize individual predictions alongside their associated ground truth points and input frames to better understand how our model works. See Figure 5.6 for select samples and predictions from our original model (with augmented training data only; see Section 5.4.1) and our fine-tuned model (for specific devices/orientations; see Section 5.4.3).

Through this analysis, we gain many insights. First, we estimate human accuracy to be low, especially without the context of other similar frames. This demonstrates the difficulty of unconstrained gaze estimation. We also notice how certain subjects have certain trends in the error (e.g., bias towards the center of the screen or towards the camera). Some of these can be explained visually (e.g., closed eyelids often produce predictions lower on the screen); in such cases, the value of calibration is clear (see Section 5.4.2).

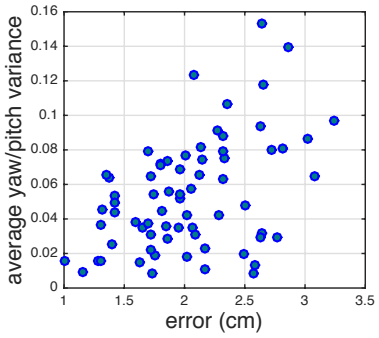


Figure 5.7: We find that the amount of subject head motion is related to their average error. Head motion is computed by averaging the variance of estimated head yaw and the variance of estimated head pitch.

5.5.5 Per-subject and Per-device Analysis

To better understand which characteristics of a participant make for better or worse results, we analyze different user traits. In Figure 5.7, we show how the relationship between the amount of head motion (quantified by the averaged yaw and pitch variance produced by the head pose model of Zhang et al. [2015]) and mean error for a subject.

