Uncovering Unintended Consequences: A Deep Learning Approach to Algorithmic Fairness in Eye Gaze Prediction

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Bachelors of Arts in the
Statistical & Data Science in the Department of
Mathematical & Computational Sciences at The College of
Wooster

by
Adam Meyer
The College of Wooster
2023

Advised by:

Dr. Christina Horr (Statistical & Data Sciences)

THE COLLEGE OF
WOOSTER

Abstract

Eye gaze is the observation and collection of images that show eye behavior, such as pupil dilation and movement, that help identifies a person's fixation on specific points concerning screen-based media. Eye gaze could be an essential tool that could be used across many industries, yet no technology exists on a large scale. One possible way to develop this technology is by using a supervised deep-learning model such as convolutional neural networks. I used a preexisting eye gaze dataset called GazeCapture to obtain predictions of the gaze locations along with obtaining algorithm fairness metrics for the race and gender variables that were added to the dataset using the FairFace demographic classification algorithm. It is important to discover the algorithm fairness of deep learning models because many deep learning models have been unfair and harmful to underrepresented groups. To test these metrics, I used image processing methods including RGB, grayscale, binarization, and edge detection to analyze how image processing methods have an effect on an algorithm's fairness.

I found that RGB image processing is the most accurate model for predicting the gaze locations with a prediction error of 2.7172 cm for mobile devices. Then when investigating algorithm fairness, I found that the Canny edge detection resulted in a fair gender algorithm that obtained a p-value of 0.94 when determining if the error prediction for males and females were different, while only decreasing the accuracy by 0.3292 cm. Additionally, the intensity grayscale method also resulted in a fair gender algorithm that obtained a p-value of 0.09. Given a p-value threshold

of 0.05, these results are significant and the method only decreases the accuracy by 0.0568 cm. Furthermore, no image processing method resulted in a fair race algorithm. Future work should include creating a dataset that includes race and gender demographics and has an equal number of participants and images for all racial and gender groups. The code and data files are available on my Github.

To Nana and Papa

Acknowledgments

I would like to express my gratitude to the individuals who have supported me throughout my academic journey and contributed to my completion of this study.

First and foremost, thank you to my family for the unwavering support, encouragement, and love that they have given me throughout my time at the College of Wooster. Their constant belief in my effort and commitment to my success has always been a source of motivation.

Additionally, I would like to acknowledge my advisor Dr. Christina Horr for invaluable guidance, support, and constructive criticism during the course of my research. Furthermore, I would also like to recognize Dr. Drew Guarnera, Dr. Sofia Visa, and Patrick May for their assistance in providing me access to the school's computer lab and the technical guidance in configuring the GPU. Without their assistance, I would not have been able to carry out my research.

Lastly, words cannot express my appreciation for my outstandingly awesome team of friends and teammates whose companionship has been invaluable during my four years at Wooster.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Topic of Interest: Eye Gaze and Algorithm Fairness

In 2016, about 3.6 billion people owned a smartphone. Today, it is predicted that that number has increased by an additional 3.2 billion [49]. This widespread use means that more advanced technology is on the way as these products become even more useful to support humans' everyday lives. Eye gaze, or eye tracking, is the observation and collection of images that show eye behavior, such as pupil dilation and movement, that help identify a person's fixation on specific points concerning screen-based media. These movements of the eyes are grouped into fixations and saccades. A fixation is a direct point or object that the gaze is focused on for a period of time, and a saccade occurs when the gaze changes quickly between two fixations [20]. Eye gaze is frequently used to develop a better understanding of human behavior and cognition. The ability to understand human behavior and cognition is useful to know because it can help improve society for the better. Currently, eye gaze technology is being developed to allow smartphone users to use their eyes instead of their hands to make interaction with these devices simpler. For instance, eye gaze is beginning to be used in more and more areas, such as by those with disabilities like cerebral palsy to help them communicate [41], play video games,

use YouTube, and interact in the world [9]. This technology has a powerful impact on these individuals' lives and those around them.

Eye gaze has exceptional potential for the future of technology. In the future, eye gaze could be used as a way for people to text on their cell phones without typing. It could be used by marketing teams to determine what customers are attracted to on their websites or in the store. It could be used to improve virtual reality headsets and video games to help the user feel like they are in the simulation. Lastly, social media companies could use eye gaze to determine what posts viewers are more attracted to by understanding where people are looking.

Presently, many eye gaze datasets can be used to create this technology, but no studies found have investigated the algorithm fairness of these datasets. Algorithm fairness is the study of understanding and correcting biases, or "the presence of systematic and repeatable tendencies in an algorithm to generate unequal outcomes and disparate impacts across population subgroups" [22]. In the past, many models have been found to be biased and discriminatory towards certain genders, races, and groups of people [43] [26] [39]. These types of biases are a concern because of their potential impact on human lives. Even datasets that do not include labels of gender, race, religion, etc. can still be biased because these biases are embedded in society.

1.2 Deep Learning Approaches

Data is now a key component across all sectors, which makes the analysis of that data even more valuable. Thus, I use an advanced technique in data science called deep learning. Here, I introduce an overview of deep learning, which will be discussed more in-depth in later chapters and then applied to my data. This technique allows

me to create a model that can discover more information about eye gaze than I can see by looking at the raw images.

Deep learning (DL) is a branch of machine learning (ML), which is a branch of artificial intelligence (AI). It is worth noting the difference between the three because they are often used interchangeably. This relationship is shown in Figure 1.1. AI allows machines to learn and process information similarly to a human. It is a field that combined computer science and data to solve problems. Similarly, ML does this too but additionally creates a system that allows the computer to accomplish different types of tasks such as regression, classification, and clustering. ML is the process of using data to learn from experience. More simply, as the computer processes more data, it will be able to predict more accurate results because it has more information. There are three types of machine learning; supervised, unsupervised, and reinforcement learning.

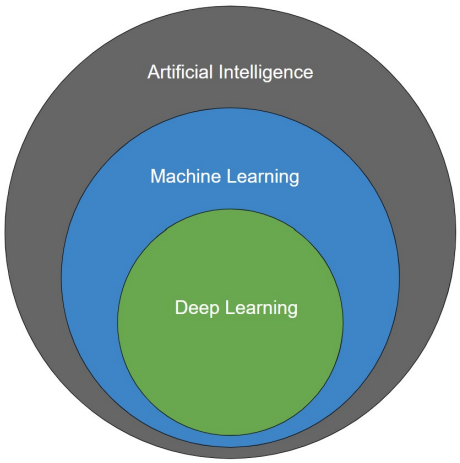


Figure 1.1: Relationship between artificial intelligence, machine learning, and deep learning.

In supervised learning, one first trains the model with input data that includes the correct classification of the category they are trying to predict. Then as the system computes the data it can discover information based on patterns that occur consistently within the dataset. Next, one tests the model with testing data where the classifications of the category they are trying to predict are unknown or hidden

from the computer. Then using these predictions one can compare them with the actual classifications to determine how accurate the model is. Increasing the amount of data that trains the model can lead to more accurate predictions. There are many different supervised learning methods, one of these being DL.

Alternatively, unsupervised learning is when an algorithm is trained with an unlabeled dataset, such that the computer does not know the correct classification of the category it they are trying to predict. And reinforcement learning is when an algorithm is not trained with any dataset, but instead, it learns from the experience it gains when working within an environment.

DL is a new type of supervised learning that has become increasingly useful in the last 10 years due to technological advances. DL also creates a system for the computer to make, but it is based on a neural network. Neural networks mimic the human mind by creating connections between input data and applying a weight to the input to create an output. And like the ML model, the DL model also requires a large amount of data to train on so that the model can make well, informed predictions.

1.3 Outline of Study

In this chapter, I have introduced the topic of eye gaze and my motivation regarding algorithm fairness. Additionally, I talked about how eye gaze is applied in current technology and the future uses of eye gaze. Then, I explored the overview of deep learning and how it is a small part of machine learning. Next in Chapter 2, I will discuss the literature on the eye gaze field, the types of datasets that already exist, and what methods scholars use in their studies. Additionally, I will discuss algorithm fairness. Then, in Chapter 3, I will explain how I will measure algorithm fairness in my study. Then, I will go further in-depth on deep learning

and explore the methods of image processing. And I will provide examples of these methodologies and how I put them into practice. After that, in Chapter 4, I will investigate the dataset of interest, called GazeCapture, the transformational procedures needed to conduct my study, and the exploratory data analysis of the GazeCapture dataset. Then, Chapter 5 explains the implementation requirements needed to perform the code and the architecture of the model used in the study. After that, Chapter 6 displays the results of deep learning. More specifically, I will discuss the accuracy and the algorithm fairness of the model. In Chapter 7, I will discuss the results found in the previous section by comparing the accuracies and algorithm fairness, as well as providing limitations and potential topics for future research. Lastly, in Chapter 8, I conclude my study by sharing the key finding and discussing the next steps for research in this field.

CHAPTER 2

Literature Review

2.1 Eye Gaze

There is a plethora of research in the eye gaze field including surveys on the different research [38] [16] [20], different eye gaze datasets, and ways in which scholars have used eye gaze to improve other fields. Eye gaze has mostly been used to develop advanced technology. Zhang et al. created a human-computer interface system that uses eye gaze as a mouse. The system incorporates keyboard functions so that users can achieve almost all their inputs without having to touch the physical keyboard [53]. Eye gaze is also being applied within the medical field. For instance, Stember et al. compare differences in using eye gaze and hand annotations to make medical diagnoses of brain meningioma [45]. Additionally, eye gaze is also incorporated into sports. For example, Panchuk and Vickers use eye gaze to conclude that a hockey goaltender's ability to make a save is highly dependent on the ability to maintain a long fixation on the puck [36]. And similarly, Cesqui et al. claim that when catching one-handed, such as in baseball, the ability to track the ball longer increased the catching probability [3]. Lastly, eye gaze can also be applied to the field of communication. For instance, Ito and Knoeferle investigate how speech rate affects eye gaze [13]. These studies highlight how extensive eye gaze is incorporated

into various fields of research on their potential for making valuable insights. Yet, it is worth noting that none of them currently address the algorithm fairness of their results or the datasets used. Hence there is a great deal to be studied. To help assist in further investigations of eye gaze, many datasets of been created. In the next section, I will go in-depth on some of these datasets in the eye gaze field by discussing how they were made, the types of models they were trained on, and the image resolution of the images in the dataset.

2.1.1 Datasets

To date, numerous large-scale eye gaze image datasets have been produced. All of these datasets fall into two categories of environments: controlled and wild. A controlled environment means that the images were taken in a laboratory with researchers guiding the participant through the process and making sure that they were following the instructions. This process ensures that data from each participant is consistent and limits the possible opportunities that could lead to errors. With an instructor in the room, it is less likely for the participant to be distracted. If the participant is distracted then they might lose focus, and therefore, they might not be looking at the correct point they are supposed to be looking at.