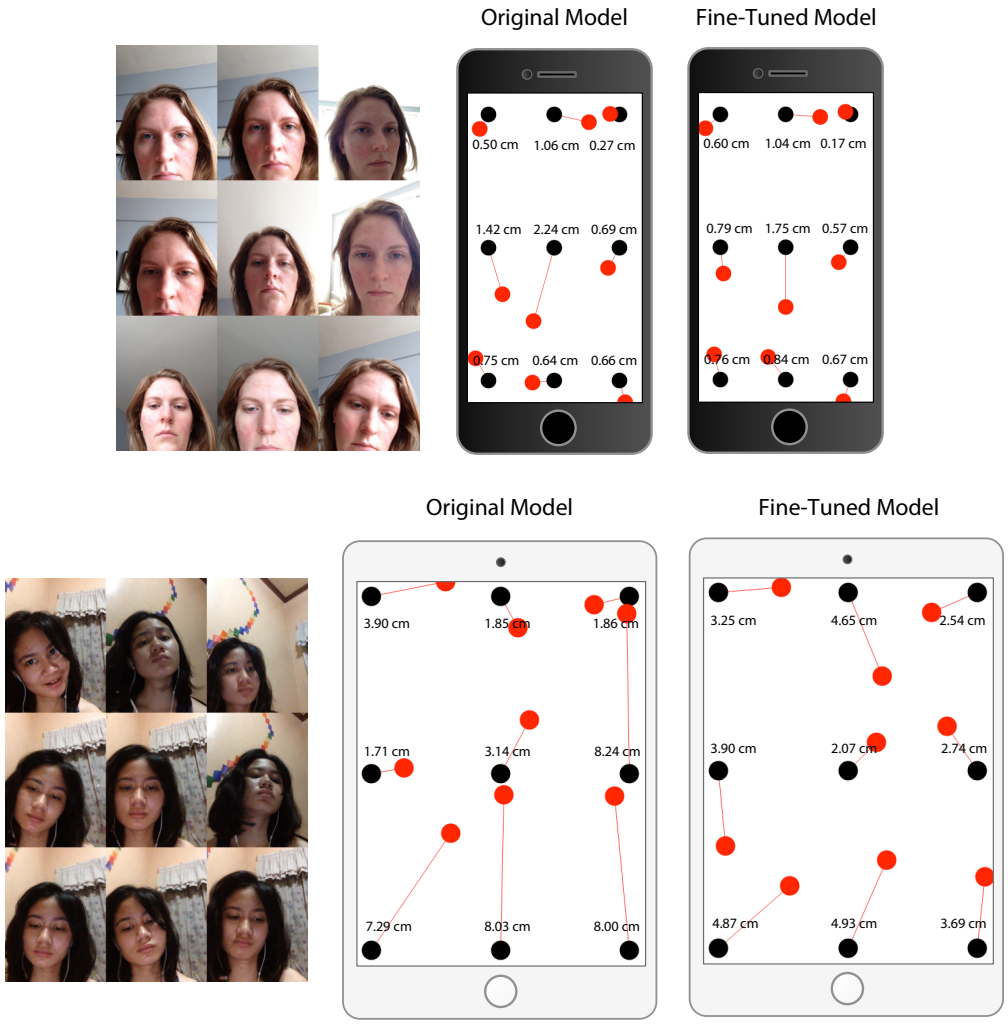


Figure 5.6: Error visualizations for select subjects in our test set. Black dots are at ground truth dot locations while red dots are predicted locations. The screens are mirrored so that subjects appear to look at the dots. Nine calibration points are used for ground truth locations and the associated input frames are shown in the same relative position to the left of the devices. The left device shows predictions from our original iTracker model (trained on augmented data, but not tested on augmented data; see Section 5.4.1). The right device shows predictions from our fine-tuned model (for specific devices/orientations; see Section 5.4.3). The top subject achieved a mean error of 1.32 cm on an iPhone 5c. The bottom subject achieved a mean error of 3.64 cm on a fourth-generation iPad.

Average subject error (without calibration) ranges from 1.00 cm to 7.81 cm with a median of 2.08 cm. As expected (see Section 5.5.2), nearly all iPad users score poorly, although the best iPad error is 2.09 cm. The top three subjects with lowest prediction error all use an iPhone 5c, although other iPhones achieve a better overall average. A larger test set would be necessary to make further observations regarding specific device models.

5.5.6 Ablation Tests

We perform ablation tests on our model to understand the importance of the various inputs. We remove the different inputs, one at a time and observe their impact on the performance. Whereas our full model achieves an error of 2.16 cm (see Table 5.3), training without a face image increases the error to 2.21 cm, and training without a face grid yields an error of 2.38 cm. This demonstrates the importance of each input for optimizing the accuracy of our model and its ability to integrate information from a variety of disparate sources.

5.5.7 Network Visualizations

We can gain some additional insight regarding *what* and *how* the network has learned by visualizing different layers of the network. In Figure 5.8, we show the progression of left eye inputs from an early layer (`pool1_1`, on the top row) to a later layer (`conv4_1`). We generate these images by examining the output of the selected layers while feeding images in our test set through our CNN. We save, normalize, and display the output that causes the maximal activation (i.e., the highest sum over all neurons) for selected channels.

We observe that the early layer helps to define important edges in the image. In the later layer (with lower-resolution images), we see indication in some channels that the pupil has been localized along with some key reference points on the eye. This information can help influence future network design changes (e.g., reducing or increasing the number of outputs for a certain layer).

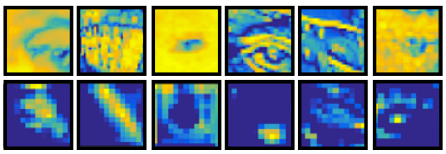


Figure 5.8: Visualization of network layers `pool1_1` (top row) and `conv4_1` (bottom row). These images are the normalized output of select neurons in the selected layers. The samples fed through the network are from the test set and we display only those which maximally activate the selected neurons.

5.6 Generalization Ability of iTracker Features

As we have shown, our deep CNN model is well-suited for using our data. We propose that our model has also learned in a way that generalizes well to new data.