Additionally, some of these controlled environments had the participant's head fixed into a position with a chin rest [47] [19] so that each image had the same head position for consistency. Not all controlled environment datasets followed this technique. RT-GENE instead used a pair of eye-tracking glasses to calibrate where the participant was looking [6]. And ETH-Gaze took images that allowed the participant to move their head around to different positions to give the dataset more variability [52]. Yet, having the dataset being created in a lab kept many other aspects similar for every image, such as the distance from the camera, brightness, and the resolution of the image.

Wild environments may include more variability as they allow for the differences in types of image resolution, head or body distance from the camera, background, and brightness. Unfortunately, this can lead to more errors within the data. For instance, the participants might not think about how brightness can leave a glare on the image. If the glare is covering the participant's face, then the algorithm could have a more difficult time making an accurate prediction of the gaze location. Additionally, the variability in these wild datasets could lead to distinct differences within the image. For example, if one image in the dataset has another face in the background, then the algorithm might have a hard time detecting where the main participant is looking since it could be using both faces to create its prediction. Even if the participant follows all the instructions they are given, there is still a possibility that they could create a distinct difference in their images from the rest of the participants.

As mentioned earlier, wild environments mean that there is no instructor with the participant walking them through the process and taking the photos. This could also lead to the participant not being focused on the task at hand, which could lead to them not looking at the point that they are supposed to be looking at. To prevent this, wild environment datasets developed programs that helped the participant focus more on the task at hand and limit the possible distractions. GazeCapture required the participant to turn their mobile device on airplane mode to ensure that the participant did not receive notifications that would change where their eyes were focused on. Additionally, they added a text feature that showed the letter L or R, which then required the participant to tap the left side of the screen for L or the right side of the screen for R to make sure that the participant was looking at the point of interest [23].

The wild environment dataset's range of variation introduces other variables to consider. For instance, Gaze360 has both indoor and outdoor images [18], which

leads to changes in the background and the brightness. Similarly, GazeFollow collected images from multiple datasets showing people performing everyday activities. They then used Amazon Mechanical Turk to identify where they believed the participant was looking [40]. This dataset had the widest range of variation in the background because no image had the same background. Researchers in the field argue that this type of wild environment method creates realistic variability in appearance and illumination [51] which makes them more suitable for day-to-day usage on mobile devices.

The largest of the datasets in terms of images is the NVGaze dataset, which includes both a synthetic dataset and a real-world dataset. The synthetic dataset includes computerized drawings of an image that illustrates the skin, sclera, visible sclera, iris, pupil, and corneal glints [19]. This dataset includes 2 million images while the real-world dataset with actual images of eyes had 2.5 million images. GazeCapture also has about 2.5 million images from 1474 participants [23]. This dataset has the largest number of participants compared to the other datasets and provides a wide range of variations of backgrounds. ETH-Gaze is the only other dataset with over 1 million images, but there are far fewer participants (110) compared to the GazeCapture dataset [52]. All other datasets, except Eye-Chimera, have around 100 thousand to 200 thousand images. Eye-Chimera is the smallest dataset and only includes 1172 images from 40 participants [7].

In the last 10 years, the convolutional neural network (CNN) has increased in popularity in all fields of image recognition. Most of the eye gaze datasets are trained and tested on CNNs, which will be explained more in the next section. For now, a CNN takes in an image to produce a predicted output. MPIIGaze, GazeFollow, NVGaze, RT-GENE, ETH-Gaze, and GazeCapture all use this method to make their predictions on the gaze direction, angle, or location. Gaze360 uses another type of method called long short-term memory, which is a type of recurrent neural

network. This model uses a CNN as a backbone of the network to predict the gaze direction [18]. UT Multiview uses a random forest algorithm, but their study was created before CNNs were beginning to gain popularity. Similarly, Eye-Chimera uses a different method called the Eye Accessing Cue model which comes from Neuro-Linguistics Processing theory. This model identifies the face, iris, and other eye landmarks to make a mental activity recognition prediction [7].

Table 2.1: Eye Gaze Datasets

Dataset Name	Year	Participants	Images	Pixel Resolution	Environment	Method
ETH-Gaze	2020	110	1,083,492	6000×4000	Controlled	CNN
NVGaze: synthetic	2020	35	2,000,000	1280×960	Controlled	CNN
NVGaze: real-world	2020	30	2,500,000	640×480	Controlled	CNN
Gaze360	2019	238	172,000	3382×4096	Wild	Long Short-Term Memory
RT-GENE	2018	15	122,531	1920×1080	Controlled	CNN
GazeCapture	2016	1474	2,445,504	640×480	Wild	CNN
MPIIGaze	2015	15	213,659	60×36	Wild	CNN
GazeFollow	2015	130,339	122,143	UNKNOWN	Wild	CNN
UT Multiview	2014	50	64,000	60×36	Controlled	Random Forest
Eye-Chimera	2013	40	1172	1920×1080	UNKNOWN	Eye Accessing Cue

Lastly, the pixel resolution of the images is important. A higher pixel resolution leads to more information and data in the images. The highest pixel resolution comes from ETH-Gaze at 6000×4000. Gaze360 has the next closest resolution of 3382×4096. Then, Eye-Chimera and RT-GENE have less than half the resolution of Gaze360 at 1920×1080. The two NVGaze datasets have different resolutions at 1280×960 for the synthetic dataset and 640×480 for the real-world dataset. GazeCapture also has the same resolution as the NVGaze real-world dataset. Finally, UT Multiview and MPIIGaze have the lowest resolutions at 60×36. All of this information on the dataset can be seen in Table 2.1.

I decided to use the GazeCapture dataset for my research because of its size and amount of variability within the dataset. Additionally, I believe that the wild environment method used by GazeCapute best suits what type of images an iPhone, or another mobile device, would use to make their predictions if eye gaze was used on a larger scale. The dataset will be described later in Section 4.

2.2 Algorithm Fairness

Algorithm fairness refers to the idea that an algorithm should treat all individuals or groups equally and should not discriminate against a particular group. It has become a crucial component in data science because of the widespread and significant impacts that results found from data have had on individuals and society. Recently many algorithms have come to light as being biased toward an underprivileged group. For example, COMPAS, also known as the Correctional Offender Management Profiling for Alternative Sanctions algorithm, predicted the likelihood that a defendant would become a recidivist, meaning that they commit another crime after being released from prison. The COMPAS model was biased against Black defendants and predicted higher recidivism rates for Black defendants

compared to White defendants [43]. Researchers found that the data that trained the model had a historical bias, meaning the model was trained on data that contained bias due to past practices or societal inequalities. In this case, the data was biased because there are more Black defendants in the past that recidivated, which therefore led this model to be biased in predicting that future Black defendants would also recidivate. Additionally, racial bias was also found within the US healthcare system when an algorithm was used to predict which patients would be more likely to need extra medical care. This algorithm heavily favored White patients over Black patients even when race variables were not included in the model. Researchers found that other variables used in the model, such as cost history, were highly correlated to race. The cost history variable was linked to historical bias that Black patients were not provided the best medical treatment in the past, which lead to a lower cost history. Alternately, White patients were given better treatment and therefore, they had higher medical costs histories.

Racial bias is not the only type of bias found in algorithms. Gender bias has been found in credit limit algorithms, search engine algorithms, and even hiring algorithms. Credit limit algorithms often allow men to have higher credit limits because data from the past shows that men tend to make more money. Also, in the hiring algorithm case, the algorithm produces biased results because the data that trained the algorithm was based on the unequal ratio of male-to-female applicant resumes submitted over the past decade [26]. Similarly, Prates et al. found algorithm bias against women when researching if Google Translate had tendencies towards using male pronouns over female pronouns [39]. In their study, they translated phrases like "He/She is an engineer" from a non-English language into English. These outputs tended to translate to using "He" pronouns when the job fell into a STEM (Science, Technology, Engineering, and Mathematics) occupation. Not only

does that prove that the male defaults are dominants, but it also creates troubling issues regarding stereotypes of what gender can be hired for certain jobs.

These biases cause the algorithms to have a serious effect on society because they create a systematic favoring of certain groups or individuals over others. This can happen in a variety of ways. Some of these ways include an unbalanced training dataset, the use of variables that are strongly connected with demographic descriptors, such as zip code [26], and generational biases within the data. These biases can result in unequal opportunities for underprivileged groups. Therefore, scholars must detect how fair the algorithm is so that it can be fair for future uses. In the next section, I will identify examples of how other scholars measure algorithm fairness.

2.2.1 Measurements

Many methods have been proposed in the literature to measure algorithm bias and fairness in machine learning. Most of these studies focus on facial recognition because there is a high priority in these algorithms being fair, but the ideas can apply to eye gaze since the ideas and training processes are similar. One approach, used by Cavazos et al. [2], used ROC (receiver operating characteristic) curves and AUC (area under the curve) tables to compare differences between recognizing Caucasian and East Asian faces. The curves helped show that racial bias was more accurate in identifying White people over other races. Krishnapriya et al. [24] also use ROC curves to compare verification rates in facial recognition. Verification rates refer to the proportion of faces correctly identified by the algorithm. In their study, they used four different algorithms, two of which performed best in recognizing Black participants, while the other two performed best on White participants. Givens et al. [8] also focused on facial recognition and race and gender biases. To measure if these variables affected the verification rate accuracy they used an ANOVA model.

Results found that race affected the algorithm fairness. More specifically, non-White subjects were easier to recognize than White subjects when training and testing on their own dataset. Then when testing on another facial recognition dataset they found contrasting results that were more accurate in recognizing White participants over non-White participants. Additionally, the model found that gender did not affect the model's fairness, meaning that there was no bias towards male or female participants.

Feldman and Peake [5] describe six different ways of measuring bias in deep learning. The first two methods are used when investigating at the individual level where similar individuals should show consistency by having similar predictions.

First, fairness through unawareness achieves fairness if the protected attribute A, such as gender or race, is not utilized to make predictions. This so-called "measurement" actually refers to the idea that the creator of the algorithm is aware that there is a possibility that the algorithm could be biased towards certain groups. While this is a helpful idea for the creator of an algorithm to remember, it can not provide statistical evidence that a model is fair or not.