To demonstrate this, we return to the TabletGaze [Huang et al., 2015] dataset. Due to the differences in the device used (and without exact measurements of the hardware), we do not expect our predictions to be immediately useful. However, we propose the use of the penultimate layer of our model (`fc1`) as a generic eye feature representation.

Because TabletGaze uses very tight eye crops, we expand the bounding box to reflect the type of input provided to our model. Then, after generating the other necessary inputs, we extract the `fc1` features for each sample. Training samples are used to train a support vector regression (SVR) model [Fan et al., 2008]. With an average prediction error of **2.80 cm**, the results of our model surpass all other baselines using TabletGaze

data. Thus, our model—even though trained on our own data—proves generally useful for gaze-related tasks.

5.7 Real-Time Inference

In Section 4.7, we describe how we can reduce the complexity of our model to improve performance in the resource-constrained environments associated with mobile devices. First, we justify the need for model optimization by timing our full iTracker model. A single forward pass takes on average 215 ms on a GPU. On a mobile device, where resources are far more constrained, this model's performance could not pass as "real-time."

By producing tighter crops and thus, smaller images, we gain significant improvement on our model execution time. Another key component is the adjustments made to the architecture; the original model has 8,152,602 parameters while the optimized model has only 589,666 parameters. With these changes, the single forward pass time is reduced to only 14 ms on a GPU. All this is done while maintaining an average of 2.83 cm prediction error. Without using dark knowledge [Hinton et al., 2015], the network takes longer to train and achieves 2.86 cm of error.

To derive an inference regarding on-device performance, we consider a deep learning framework optimized for iPhone ("Jetpac" with "DeepBeliefSDK"). They claim they can process an image through ImageNet [Krizhevsky et al., 2012] in under 300 ms on an iPhone 5s. Given that our model has approximately 100 times fewer parameters than ImageNet, we estimate an on-phone running time of no more than 50 ms.

Additional time is required to detect tighter crops, but integrated into our Gaze-

Capture pipeline, we find that the OpenCV detector (used to provide tight crops in TabletGaze [Huang et al., 2015]) can be run on an iPhone in under 15 ms on average. Thus, with a total of 65 ms required to process each frame, we estimate that we can achieve a frame rate of 15 frames per second using existing mobile technology. Furthermore, because the number of parameters in the network is significantly reduced, the memory footprint is far more reasonable than our original model.

Chapter 6

Conclusion

Eye tracking has a long history. Solutions for measuring and recording eye movements date back multiple centuries. Interest in studying eye movements goes back even further. With each advance, the ability to study and build on eye movements has improved. This work represents another step forward for eye tracking—hopefully one that will advance interest, research momentum, and practical applications.

We introduced an end-to-end eye tracking solution targeting mobile devices. First, we showed how a large-scale dataset could be collected via crowdsourcing. Crowdsourcing has never before been applied to collecting ground truth gaze data. We analyzed the resulting dataset to show how our collection methodologies produced data with good variety. We also showed how collecting data from more subjects was more important than simply collecting more data. This dataset represents a key contribution by virtue of its size, variability, and type of data (e.g., full camera frames, motion data).

This data was then used to train a novel deep learning model that was designed to

make accurate predictions in real-world conditions. The model is unique in its ability to learn from large-scale data. By not relying on hand-engineered features, we allow the model to learn the best representation possible. We evaluate the strengths and weaknesses of our model through a thorough evaluation and we demonstrate ways to reduce error even further. Finally, we consider ways to bolster performance, enabling real-time performance on existing mobile technology.

6.1 Future Work

The contributions of this work have an immediate impact—gaze predictions can be made from images in an unconstrained setting with unrivaled accuracy. In many ways, though, we anticipate the value of this work will be fully realized through future work building on our data and methodologies.

6.1.1 Reducing Error

In a qualitative evaluation of the TabletGaze model [Huang et al., 2015] which has an average error rate of over 3 cm, the observed accuracy in an unconstrained setting is still impressive. Relatively large targets (e.g., the screen divided up into four quadrants) could be hit with high accuracy, which may be enough for some applications; however, many applications require more accuracy. For example, to identify a standard app icon on a mobile phone, we estimate the average error should be well under 1 cm.

More Data

Our model hasn't achieved our desired error rate yet, but the plan to improve it is clear. As shown in Figure 5.3, collecting more data is a simple and reliable way to reduce error. We plan to continue crowdsourcing data as we have been. Focusing on recruiting participants with iPads (of which we have relatively few) will likely have the largest immediate impact due to the steep rate of change early on in collecting additional subjects.

The promising results from this work will likely make funding future data collection easier; however, recruiting participants requires considerable effort. Managing incoming data and addressing the problems and concerns of participants requires oversight. Furthermore, even with an incentive, finding a population with willing participants and valid devices can be difficult. We plan to explore crowdsourcing services other than Amazon Mechanical Turk and possibly promote the research where others may be interested.

6.1.2 Blink Detection

Many gaze estimation models (e.g., [Huang et al., 2015]) detect and remove closed eyes as a preprocessing step. For our model, we do not remove blinks. Blinks comprise a very small percentage of our overall dataset and we expect our model to treat such samples as noise during training. We propose that the ideal solution would be to incorporate a blink detector into our end-to-end model. In this way, we can avoid adding an additional detection step to the pipeline, and we expect our model should be able to achieve higher accuracy than many existing blink detection models, given our volume of data. Naturally, we would first need to acquire ground truth labels for blinks.

Better Architecture

Another way to improve our results is by tweaking the model. From tweaking the learning hyperparameters to experimenting with new architectures, we anticipate improvements could be made. A deeper analysis of what the network is learning could help guide these improvements. For example, if we were to determine that only certain neurons were actually useful in producing predictions, we could strategically reduce the size and complexity of the model.

Temporal Domain

Additionally, the model could be reworked to operate in the temporal domain. Whereas our current model works on an image-by-image basis (making independent predictions for each input) the expected usage is in a video setting. We expect multiple correlated predictions will be made back-to-back, so we recommend the use of temporal smoothing (see Section 5.3).

Perhaps a better approach though, would be to design the model such that it can learn temporal patterns in gaze data. *Recurrent neural networks* (RNNs)— specifically, long short-term memory networks (LSTMs)—have had much success as deep learning approaches to modeling temporal data. Such a network could be added to the end of our existing model or integrated directly. In this way, we anticipate our model could achieve an error rate more like that of the *dot error* described in Section 5.3 without using ground truth data.

Motion Data

In our dataset, we collect motion data from the sensors built into many modern mobile devices. Currently, we do not make use of that data in our model. The reasoning behind collecting such data was the thought that it might help describe the pose of the camera and thus better help gaze predictions. We could use this data as an additional input to our model, although it is somewhat of a deviation from our original goal of a camera-only approach. With additional data we expect the face input to be a strong predictor of head pose relative to the camera.

6.1.3 Expanding to Other Domains

Another goal is to extend our dataset and model beyond iPhones and iPads. Other mobile phones and tablets on the market could be used as well as webcams on laptop and desktop computers. Ultimately, we want our approach to generalize to work on any device.

A key challenge with this is dealing with the different poses of a camera relative to the screen, particularly if the webcam is not built into (and thus oriented perpendicular to) the screen. Determining screen size may be difficult in a generalized setting, particularly with crowdsourcing, but with a one-time calibration process, we could learn all necessary parameters. Generalizing beyond any specific set of devices would make for an even more useful all-purpose model.

6.1.4 Software Library

Once we achieve the desired error rate, we would like to fine-tune the performance (in terms of the computation and memory footprint) to build a live working demo of the model. We motivate this in Section 4.7 but we could perform a more in-depth and insightful analysis with a real on-device implementation. The plan would be to release the implementation as an open source software library. In this way, we anticipate developers and researchers will take this research beyond what we could imagine.