The second, counterfactual fairness suggests that the predictions should be based on the idea that the participant belongs to a different group rather than the one that they are actually a part of. Meaning that when if the group was switched, then the predictions would remain the same. For instance, if the participant's race was changed from Black to White, then the gaze location accuracy would remain the same. It is used when "a predictor $\hat{Y}$ is counterfactually fair if for any attributes $X = x$ and sex $A = a$", such that:

$$P(\hat{Y}A \leftarrow a = y|X = x, A = a) = P(\hat{Y}A \leftarrow a' = y|X = x, A = a) \tag{2.1}$$

Then, the other four methods proposed by Feldman and Peake focus on group

fairness to make sure that the prediction probabilities are equal for all groups. From an eye gaze standpoint, this means that if people from one group tended to look at a certain point, the accuracy of that prediction would be equal to the predictions of people in the other group looking at that point. The first of these group fairness methods is called Statistical Parity Difference (SPD) which takes the probability of an outcome ($\hat{Y}$) for someone belonging to group A and subtracts the probability of an outcome for someone belonging to group B, such that:

$$SPD = P(\hat{Y} = 1|A = Group\ A) - P(\hat{Y} = 1|A = Group\ B) \tag{2.2}$$

For the model to achieve perfect fairness if SPD must equal 0. Next, Equal Opportunity Difference (EOD) is found by taking the probability of the predictor $\hat{Y}$ for a person from Group A and has a true label Y and then subtracting the same probability from the person from Group B. In other terms, you find the probability of the true positives of one group and subtract the probability of the true positives for the other group, such that:

$$EOD = P(\hat{Y} = 1|A = Group\ A, Y = 1) - P(\hat{Y} = 1|A = Group\ B, Y = 1) \tag{2.3}$$

where again the model achieves perfect fairness if EOD = 0. Then, the Average Odds Difference (AOD) of a model is found by:

$$AOD = 1/2[(FPR_{Group\ A} - FPR_{Group\ B}) + (TPR_{Group\ A} - TPR_{Group\ B})] \tag{2.4}$$

where FPR represents the false positive rate and TPR represents the true positive rate. And again, the model achieves perfect fairness if AOD = 0. Lastly, the Disparate Impact is given:

$$DI = \frac{P(\hat{Y} = 1|A = Group\ A)}{P(\hat{Y} = 1|A = Group\ B)} \tag{2.5}$$

where the model achieves perfect fairness if DI = 1.

All of these measurements are important because they provide a baseline to evaluate the fairness of predictive models. Additionally, they assist researchers by identifying potential biases in the models and judging the impact of the biases on certain groups. Furthermore, by using these measurements, models can be made to be more equal and just for underrepresented groups who have felt the hardest impact of the biases in the past.

Kulkarni et al. propose another method to measure the bias of the model. This method only looks at the accuracies of the predictions, α, such that [26]:

$$\alpha = \frac{n}{N} \tag{2.6}$$

where n is the number of correctly identified observations and N is the total number of observations. The accuracy is also known as the true positive of the predictions. Using these accuracies, Kulkarni et al. calculates the bias of the model given:

$$Bias = \alpha_{GroupA} - \alpha_{GroupB} \tag{2.7}$$

where if the accuracies of the two models are equal to each other, then there is no bias and the model is fair.

While most of those previous methods work well for discovering the algorithm fairness of classification problems, it doesn't help too much in my study since the model used in this thesis finds the error as the average Euclidean distance between the predicted gaze location and the actual gaze location. Some methods that have been proposed that are more suitable for these types of metrics include t-tests. For example, Prates et al. [39] ran a t-test to determine differences in the gender bias of Google Translate, which found that the algorithms favored the male default pronouns for jobs within STEM occupations.

In summary, the literature suggests that there are various methods to calculate algorithm bias and fairness metrics to help reduce the biases of machine learning algorithms. In the next section, I will discuss how I will measure algorithm fairness along with the methods of deep learning and image processing.

CHAPTER 3

Methods

This chapter will delve into the topics of how I measure algorithm fairness, deep learning, and image processing. All of these topics are crucial in understanding the backend mathematics that produced my results.

3.1 Algorithm Fairness

To measure algorithm fairness in this study, the model calculates the error as the average Euclidean distance in centimeters between the predicted and actual gaze locations. Additionally, it calculates this error for all groups in gender and race categories. To measure gender fairness, I will compare the male average and female average distance using a two-sample t-test which is used to determine if there is a statistically significant difference between the means of the two groups. The t-test requires three assumptions to be true. First, the observations in each sample are independent of the observations in the other sample. Next, the data is normally distributed. And lastly, both samples should have equal variance or at least approximately equal variances. I will calculate the t-test using [12]:

$$t = \frac{x_{males} - x_{females}}{\sqrt{\frac{(s_{males})^2}{n_{males}} + \frac{(s_{females})^2}{n_{females}}}} \tag{3.1}$$

Where s_{males} and $s_{females}$ are the sample standard deviations, x_{males} and $x_{females}$ are the sample means, and n_{males} and $n_{females}$ are the sizes of each of the male and female samples. Then using the two-sample t-test, I will find the corresponding p-value to determine if the two means are statistically different from each other, such that the hypothesis for gender fairness is written as:

Null Hypothesis: $\mu_{males} = \mu_{females}$

Alternative Hypothesis: $\mu_{males} \neq \mu_{females}$

Where μ_{males} and $\mu_{females}$ are the population means.

Then to measure race bias in the model, I must use a different method because there are more than two races within the dataset. The race variable is separated into 7 different races; White, Black, Latino, East Asian, Southeast Asian, Middle Eastern, and Indian. One should use the ANOVA test to determine if there is a statistically significant difference between the means of 3 or more groups of people. Since the race variable has 7 different groups I will use an ANOVA test to determine if the mean distances are statistically different. For ANOVA the exact requirements of independence, normality, and equal variance for a two-sample t-test must also be true. ANOVA calculates the F statistic using the mean square of the variance (MS) between the samples and within the samples, such that [12]:

$$F = \frac{MS_{between}}{MS_{within}} \tag{3.2}$$

To better understand how this equation works, I will begin by defining the basic variables that go into the other equations that make up the F statistic. First, k represents the number of different groups, so for my study looking at race $k = 7$. Next, n_j is the size of the j^{th} group. Then, s_j is the sum of the values in the j^{th} group. After that, n is the total number of all the values combined or the total sample size. Lastly, x is one value from one of the groups, and $\sum x^2$ is the sum of all the values

from every group combined. Using $\sum x^2$ one can find the between-group variability that is given:

$$SS_{total} = \sum x^2 - \frac{(\sum x^2)}{n} \tag{3.3}$$

Additionally, the variation within the samples is given:

$$SS_{between} = \sum [\frac{(s_j)^2}{n_j}] - \frac{(\sum s_j)^2}{n} \tag{3.4}$$

And then, the sum of squares that represents the variations within samples is given $SS_{within} = SS_{total} - SS_{between}$. Lastly, the mean squared variance between groups is found using $MS_{between} = \frac{SS_{between}}{k-1}$, and the mean squared variance within the group is found using $MS_{within} = \frac{SS_{within}}{n-k}$. Using the ANOVA test, I will find the corresponding p-value to determine if at least one of the seven means is statistically different, such that the hypothesis for racial fairness is written as:

Null Hypothesis: $\mu_{White} = \mu_{Black} = \mu_{Latino} = \mu_{East\ Asian} = \mu_{Southeast\ Asian} = \mu_{Middle\ Eastern} = \mu_{Indian}$

Alternative Hypothesis: At least one of the means is not equal

Through the use of the t-tests and ANOVA tests, I will be able to determine the fairness of the deep learning algorithms. Additionally, these tests will allow me to determine how different types of image processing affect the fairness of the algorithms. In the next section, I will discuss how these deep learning algorithms are made and the types of image processing used in this thesis.

3.2 Deep Learning

In this section, I first discuss the background of neural networks, how they work, and their application in a real-world example. Then I thoroughly explain the type of neural network used in my study, called a convolutional neural network, along

with other types of methods that are used in GazeCapture's iTracker model to help make informed predictions.

3.2.1 Introduction

Over millions of years, the brain has evolved and created neurons so that now it can identify images of people and objects instantly without any effort. While computers do not have a brain or actual neurons, we can create artificial neurons which are also known as perceptrons. A perceptron takes in multiple inputs, $x_1, x_2, x_3, ..., x_n$. The inputs are then weighted $w_1, w_2, w_3, ..., w_n$ based on their importance to the output. The output, a or b, which is some form of prediction is then determined given that

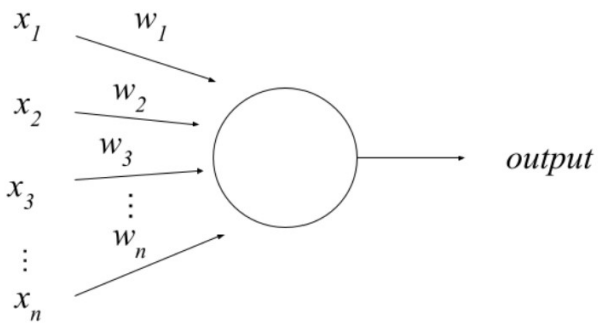


Figure 3.1: A perceptron with inputs $x_1, x_2, x_3, ..., x_n$ that are weighted $w_1, w_2, w_3, ..., w_n$ to create an output.

the weighted sum $\sum_j x_j w_j$ is less than or greater than the given threshold value. In more strict algebraic terms [32]:

$$\text{output} = \begin{cases} a, & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ b, & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{3.5}$$

An easy way to think about perceptrons is that they make their decisions by weighing in evidence to make their answer. To get a better understanding of how

perceptons work, I will provide an example. Suppose you are deciding to go to a baseball game or not. You make your decision by weighing four factors:

1. Are the tickets reasonably priced?
2. Is the baseball team good?
3. Can you make it in time for the opening pitch?
4. Are your seats in the shade?

We can rewrite these four factors by corresponding binary variables x_1, x_2, x_3, and x_4 and their weights then as w_1, w_2, w_3, and w_4. For instance, if the tickets are reasonably priced, then $x_1 = 1$, otherwise $x_1 = 0$. Next, if the team is good, then $x_2 = 1$, otherwise $x_2 = 0$. Similarly, if you can make it for the opening pitch, then $x_3 = 1$ and $x_3 = 0$ if not. And again $x_4 = 1$ if your seats are in the shade, and $x_4 = 0$ if not.

Now, imagine that you love baseball, so much that you would be willing to go to a game even if your team is not good. But you are hesitant to go to a baseball game if your seats are looking into the sun the entire game. Since you care the most about that factor, its weight, w_4, is the most significant and largest out of the other factors. In Figure 3.2 and Figure 3.3 we will denote it as $w_4 = 8$. As mentioned before, you do not care if the team is good so $w_2 = 1$. But, you are also slightly worried about getting to the game in time for the opening pitch and the cost of the tickets, so $x_1 = 3$ and $x_3 = 3$. And finally, suppose you choose a threshold of 7 for the perceptron, so our algebraic equation is:

$$\text{output} = \begin{cases} \text{do not go to the game,} & \text{if } \sum_j w_j x_j \leq 7 \\ \text{go to the game,} & \text{if } \sum_j w_j x_j > 7 \end{cases} \tag{3.6}$$

With these choices, the perceptron implements the decision-making model to determine if you go to the baseball game or not. If your seats are in the shade, meaning $x_4 = 1$ then you will be above the threshold and the other factors will not impact your decision because $x_4 * w_4 = 1 * 8 = 8$. Now, let's pretend that they are not

in the shade and are facing the sun. Additionally, the baseball team is good and you can make the game in time for the opening pitch, but the tickets are not reasonably priced. Therefore using the formula $x_2 * w_2 + x_3 * w_3 = 1 * 1 + 1 * 3 = 1 + 3 = 4$. Based on these factors you would end up not going to the baseball game because your output is 4.