6.1.5 Theoretical Limits

As error decreases, we must start considering the theoretical limitations of our system. The primary issue is the quality of the ground truth labels in our dataset. First, even though we implement a quality control system that verifies that the participant saw each dot (see Section 3.1.2), it provides no guarantee. With a 50% chance of passing the verification task, a participant could get away without looking at the dot at all. Even a well-intending participant may grow uninterested in the task and inadvertently learn to look away until the time when the letter flashes. Furthermore, even with an attentive user, covert attention and fixational eye movements (see Section 1.2) may cause the eye position to be misaligned with the fixation point.

One solution would be to use a different approach to data collection. We believe the speed of our process and the volume of our data yields a high signal-to-noise ratio useful for training our model. We expect deep learning to deal with noisy samples well. However, even with a perfect system, the noise would show up as error during evaluation, so a different approach for evaluation could be useful. Such a dataset could make use of

reliable subjects to self-report fixations, as is suggested for calibration in Blignaut et al. [2014]. Inattentiveness, covert attention, and even fixational eye movements can be controlled voluntarily [Engbert and Kliegl, 2003]. This dataset would likely be smaller due to the challenge of finding reliable participants and may not accurately reflect an unconstrained setting, but is a worthwhile consideration for accurate error measurements.

6.2 Closing Remarks

The applications of unconstrained real-time eye tracking are far-reaching. By working to remove limitations of mainstream eye trackers, we hope to enable eye tracking for everyone. This would inspire innovators in ways we cannot imagine today; however, we are certain it would result in more novel data-driven solutions. Following the historical trend, we expect it would further progress our understanding of the the human visual system.

Though eye tracking has been around for centuries, we believe that this work will stand out as a landmark in the era of big data. We also hope our model and data will serve as a benchmark for the next generation of eye tracking solutions. Ultimately, the hope is that this work will enable the widespread use of eye tracking for a variety of novel applications never before possible.

Appendix A

Extended Results

Our iTracker model architecture is inspired by the results of an extensive baseline analysis (see Section 4.3). Certain key results are presented in Table 5.2, but we present all results in this appendix (Table A.1).

We observe that features from the `conv3` ImageNet [Krizhevsky et al., 2012] layer produce the best results compared to earlier (e.g., `pool2`) or later (e.g., `conv4`) layers. Also, we find that these approaches do not make much use of our face grid input (see Section 4.1) while it is important to our model (see Section 5.5.6).

Model	**Error (cm)**	***x* Error (cm)**	***y* Error (cm)**
Published Models			
Ours	**2.80**	1.56	1.99
TabletGaze	4.04	—	—
TabletGaze (leave-one-out cross validation)	4.05	—	—
MPIIGaze	3.63	—	—
TurkerGaze raw pixel features (normalized) [1]	4.77	2.96	3.12
Naïve Baselines			
Center	7.54	5.78	3.80
Gaussian with truncation (σ = 0.2)	8.75	6.75	4.40
Uniform random	10.02	7.70	4.95
ImageNet Left Eye (no truncation, no bias term)			
`fc6` (c = 0.01, no normalization)	5.00	3.67	2.68
`fc7` (c = 0.1, no normalization)	5.24	4.00	2.62
`fc6` (c = 1)	4.92	3.64	2.60
`fc7` (c = 100)	5.20	3.98	2.58
`fc6` with ridge regression (λ = 1000)	22.08	7.18	20.11
`fc7` with ridge regression (λ = 0.01)	15.43	5.27	14.01
ImageNet Left Eye (no truncation)			
`fc6` (c = 1)	4.90	3.61	2.61
`fc7` (c = 1000)	5.20	3.99	2.57
ImageNet Left Eye			
`conv3` (c = 100)	3.89	2.63	2.33

[1] Using SVR (c = 1000, vs. only 0.0001 and 1)

conv4 (c = 10000)	3.93	2.69	2.32
pool5 (c = 1)	4.27	2.98	2.46
fc6 (c = 10000)	4.83	3.56	2.55
fc7 (c = 1000)	5.17	3.97	2.54
ImageNet Right Eye			
conv3 (c = 10000)	3.64	2.38	2.24
conv4 (c = 1000)	3.68	2.42	2.24
pool5 (c = 1)	4.15	2.82	2.43
fc6 (c = 1)	4.76	3.46	2.55
fc7 (c = 100)	4.91	3.63	2.57
ImageNet Left Eye and Right Eye Concatenated			
conv3 (c = 1)	3.11	1.90	2.03
fc6 (c = 10)	4.11	2.81	2.40
fc7 (c = 100)	4.43	3.21	2.39
ImageNet Left Eye, Right Eye, and Face Grid Concatenated (face grid not normalized)			
fc6 (c = 100)	4.13	2.86	2.38
fc7 (c = 10000)	4.69	3.42	2.51
ImageNet Left Eye, Right Eye, and Face Grid Concatenated			
pool2 (c = 1)	3.18	1.92	2.10
conv3 (c = 1)	3.11	1.90	2.03
fc6 (c = 100)	4.03	2.73	2.38
fc7 (c = 10)	4.33	3.08	2.40
ImageNet Left Eye, Right Eye, Face Grid, and Face Concatenated			
left/right conv3, face conv3 (c = 10)	3.29	2.03	2.12

left/right `conv3`, face `conv4` ($c = 1$)	3.31	2.01	2.18
left/right `conv3`, face `pool5` ($c = 10$)	3.13	1.78	2.17
left/right `conv3`, face `fc6` ($c = 1$)	3.09	1.90	2.01

Table A.1: All baseline results used to inspire and evaluate our model. All results make use of our train/test splits on the TabletGaze [Huang et al., 2015] dataset (unless otherwise specified). Further, unless otherwise specified, ImageNet approaches use L2-normalized features, support vector regression (SVR), the best parameter chosen from $\{0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000\}$, and truncated predictions.

Bibliography

Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. Constrained local neural fields for robust facial landmark detection in the wild. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 354–361. IEEE, 2013.

Shumeet Baluja and Dean Pomerleau. Non-intrusive gaze tracking using artificial neural networks. Technical report, 1994.

Pieter Blignaut, Kenneth Holmqvist, Marcus Nyström, and Richard Dewhurst. Improving the accuracy of video-based eye tracking in real time through post-calibration regression. In *Current Trends in Eye Tracking Research*, pages 77–100. Springer, 2014.

Ali Borji and Laurent Itti. State-of-the-art in visual attention modeling. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, 2013.

Guy Thomas Buswell. *How people look at pictures*. University of Chicago Press Chicago, 1935.

Jixu Chen and Qiang Ji. 3d gaze estimation with a single camera without IR illumination. In *ICPR*, 2008.

Jixu Chen and Qiang Ji. Probabilistic gaze estimation without active personal calibration. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2011.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Robert Waring Darwin and Erasmus Darwin. New experiments on the ocular spectra of light and colours. by robert waring darwin, md; communicated by erasmus darwin, mdfrs. *Philosophical Transactions of the Royal Society of London*, 76:313–348, 1786.

Edward L Deci and Richard M Ryan. *Intrinsic motivation and self-determination in human behavior*. Springer Science & Business Media, 1985.

Edmund B Delabarre. A method of recording eye-movements. *The American Journal of Psychology*, 9(4):572–574, 1898.

Raymond Dodge. Five types of eye movement in the horizontal meridian plane of the field of regard. *American Journal of Physiology–Legacy Content*, 8(4):307–329, 1903.

Raymond Dodge. The participation of the eye movements in the visual perception of motion. *Psychological Review*, 11(1):1, 1904.