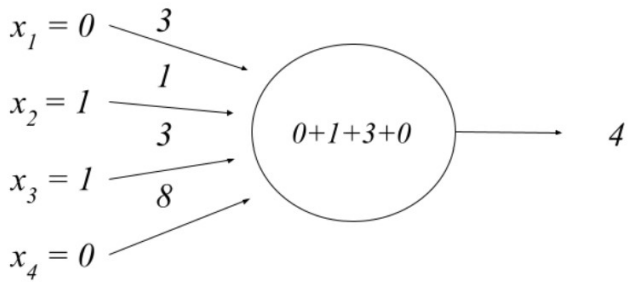


Figure 3.2: Baseball game decision perceptron with the input (0,1,1,0).

But if the tickets were reasonably priced, then you would go to the game because the formula would now be $x_1 * w_1 + x_2 * w_2 + x_3 * w_3 = 1 * 3 + 1 * 1 + 1 * 3 = 3 + 1 + 3 = 7$, where your output would be 7, as shown in Figure 3.3.

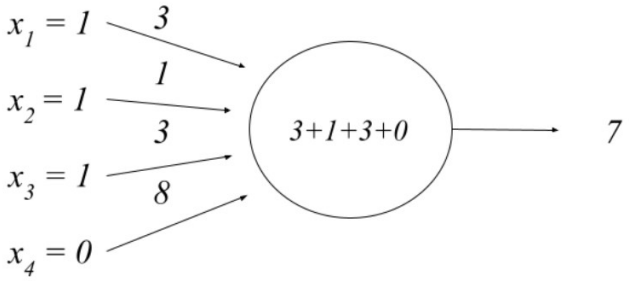


Figure 3.3: Baseball game decision perceptron with the input (1,1,1,0).

Many more factors go into making our decisions like going to a baseball game, but this simple perceptron model helps give a quick understanding of how the human mind makes its decisions.

Additionally, we can also add bias, b, to the perceptron. A bias in a neural network is a value that increases the probability of generating a certain output. The larger the bias the easier it is to get the desired output, but if the bias is negative, then it is difficult for the perceptron to give the desired output because you will need either more of the inputs to a positive number, meaning $x_n = 1$ or the weights of the inputs to be more positive. While the bias is only a small change to the way that the network is perceived, it does change how the network is calculated. When using bias, we can make two changes.

To being, one can replace the condition $\sum_j w_j x_j >$ threshold by representing the sum as a dot product $w * x \equiv \sum_j w_j x_j$, where w and x are vectors composed of the weights and inputs respectively. Additionally, one can relocate the threshold to the opposite side of the inequality and substitute it with the bias. By replacing the bias with the threshold the perceptron is now written as [32]:

$$\text{output} = \begin{cases} a, & \text{if } w * x + b \leq 0 \\ b, & \text{if } w * x + b > 0 \end{cases} \tag{3.7}$$

For example, as shown in Figure 3.4, if we have a perceptron with three inputs, with $w_1 = 2$, $w_2 = 4$, and $w_3 = -2$ and an overall bias of 2. The algebraic equation for this perceptron would be given:

$$\text{output} = \begin{cases} 0, & \text{if } w * x + b \leq 0 \\ 1, & \text{if } w * x + b > 0 \end{cases} \tag{3.8}$$

Then our perceptron would look like this:

If we have an input of (2,2,8) the output produces 0, since (2 * 2) + (4 * 2) + (-2 * 8) + 2 = -4, which is not positive. And similarly, if we have an input of (1,3,3) the output produces 1, since (2 * 1) + (4 * 3) + (-2 * 3) + 2 = 10 which is positive.

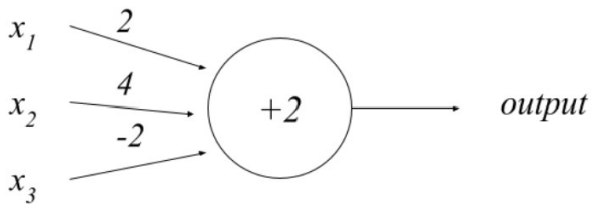


Figure 3.4: Perceptron using bias instead of the threshold.

Now we can connect perceptrons by adding other layers and levels such that they are all connected, which makes it a network. The first layer makes simple decisions by deciding the importance of each input and scales that value to produce the output for that layer. This process is often called weighing. The second layer is deciding by weighing up the results from the first layer. This causes the perceptrons in the second layer to make more complex and more abstract decisions than the perceptron in the first. The more layers one adds to the network the more sophisticated the decision is. These multiple-layered perceptrons are called neural networks which will be discussed in our next section.

3.2.2 Neural Networks

Neural networks are a combination of perceptrons that take in input vectors of n variables $x_1, x_2, x_3, ..., x_n$ to produce a non-linear function $f(x)$ that produces a prediction of the output Y. There are a few distinctions that differentiate neural networks and a single perceptron. First, neural networks have a unique structure, such that they include at least three different layers. The input layer is the farthest layer to the left. The output layer is the farthest layer to the right. And the hidden layers are all the other layers in between the input and output layers. These layers are simple collections of nodes that are connected to every other node in the previous and next layers. The layers can compute activations $A_k = h_k(X)$ which are nonlinear transformations of the linear combination of the inputs, weights, and biases. These

inputs are again weighted $w_1, w_2, w_3, ..., w_n$ based on their importance to the output. Common activation functions include the ReLU and Sigmoid functions which will be discussed later in this paper. The output layer then uses these activations, A_k, as inputs, which result in a function $f(x)$ that makes predictions. Here K represents the number of different levels of activations in the hidden layer. Figure 3.5 is an example of a neural network with one hidden layer with activations $A_1, ..., A_K$ and inputs $x_1, ...x_n$ [14].

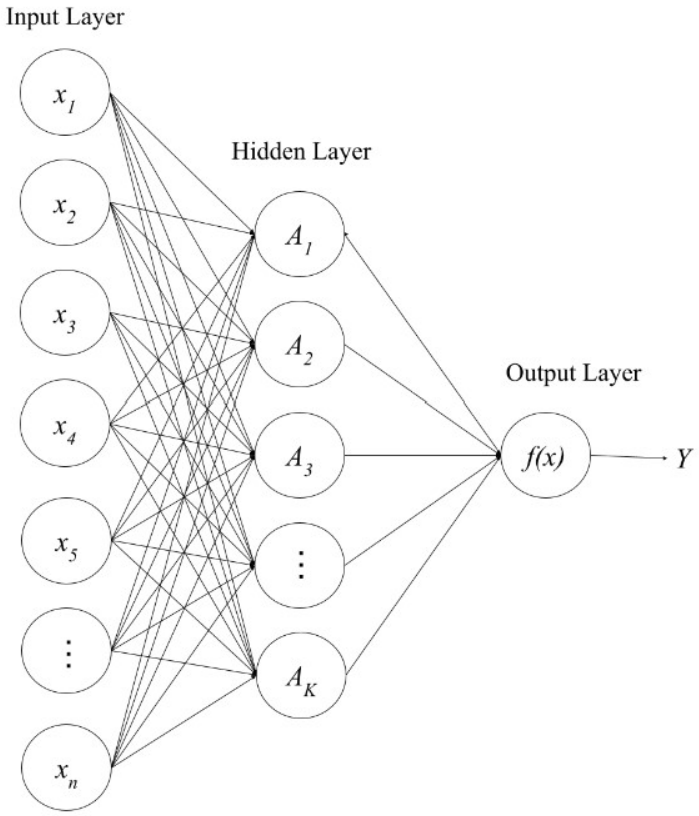


Figure 3.5: Neural network with a single hidden layer.

The arrows in the network are shown in Figure 3.5 indicate that each of the inputs from the input layer feeds into each of the K activations. Then each of the inputs from the activation layers is fed into the output layer to create an output $f(x)$. The arrows serve as weights $w_1, ..., w_n$. The weights are then multiplied by the

inputs $x_1, ..., x_n$ using the dot product to help assign importance. These results are then summed up by the activation function that computes a non-linear activation, $g(z)$. These activations are not specified in advance of making the model, but they are learned during the computation, where each activation function can be written as:

$$A_k = h_k(x) = g(w_{k0} + \sum_{j=1}^{n} w_{kj} x_j) \tag{3.9}$$

When the calculated output from the activation layer is fed into the output layer, these calculated outputs are again multiplied by the weights connecting them to the output layer, such that the output layer $f(x)$ is given:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k \tag{3.10}$$

where $\beta_0, ..\beta_K$ serve as the new weights going to the output layer.

To sum up, neural networks are a loop of the function $f(x)$ where $\beta_0, ..\beta_K$ are the weights and A_k are the inputs, where the inputs come from another layer such that:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{n} w_{kj} x_j) \tag{3.11}$$

If the network was to add an additional layer then the output $f(x)$ would then be given:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{n} w_{kj} g(w_{k0} + \sum_{j=1}^{n} w_{kj} x_j)) \tag{3.12}$$

This process continues for each additional layer added to the network. Therefore, as more layers are added to the network, we will receive a higher complexity output and it will also take longer to compute the calculations.

Most modern neural networks tend to have multiple hidden layers and often many levels per layer. Similarly to single-layer neural networks, multi-layer neural

networks take in inputs $x_1, ..., x_n$ that are weighed and fed into each level of the first hidden layer. Then, these outputs serve as the new inputs for the next hidden layer. These inputs are also weighed before going into each level of the hidden layer. A two-hidden-layer neural network is shown in Figure 3.6.