Raymond Dodge and Thomas Sparks Cline. The angle velocity of eye movements. *Psychological Review*, 8(2):145, 1901.

Andrew Duchowski. *Eye tracking methodology: Theory and practice*, volume 373. Springer Science & Business Media, 2007.

Andrew T Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*, 34(4):455–470, 2002.

Louis Émile Javal. Essai sur la physiologie de la lecture. In *Annales D'Oculistique*, 1879.

Ralf Engbert and Reinhold Kliegl. Microsaccades uncover the orientation of covert attention. *Vision research*, 43(9):1035–1045, 2003.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 2008.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2014.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Dan Witzner Hansen and Qiang Ji. In the eye of the beholder: A survey of models for eyes and gaze. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, 2010.

Dan Witzner Hansen and Arthur EC Pece. Eye tracking in the wild. *CVIU*, 2005.

Craig Hennessey, Borna Noureddin, and Peter Lawrence. A single camera eye-gaze tracking system with free head motion. In *ETRA*, 2006.

Ewald Hering. Der raumsinn und die bewegungen des auges. *Hermann L.: Handbuch der Physiologie III (Teil I)*, 1879.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

Philip S Holzman, Leonard R Proctor, Deborah L Levy, Nicholas J Yasillo, Herbert Y Meltzer, and Stephen W Hurt. Eye-tracking dysfunctions in schizophrenic patients and their relatives. *Archives of general psychiatry*, 1974.

Qiong Huang, Ashok Veeraraghavan, and Ashutosh Sabharwal. Tabletgaze: A dataset and baseline algorithms for unconstrained appearance-based gaze estimation in mobile tablets. *arXiv preprint arXiv:1508.01244*, 2015.

Edmund B Huey. Preliminary experiments in the physiology and psychology of reading. *The American Journal of Psychology*, 9(4):575–586, 1898.

Edmund B Huey. On the psychology and physiology of reading. i. *The American Journal of Psychology*, 11(3):283–302, 1900.

Edmund Burke Huey. *The psychology and pedagogy of reading*. The Macmillan Company, 1908.

Takahiro Ishikawa. Passive driver gaze tracking with active appearance models. 2004.

RJ Jacob and Keith S Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2003.

William James. The principles of psychology. 1890.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

Charles H Judd. The muller-lyer illusion. *The Psychological Review: Monograph Supplements*, 1905.

S Karthikeyan, Vignesh Jagadeesh, Renuka Shenoy, Miguel Eckstein, and BS Manjunath. From where and how to what we see. In *Computer Vision (ICCV), IEEE International Conference on*, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

M Lamare. Des mouvements des yeux dans la lecture. *Bulletin et Mémoire de la Societé Franç-aise d'Ophthalmologie*, 10:354–364, 1892.

Feng Lu, Takahiro Okabe, Yusuke Sugano, and Yoichi Sato. Learning gaze biases with head motion for head pose-free gaze estimation. *Image and Vision Computing*, 2014a.

Feng Lu, Yusuke Sugano, Toshiya Okabe, and Yuuki Sato. Adaptive linear regression for appearance-based gaze estimation. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, 2014b.

Päivi Majaranta and Andreas Bulling. Eye tracking and eye-based human–computer interaction. In *Advances in Physiological Computing*. Springer, 2014.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Christopher D McMurrough, Vangelis Metsis, Jonathan Rich, and Fillia Makedon. An eye tracking dataset for point of gaze detection. In *ETRA*, 2012.

Kenneth Alberto Funes Mora, Florent Monay, and Jean-Marc Odobez. Eyediap: A database for the development and evaluation of gaze estimation algorithms from rgb and rgb-d cameras. *ETRA*, 2014.

Carlos H Morimoto and Marcio RM Mimica. Eye gaze tracking techniques for interactive applications. *CVIU*, 2005.

Ulric Neisser. Cognitive psychology. 1967.