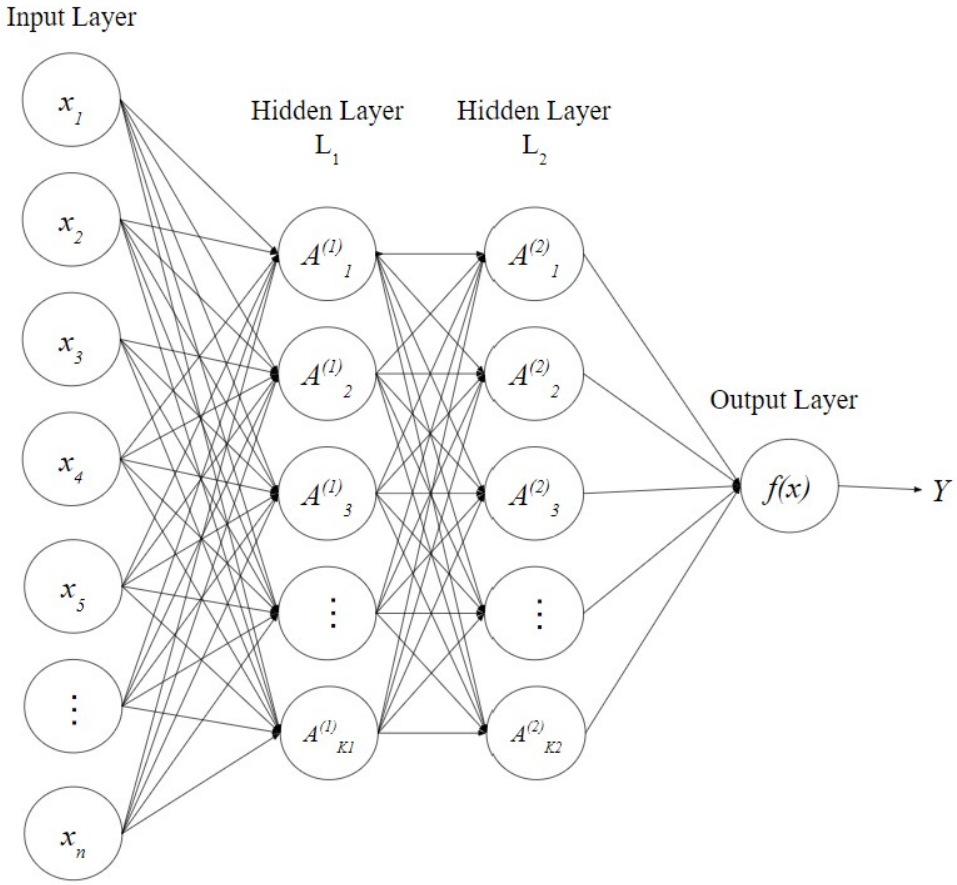


Figure 3.6: A multi-layer neural network with hidden layers L_1 and L_2.

Additionally, neural networks can have multiple outputs. These are represented by a vector $Y = Y_0, Y_1, ..., Y_m$, where m is the number of different outcomes. For instance, if you were predicting handwritten digits, then you would have 10 outputs (0, 1, 2,..., 9). The results of the network would end with a one in the position corresponding to the label and zeros elsewhere. This is also known as one-hot encoding [14]. This type of neural network is shown in Figure 3.7.

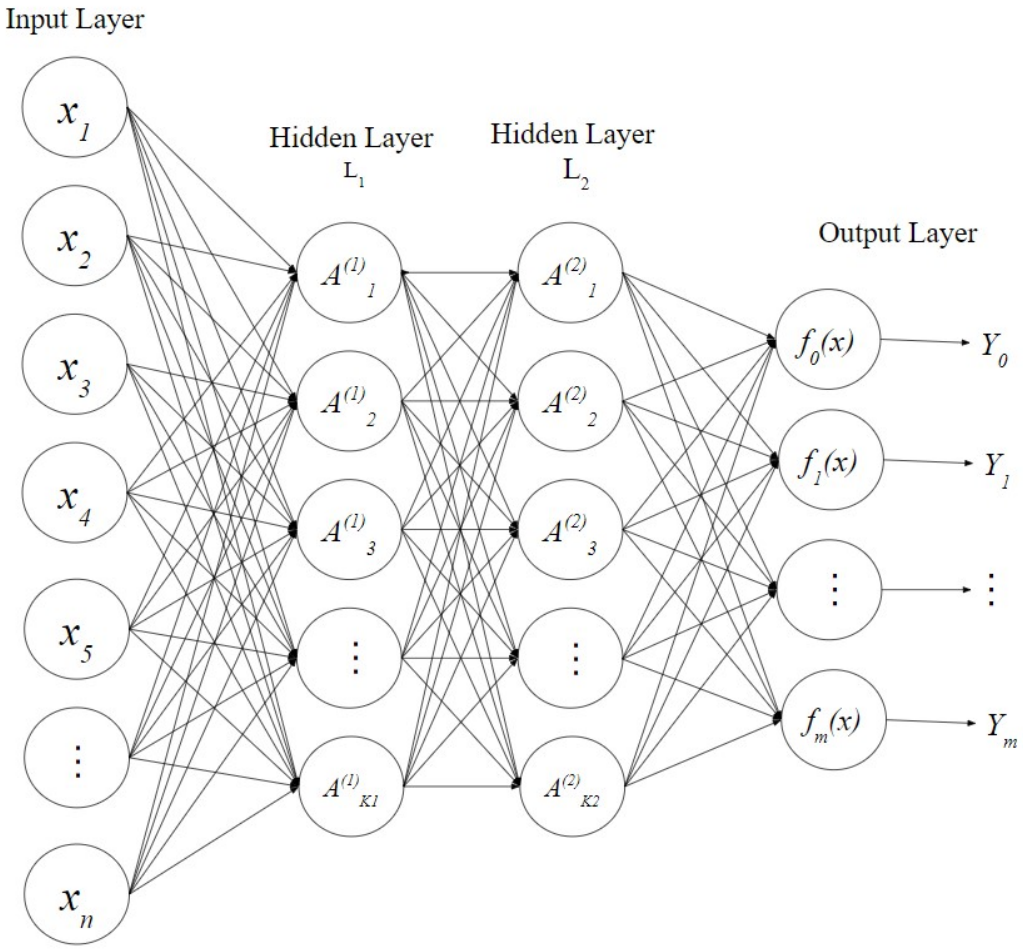


Figure 3.7: A multi-layer neural network with hidden layers L_1 and L_2 and multiple outputs $Y_0, Y_1, ..., Y_m$.

There are many different types of special families of neural networks. In my study, I use convolutional neural networks because they are the most effective in reducing the dimensionality of images without losing information. The GazeCapture dataset has a smaller pixel size (640×480) compared to other eye gaze datasets. Therefore, it is important to keep as much information in the image as possible so that the model can make accurate gaze location predictions. There are other types of neural networks, like recurrent neural networks, but they are best used when investigating topics of time series and language processing, while convolutional neural networks are best when used for classifying images, like eye gaze.

3.2.3 Convolutional Neural Networks

Neural networks have become popular recently due to their ability to classify large image datasets. As dataset sizes were able to increase, so did the number of possible outputs for these images, meaning that we have been able to classify more and more objects. Convolutional neural networks are now the main type of neural network when classifying images because they mimic the human brain by recognizing specific features or patterns in the images that help the algorithm distinguish which output class the object belongs to.

To help get a better understanding we will provide an example of detecting the difference between a cat and a dog, which is shown in Figure 3.8. A person reading this paper can easily recognize that the image on the left is a cat; and that the image on the right is a dog. But we only know this with our prior knowledge of what cats and dogs are. A newborn baby who has never seen a cat or dog before would have a harder time determining the classification of the images. If the baby was shown the picture on the left first and was told that it is indeed a cat and then is shown the image on the right of the dog, the baby might assume that the new image is also a cat because they both share similar features; the hair on their body, four legs, a head, two eyes, pointy ears, etc. Therefore, the baby says the picture of the dog is a cat. Luckily, the baby has smart parents who know the difference between a cat and a dog, and they can correct the baby and tell them that the image on the right is a dog. Now, the next time the baby is shown an image of a dog, it will have a better chance of correctly identifying that it is a dog.

Let us assume that the baby is noticing that the dog has a long snout while the cat does not have a snout. Therefore the baby is weighing this feature over the other features seen in the image. This can lead to a problem. For instance, if the baby was shown an image of a bulldog they might be confused. The bulldog does not have a long snout, so the baby assumes that it is a cat. But again the parents correct the

Figure 3.8: Pictures of my cat and dog show how the human mind can easily identify the difference between the two species.

baby and tell them that it is a dog. So the baby decreases its perception that a long snout is for dogs and no snouts are for cats. But the baby not going to get rid of this perception completely because it is still true in most cases.

Convolutional neural networks work similarly. The network identifies low-level features in the image such as small edges, shapes, and patches of color. These low-level features are then combined to form higher-level features, such as eyes, ears, snout, legs, etc. The network will be able to distinguish which specific features make the input the specific class. And as the model is trained on more images, of the same or a different class, it should be able to make more accurate predictions. Therefore, when the network is being trained it receives a new input image. Then, the presence or absence of these higher-level features will help contribute to the probability that the given image is of the same or different output class. If the predictions are correct, then the network can use that information to adjust its weights for the future, such that if it receives a similar image then it should also be of the same class.

After the network calculates its predictions, the model uses another metric called

the cost function, which is often referred to as the loss function or error function. This metric evaluates the model's accuracy based on the actual and predicted output. The goal of training a convolutional neural network is to minimize the cost function. There are a few different types of cost functions, such as the mean squared error, cross-entropy, and hinge loss. Each type of these functions is suited for a different type of problem and the type of model being used. The function assists the network in adjusting the weights within the model of each layer using backpropagation to improve its accuracy. Backpropagation works by calculating the gradient of the cost function with respect to the weights and biases. The gradient is then used to update the weights and biases in a direction that results in a lower cost function. Backpropagation and gradient descent will be discussed in the next sections.

3.2.4 Gradient Descent and Backpropagation

Now that we know how these networks look and the idea behind how they change we must discuss how we fit and revise the weights and biases so that the model can return more accurate predictions. Most deep learning algorithms incorporate an optimization of some sort, meaning the minimizing or maximizing of a function by changing part of the neural network equation. To improve our networks, we use gradient descent. This process helps increase prediction accuracy and lower the cost function. In a perfect world, we would find an algorithm that finds the weights and biases so that every training input results in the correct output. To achieve this goal we define our cost function [10]:

$$C(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(y(x) - a^{L}(X)^{2}) \tag{3.13}$$

In the equation w denotes the collection of all the weights in the network; b is all the bias; m is the number of inputs; a is the vector of outputs from the network

when x is inputted; L denotes the number of layers in the network, and C is the quadratic cost function which is also known as the mean squared error (MSE). The goal of training the network is to find the weights and biases so the $C(w, b) \approx 0$. This process is known as gradient descent. But before we can use gradient descent, we must use backpropagation. Backpropagation and gradient descent are performed in a sequential manner, where backpropagation is performed first to compute the gradients of the cost function with respect to the parameters, meaning the weights, biases, and nodes, and then gradient descent is performed to update the model parameters based on the computed gradients.

Backpropagation allows the information from the cost function to flow back within the network to compute the gradient and change the weight, biases, and nodes to improve the network in a simple and inexpensive process. Backpropagation is misconceived as only being a process in neural networks, but actually, it can compute the derivatives of any function in mathematics. Backpropagation uses the chain rule of calculus, which computes derivatives of function by using other functions where the derivatives are known. It does this in a specific order and becomes highly efficient.

After we run through the algorithm and compute the cost function, we can use the backpropagation formulas to find the gradients of the parameters. Then when they have been computed, the gradient descent algorithms, or another optimization algorithm, use these gradients to update the parameters. In order to understand these formulas we first must get a better understanding of the notation of the weights, biases, and nodes.

In a neural network, the inputs are connected to each node in a hidden layer. Then each node in that first hidden layer serves as the new input for the next level. Each arrow connecting the input to a hidden layer level is a weight that can be denoted as w^l_{jk}, where l represents which layer the input is going to, j is the j^{th}

node, where 1 is at the top, in the l^{th} layer, where the first is on the left side, and k represents the k^{th} node in the $(l-1)^{th}$ layer. Each of these nodes can also be called a neuron, which gives the name neural network. Two examples of weight notation are shown in Figure 3.9, where the red arrow would be written as w^1_{23} and the blue arrow would be w^2_{31}.

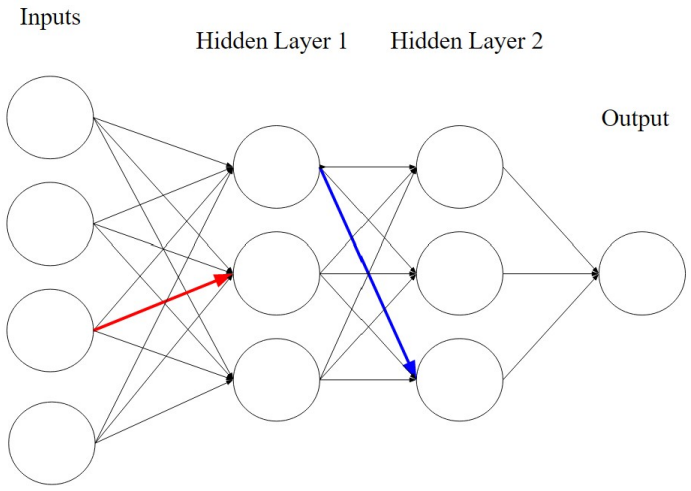


Figure 3.9: Schematic to show how we denote weights as w^l_{jk}.

First, we can compute the gradient on the cost function from the output layer. Next, we can calculate the gradient of the last node in the network, denoted as L by computing element-wise multiplication of the gradient of the cost function and the derivative of the activation function for the last layer. Such that:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{3.14}$$

Then, we can find the gradient of the other nodes, l, using element-wise multiplication of the transpose of the weight in the layer before it, $l+1$, and the gradient of the node in the layer before it, multiplied by the derivative of the current node, such that:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L) \tag{3.15}$$

Also, we can find the gradient of each of the biases fairly easily because they are equal to the gradient of the node of that layer, using:

$$\frac{\partial C}{\partial b^l} = \delta^l \tag{3.16}$$

Lastly, we can compute the gradient of the cost function with respect to any weight by multiplying the transpose of the activation from the node that the weight is from before the weight by the gradient of the node that the weight is going into, such that:

$$\frac{\partial C}{\partial w^l} = \delta^l (a^{l-1})^T \tag{3.17}$$

Now that the gradients of the parameters have been calculated gradient descent can be used to update these parameters. One can visualize gradient descent by pretending that they are at the top of a mountain range and they want to get to the bottom through a series of steps. As long as each step continues to go downhill then you should eventually reach the bottom. Gradient descent finds the derivative $f'(x)$ to find the slope of the function $f(x)$ at point x. Therefore, if the derivative is a negative value then we continue to move in the same direction until we reach a point where $f'(x) = 0$. Points where the derivative equals zero provide no information about which direction to move and are called critical points.

A local minimum is a point in the equation where $f(x)$ is lower than all the points next to itself, such that it is no longer able to decrease the cost function. Another point where $f'(x) = 0$ is called the global minimum. This point results in the lowest value of the cost function. Both of these points can be seen in Figure 3.10. To find the global minimum, we use the following gradient descent formula in order to move in the direction of the steepest descent:

$$\vec{x'} = \vec{x} - \epsilon \nabla_{\vec{x}} f(\vec{x}) \tag{3.18}$$

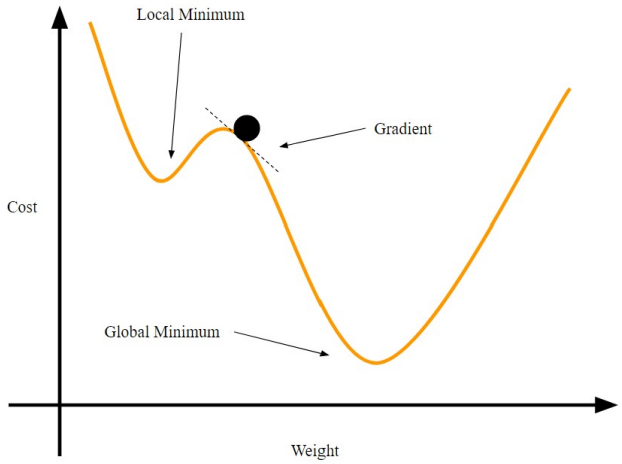


Figure 3.10: Gradient descent visualization.

Where $\nabla_{\vec{x}} f(\vec{x})$ is the gradient of f where all the partial derivatives are contained in a vector. Additionally, ϵ is the learning rate of the algorithm, meaning that it determines the size of the step in order to help determine new weights and biases within the model. It is common to set ϵ to a small constant, but sometimes we can calculate it so that the step size makes the directional derivative vanish [10]. Going back to the analogy about walking down a hill, if you take too large of steps then you might end up at a point higher on the cost function. Furthermore, the spaces that we are hypothetically thinking about are not just in two dimensions as in Figure 3.10. Instead, they occur in larger dimensions with magnitudes in the tens, hundreds, or thousands which makes it extremely hard to understand the appearances of the cost function. So, we tend to use a small value for the learning rate to make the model more efficient when we are near areas closer to minimums.

So far, we have only described gradient descent with a fixed learning rate, but it is also important to know that one can adapt the learning rate of the parameters so that the model can become more accurate as we move on to different iterations. The first iteration inputs all the training data into the model. Then using our cost function we can update the learning rate and the parameters so that when we

input all the training data again for our next iteration our results are more accurate. When creating a deep learning network there are a few different ways to do this. The method used in my study is called Stochastic Gradient Descent (SGD). The SGD algorithm adapts the learning rate to produce a more accurate model for each iteration. Additionally, it is one of the most used optimization algorithms for deep learning [10]. SGD calculates an estimated gradient by using a small set of samples. Using the entire dataset would take an extended period of time to calculate. Therefore, during each step of the model, a mini-batch of samples $\mathbb{B} = \{x^{(1)}, ..., x^{(m')}\}$ is taken from the training set to create an expected gradient, which is given:

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta) \tag{3.19}$$

Where $x^{(i)}$ is an sample from the mini-batch and $y^{(i)}$ is the corresponding output. The mini-batch or batches are small subsets of the dataset that are fed into the network gradually, one after the other. After the model runs through the entire batch, the model computes the estimated gradient that is used in the SGD algorithm to update the parameter using [10]:

$$\theta \leftarrow \theta - \epsilon_k g \tag{3.20}$$

Where θ is an initial parameter and ϵ is the learning rate at the k^{th} iteration of the network. One can choose the learning rate with trial and error, but it can also be calculated to create the optimal output using:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{3.21}$$

where $\alpha = \frac{k}{\tau}$ and τ is maximum number of iterations. Such that after iteration τ, it is common to keep the learning rate constant.

Lastly, one of the most important aspects of using SGD is that the computation

time for each update does not increase as the size of the number of training examples is increased. This allows us to obtain large datasets that can create even more accurate results.

There are a few other ways to tackle this problem, such as Adaptive Gradient Algorithm (AdaGrad), Root Mean Square Propagation (RMSprop), and Adaptive Moment Estimation (Adam). But since they are not used in this study they will not be discussed further.

This process of backpropagation and gradient descent computation continues over and over for the determined number of training sequences, which are called epochs. Each epoch is one full pass through the entire dataset and each epoch consists of mini-batches of iterations where a small sample of the data is put in the model. Based on the size of that dataset, the number of epochs can increase the computation time greatly. However, one problem with increasing the number of epochs is that it can also increase the risk of overfitting the training data. Therefore it is important to choose a number of epochs that are based on the performance of the network and the complexity of the problem being solved. For small datasets, it is common to use up to 50 epochs. And for larger datasets, one can use up to 200 epochs.

Convolutional neural networks assemble this structure by combining two specialized types of hidden layers: convolution and pooling layers. The convolution layers forage for small patterns in the image, while the pooling layers downsample these to select a major subset. To obtain better results, convolution neural network architecture uses many different levels of convolution and pooling layers. The model used in my study uses four different types of layers including convolution layers which are followed by a ReLU activation function, pooling layers, local response normalization layers, and linear layers. These layers, along with two

other commonly used layers often used with convolution will be described in the following sections.

3.2.4.1 Convolution

The convolution layer is made up of convolution filters, each of which determines whether a particular local feature can be found in an input image. These filters rely on the process of convolution, which results in repeatedly multiplying matrix elements and then adding the results. The product of these results in a reduction of the dimensions of the original image. To understand how a convolutional filter works, we will consider a simple example of a 4×4 image:

$$\text{Input image} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \tag{3.22}$$

And we will convolve the image with a 2×2 filter:

$$\text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \tag{3.23}$$

We then get the results:

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + e\gamma + f\delta & b\alpha + c\beta + f\gamma + g\delta & c\alpha + d\beta + g\gamma + h\delta \\ e\alpha + f\beta + i\gamma + j\delta & f\alpha + g\beta + j\gamma + k\delta & g\alpha + h\beta + k\gamma + l\delta \\ i\alpha + j\beta + m\gamma + n\delta & j\alpha + k\beta + n\gamma + o\delta & k\alpha + l\beta + o\gamma + p\delta \end{bmatrix} \tag{3.24}$$

The convolutional filter is applied to each 2×2 patch in the original image, and

when the original input image has a dimension of 4×4 then the convolved image will be a 3×3. Mathematically, we can use the formula:

$$(n_h - k_h + 1) \times (n_w + k_w + 1) \tag{3.25}$$

to find the shape of the new output where the shape of the input array is $n_h \times n_w$ and the convolutional filter has a shape of $k_h \times k_w$ [50].

Convolution filters find local features in an image, such as edges, small shapes, and colors, and make them more noticeable in the convolved image. Therefore the images that are produced from these filters look slightly different from the original, not just in size. In the image shown in Figure 3.11, we have a 192×179 pixel image of a tiger. The convolutional filters in this example are 15×15 images mostly containing zeros, which make the pixels black, and a thin strip of ones, which make the pixels white; both horizontally and vertically. These filters are applied to convolve the image to identify other horizontal and vertical lines in the original image [14]. We can think of the original image as the input layer to the convolutional neural network, and the convolved image as the levels in the first hidden layer. Additionally, we can think of the filters as the weights from the inputs to the hidden layers.