J Orschansky. Eine methode die augenbewegungen direct zu untersuchen. *Centralblatt für Physiologie*, 12:785–790, 1899.

Michael I Posner. Orienting of attention. *Quarterly journal of experimental psychology*, 32(1):3–25, 1980.

Keith Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 1998.

Adria Recasens*, Aditya Khosla*, Carl Vondrick, and Antonio Torralba. Where are they looking? In *Advances in Neural Information Processing Systems (NIPS)*, 2015. * indicates equal contribution.

Daniel C Richardson and Michael J Spivey. Eye tracking: Characteristics and methods. *Encyclopedia of biomaterials and biomedical engineering*, 3:1028–1042, 2004.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Weston Sewell and Oleg Komogortsev. Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network. In *SIGCHI*, 2010.

Brian A Smith, Qi Yin, Steven K Feiner, and Shree K Nayar. Gaze locking: Passive eye contact detection for human-object interaction. In *UIST*, 2013.

George Malcolm Stratton. Eye-movements and the aesthetics of visual form. *Philos. Stud.*, 1902.

Yusuke Sugano, Yasuyuki Matsushita, Yoichi Sato, and Hideki Koike. An incremental learning method for unconstrained gaze estimation. In *Computer Vision–ECCV*, pages 656–667. Springer, 2008.

Yusuke Sugano, Yuki Matsushita, and Yuuki Sato. Appearance-based gaze estimation using visual saliency. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, 2013.

Yusuke Sugano, Yuki Matsushita, and Yuuki Sato. Learning-by-synthesis for appearance-based 3D gaze estimation. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2014.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2014.

Kar-Han Tan, David J Kriegman, and Narendra Ahuja. Appearance-based eye gaze estimation. In *WACV*, 2002.

Earl A Taylor. *Controlled reading*. 1937.

Diego Torricelli, Silvia Conforto, Maurizio Schmid, and Tommaso D'Alessio. A neural-based remote eye gaze tracker under natural head motion. *Computer methods and programs in biomedicine*, 2008.

Roberto Valenti, Nicu Sebe, and Theo Gevers. Combining head pose and eye location information for gaze estimation. *TIP*, 2012.

Nicholas J Wade. Scanning the seen: vision and the origins of eye movement research. *Eye movements: A window on mind and brain*, pages 31–61, 2007.

Nicholas J Wade. Pioneers of eye movement research. *i-Perception*, 1(2):33, 2010.

Nicholas J Wade. How were eye movements recorded before Yarbus? *Perception*, 2015.

Nicholas J Wade and Benjamin W Tatler. Did Javal measure eye movements during reading. *Journal of Eye Movement Research*, 2(5):1–7, 2009.

U Weidenbacher, G Layher, P-M Strauss, and H Neumann. A comprehensive head pose and gaze database. 2007.

William Charles Wells. *An essay upon single vision with two eyes: Together with experiments and observations on several other subjects optics.* T. Cadell, in the Strand, 1792.

Pingmei Xu, Krista A Ehinger, Yinda Zhang, Adam Finkelstein, Sanjeev R Kulkarni, and Jianxiong Xiao. Turkergaze: Crowdsourcing saliency with webcam based eye tracking. *arXiv preprint arXiv:1504.06755*, 2015.

Alfred L Yarbus. *Eye movements during perception of complex objects*. Springer, 1967.

Dong Hyun Yoo and Myung Jin Chung. A novel non-intrusive eye gaze estimation using cross-ratio under large head motion. *CVIU*, 2005.

Laurence R Young and David Sheena. Survey of eye movement recording methods. *Behavior research methods & instrumentation*, 7(5):397–429, 1975.

Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 4511–4520, June 2015.

Zhiwei Zhu and Qiang Ji. Eye gaze tracking under natural head movements. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2005.

Zhiwei Zhu, Qiang Ji, and Kristin P Bennett. Nonlinear eye gaze mapping function estimation via support vector regression. In *Pattern Recognition*, 2006.