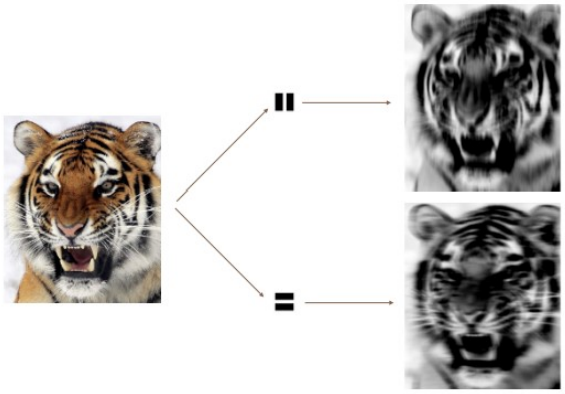


Figure 3.11: Convolutional filters find local features in an image [14]

Furthermore, at the first hidden layer, we use K different convolutional filters to get K different output arrays. We view each of the K output arrays as a separate channel of information, so now we have K channels compared to the three-color channels of the original input array [14]. Therefore, as we move from one layer to the next, the number of channels can increase.

For colored images, which are also known as RGB for the red, green, and blue color components in the image value array, we have three changes represented by a three-dimensional array. Each of these channels is a two-dimensional array; one for red, one for green, and one for blue [14]. Similarly, a single convolutional filter will have 3 channels for each color that is a 3×3 matrix. These matrices change based on the weights needed in the filter. After the first layer, all color information is lost and is not passed onto the following layers.

Alternately, grayscale, binarization, and edge detection images only have one two-dimensional array because there is only one color component to them. Since there is no color component within the first input array, no color information can be lost as it goes into the first layer. These types of images and the shapes of their image arrays will be discussed later in this section.

One difficult part about working with convolutional layers is that they tend to disregard pixels on the perimeter of the image. Therefore, convolution is often combined with other techniques, such as padding, to control the size and shape of the output. Padding refers to the number of pixels added around the border of the input image during the processing of the layer. This results in a new output image that can be the same size as the input image or even bigger. In Equation 3.26 the value of the padding is set to one for both the height and the width. When this happens a border of length one is added all around the input. In the border, all the pixels are set to zero and the convolution filter is applied to the image as normal

[50].

$$\text{Input Image} = \begin{bmatrix} 5 & 1 & 2 \\ 3 & 8 & 5 \\ 2 & 3 & 8 \end{bmatrix} \tag{3.26}$$

and

$$\text{Convolutional Filter} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.27}$$

with padding equal to one results in

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 1 & 2 & 0 \\ 0 & 3 & 8 & 5 & 0 \\ 0 & 2 & 3 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 1 & 2 & 0 \\ 3 & 18 & 7 & 4 \\ 2 & 9 & 24 & 10 \\ 0 & 4 & 6 & 16 \end{bmatrix} \tag{3.28}$$

Mathematically, the new output array will have the shape of:

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1) \tag{3.29}$$

where p_h is the size of the padding for the height and p_w is the size of the padding for the width.

Another way in which we control the size and shape of the output is by changing the stride of the convolution. When we run through the convolution, one first begins in the top left corner of the input image, and then slides the filter to the right by one pixel over all locations across the top; and when we reach the top right corner we shift the filter back to the left side and move down one pixel level. This process continues until all the pixels are convolved with the convolution filter. But when we wish to increase the computational efficacy or we want to downsample,

we change the number of pixels by which shift by, and therefore, some locations are skipped [50]. The number by which we shift is called the stride. Stride is often combined with padding because you must have the correct dimensions to be able to move by the stride amount. For example, we will use the same input image and convolutional filter as before where we have a padding of one. But in this example, we will also have a stride of 3 for the height and 2 for the width.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 1 & 2 & 0 \\ 0 & 3 & 8 & 5 & 0 \\ 0 & 2 & 3 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 2 \\ 0 & 6 \end{bmatrix} \tag{3.30}$$

Now, with the new addition of the stride, where s_h is the stride for the height and h_w is the stride of the width, the new output array will have a shape of:

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor \tag{3.31}$$

Where one round down to the closest integer, such that if we have $\lfloor 3.999 \rfloor = 3$.

Furthermore, choosing the right padding and stride values can have a significant impact on the performance of the network and its ability to effectively learn the underlying patterns in the data. It is important to experiment and find the optimal values for padding and stride, as they can vary depending on the specific task and data being used. By changing the output sizes, the convolution filters can save computation time and help identify important information within the data that might otherwise be missed. Lastly, convolution layers can be followed by activation functions. In my study, the convolution layers are followed by a ReLU activation function, which will be discussed in the next section.

3.2.4.2 Activation Function: ReLU

Each node in a layer of a neural network has an activation function $A(k)$. An activation function is a mathematical function that can be applied to the output of each neuron in a network, and determine whether the neuron should be activated or not. The most common activation function and the type we will use in our neural network is the Rectified Linear Unit (ReLU) activation function. which takes the form:

$$R(z) = max(0, z) \tag{3.32}$$

The function takes in an input and outputs either a zero if the input is negative, or the input itself if the input is non-negative. This means that the activation function only activates the neuron if the input is positive, otherwise, it is turned off. A plot of the ReLU functions can be seen in Figure 3.12.

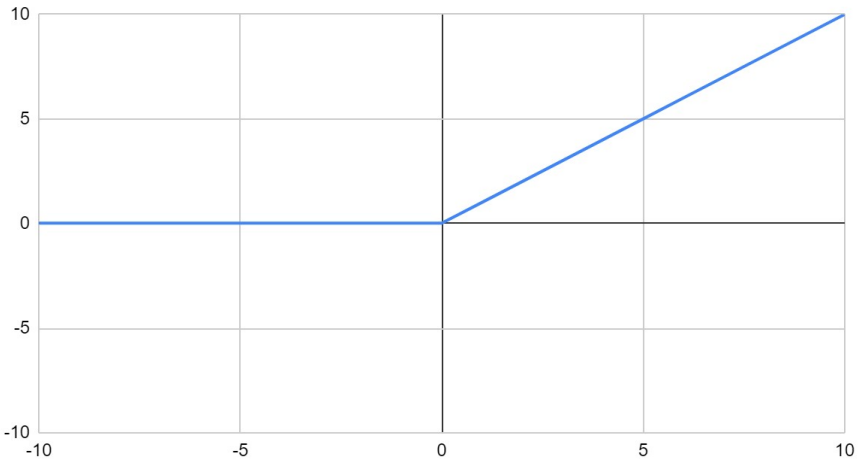


Figure 3.12: ReLU activation function.

The ReLU activation is computed quickly and efficiently compared to other activation functions, such as sigmoid or tanh, because it makes a simple comparison and does not require more expensive mathematical operations. Since the model does not use the sigmoid or tanh functions, these activation functions will not be discussed further.

Another reason that makes the ReLU activation function so common is that the gradient of the function is either zero or 1, depending on whether the input is negative or not. This makes it simpler for the gradient-based algorithms to update the weight of the network and improve the performance. In this study, the output from the ReLU activation function is fed into a max pooling layer, which will be discussed in the next section.

3.2.4.3 Pooling

Pooling is a common technique used in neural networks to reduce the size of the input, and therefore the amount of computation required to process it. The pooling layer takes in the matrix of the convolutional layer as input and again reduces the dimensionality. The layer then makes the more important features of the image more dominant, meaning that it divides the input into a grid of non-overlapping regions and then computes a summary statistic for each region. Strongly defined shapes will result in the output and weaker shapes will be left behind.

The most common type of pooling is max pooling, which takes the highest value of each 2×2 patch in the input images. But this is only applied after the activation has been applied to the feature map. Below is a simple example of max pooling being applied to the input matrix:

$$\text{Max Pooling} \begin{bmatrix} 24 & 17 & 67 & 13 \\ 22 & 18 & 21 & 52 \\ 2 & 23 & 18 & 32 \\ 8 & 29 & 25 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 24 & 67 \\ 29 & 32 \end{bmatrix} \tag{3.33}$$

There are other types of pooling, like average pooling, but the model used in this study only uses max pooling. This technique helps the model reduce its size and reduces the computation time for the model. Additionally, pooling helps reduce

overfitting because it provides another form of regularization. Pooling is commonly applied after one or more convolution layers in the neural network. In the model used in this study, the max-pooling layers are followed by a local response layer, which will be discussed next.

3.2.4.4 Normalization

The next type of layer that is used in the model for this study is the Local Response Normalization (LRN) layer which applies normalization across the channels. Normalization is a common technique used in convolution neural networks and other types of machine learning methods. This technique is used to improve the run time of the network by scaling the input data. The LRN layers are often applied after a ReLU activation function. One advantage of using a ReLU activation function is that it can avoid saturations without requiring normalization. Saturations arise when the weights and biases are initialized poorly. Additionally, they can happen during training when the gradients become too large or too small, which can result in the network becoming stuck at a certain minimum or even failing to find a minimum at all. As long as some of the training samples in the batch produce a positive input to the ReLU, the neuron will learn from it. Nevertheless, it can still be beneficial to the network. The responses normalized activity $b^i_{x,y}$ is given [25]:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^i_{x,y})^2 \right)^{\beta} \tag{3.34}$$

where $a^i_{x,y}$ is the activity of a neuron computed by applying node i at position (x, y), N is the number of in the layer, and $k, n, \alpha,$ andβ are hyper-parameters whose values are determined using a validation set.

3.2.4.5 Linear

The last type of layer that is used in this study is a linear layer. Linear layers, which are also known as fully-connected layers, are the simplest type of layers in neural networks because they only connect every input neuron to every output neuron. In simple terms, the layer adds an additional weight to each input such that it applies a linear transformation to the incoming data. An example of one of these layers is seen back in Figure 3.5.

The next two sections discuss two other types of layers that are also commonly used in convolutional neural networks; dropout and softmax. The dropout layer is a technique used to prevent overfitting and the softmax layer is used to make the model more accurate by adding and helping interpret the probability of certain outcomes.

3.2.4.6 Overfitting

In deep learning, dropout is used as a method to prevent overfitting within the model. Overfitting occurs when statistical models fit a training dataset too precisely, meaning that it is very accurate for the training dataset. When the overfitted algorithm is used to predict a new dataset or a testing dataset, we would get inaccurate predictions because it is not accustomed to some aspects of the unseen data. Avoiding overfitting allows us to make highly accurate predictions on unseen data. This is the overall goal of the network, so we must prevent overfitting. Overfitted models might become too reliant on an aspect of the data. In neural networks, we find specific features, shapes, and characteristics to help make our predictions, but if some of these features are not within the testing data it would lead to inaccurate results. For instance, imagine that within our eye gaze dataset, all the participants had symmetrical faces, such that the distance between their eyes was proportional to the size of their faces. When this algorithm is trained it learns that it

can use this scale to help make its prediction for where the participant is looking because it is falsely correlating the proportional distance between their eyes with where they are looking. Furthermore, when the network is performed on a new dataset it will assume that the distance between the eyes of the new participants works similarly, but instead, it does not which results in inaccurate predictions. While it is still possible for the model to use this distance in some way in a diverse dataset, since a local minimum could be found by a gradient that relies heavily on neurons that represent the distance between two eyes, we need to ensure that dropout is used to make sure that we are not overfitting the model.

Dropout works by randomly dropping out, or setting to zero, a certain number of outputs from a layer during training. This effect makes the network less complex and more resilient to overfitting because the model must learn multiple independent representations of the same data.

To implement dropout, we randomly choose a fraction from a layer to set to zero during each training step. For instance, if the dropout is set to 0.5, this means that half of the outputs from the layer will be set to zero during each step. Then, we must scale the remaining outputs by a factor of 2 so that the expected output of the layer remains the same. This process is not implemented during the testing of the network and instead, we use the full network. This allows for the network to make use of all the outputs and make more accurate predictions, which is the overall goal of the model.

3.2.4.7 Softmax

The softmax function is commonly used in neural networks, especially in the context of classification problems. It maps the outputs of a network to a probability distribution over classes so that the output for each class can be interpreted as the probability that the input belongs to a certain class. The function takes in a vector or

inputs and produces a new vector of outputs in the range of zero to one, where the sum of all the outputs is equal to 1. This means that each output can be interpreted as a probability, with the input that produces the highest output being the most likely class.

The softmax function can be written as:

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{i=1}^{K} e^{y_i}} \tag{3.35}$$

where y is the input vector, σ is the softmax function and K is the number of classes. Additionally, the softmax function is well suited for gradient-based algorithms because it is always monotonically increasing and has a smooth, differentiable curve. Overall, the softmax function helps the model to be more accurate in making its predictions.

While the softmax function is not used in the eye gaze mode, it is used in the FairFace demographics classification model to classify the races and genders of the participants.

3.3 Image Processing

Color is a useful technique that helps simplify and identify certain objects and shapes in an image. In my study I will be using four different types of image processing techniques, RGB, grayscale, binary, and edge detection, to attempt to see what type of image processing leads to higher accuracy of eye gaze and high algorithm fairness. The human eye can discern thousands of different shades and intensities [27] and computers process images similarly by assigning each color a numerical value for each pixel. Therefore, each image is a matrix of numbers, where each value is corresponding to a pixel in the image. In this section, I will describe what these four different types of image progressing techniques look like, the different

methods of each technique that I will use in my study, how the computer reads them, and the mathematics behind the changes.

3.3.1 RGB

The first type of image processing that I will use is RGB, which is the true color of an image. They are called RGB because it includes arrays for the red, green, and blue primary colors of light. These three colors create an intuitive way to describe a wide range of colors. When adding these three values together we can obtain a full spectrum of color that includes 16,777,216 different colors [42]. We use these three colors because the human visual system has three types of color receptors that respond to different wavelengths of light and using these three colors create the closest match to the way that we perceive colors with our eyes. The images in Figures 3.13, 3.14, and 3.15 illustrate this style by showing RGB on a GazeCapture participant, a colorblind test, and nature. RGB images consist of a $m \times n \times 3$ array of values of each pixel. For a 24×24 pixel image, the array would be $24 \times 24 \times 3$ for a total of 1728 different values being inputted. Additionally, the 3 represents the three different levels in the array for the red, green, and blue color components of each pixel. Meaning that in the pixel (R, G, B) R represents the red value, G represents the green value, and B represents the blue value. Each of these values is between 0 and 225, where 0 represents no intensity or no color, and 225 represents maximum intensity or full color. When a pixel value has 225 in one position and zeros in the rest, then the pixel is the color corresponding to which column has the 225 value. For example, if the pixel value is (225, 0, 0) then the pixel is red. Also, a (0, 0, 0) value for a pixel means that the pixel in the image is black; and a (225, 225, 225) value of a pixel means that the pixel is white. This wide range makes them useful in image processing. Lastly, each RGB image is 3 times larger than other types of images because it has those three different levels for each of the red, green, and

blue components, where each value occupies 8 bits. Therefore the RGB images are stored as 24-bit images. This leads to these files being much larger than other types of images, like grayscale, which will be mentioned next.

Figure 3.13: RGB on GazeCapture participant.

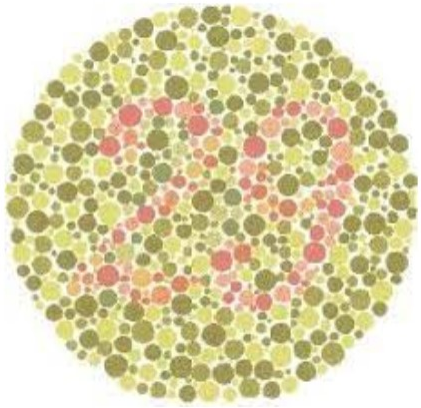


Figure 3.14: RGB on a colorblind test image.

3.3.2 Grayscale

Grayscale images consist of shades of gray ranging from black to white. As opposed to an RGB image, grayscale is a single-channel image, meaning that it only has one component instead of three. Grayscale images are represented as 2D arrays, with each element being a $m * n$ array. They are represented by the intensity of

Figure 3.15: RGB on an image of nature.

the corresponding pixel in the original image that ranges from 0 to 255, where 0 represents black, or no intensity, and 255 represents white, or maximum intensity. These images are commonly used in image processing because they are simpler to work with than color images because they only use 8 bits. The five types of grayscale images are shown in Figures 3.16, 3.17, and 3.18 by showing the different methods of grayscale on a GazeCapture participant, a colorblind test, and nature. There are many different types of grayscale conversions. Kanan and Cottrell use eight different conversion methods [15], but in our study, we are only using four of them: intensity, luminance, value, and luster. These different types of grayscale algorithms use the dot product to multiply each of the RGB pixel's color values by a factor and then add up all the values, such that the sum of the values cannot exceed 255. The simplest color-to-grayscale algorithm is the intensity function, such that:

$$G_{intensity} = 1/3(R + G + B) \tag{3.36}$$

Another grayscale method that is frequently used in computer vision and in this study is the luminance function which is designed to match human brightness

perception by using a weighted combination of the RGB channels, such that:

$$G_{luminance} = 0.3R + 0.59G + 0.11B \tag{3.37}$$

The numbers that are multiplied by the color components add up to 1, and is used based on the way that the human eye is more sensitive to certain colors. Therefore, since the eye is more sensitive to green it has a more significant weight applied to it. Next, the value algorithm provides the brightness information by taking the maximum of the RGB channels, such that:

$$G_{value} = max(R, G, B) \tag{3.38}$$

After that, the luster algorithm uses a similar technique as the value algorithm but also finds the minimum value of the channels. After that, the algorithm finds the average of the two values to convert the image to grayscale, such that:

$$G_{luster} = 1/2(max(R, G, B) + min(R, G, B)) \tag{3.39}$$

There are many other types of grayscale methods that can be used, like Saravanan's method [42]. This method can be found in the Appendix. But due to time constraints, it was not included in this study.

3.3.3 Binarization

Binarization is the conversion of a grayscale image into a binary image, which is a black-and-white image that consists of only black and white pixels. The process involves determining a threshold value that decides if the pixel is black or white. Thresholding is an effective tool for separating objects in the image from the background. Binarization techniques in image processing are grouped into

Figure 3.16: Grayscale methods on GazeCapture participant.

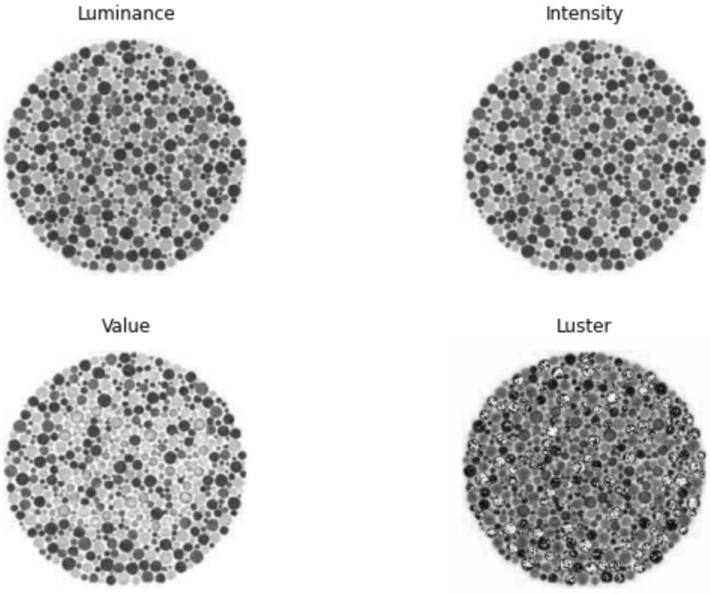


Figure 3.17: Grayscale methods on a colorblind test image.

two categories; global binarization and local binarization. I investigate one type of global binarization. The binarized images are shown in Figures 3.19, 3.20, and 3.21, where Figure 3.19 shows binary on a GazeCapture participant, Figure 3.20 shows binary on a colorblind test, and Figure 3.21 shows binary on nature.

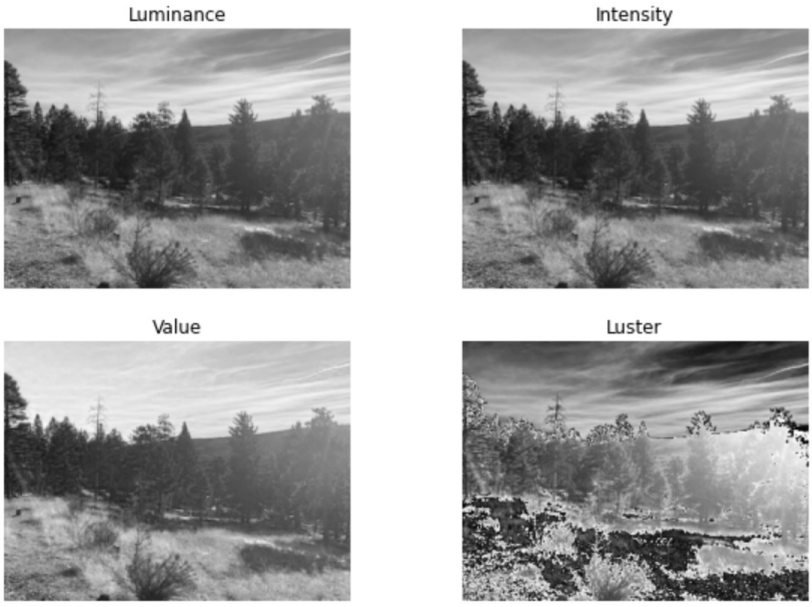


Figure 3.18: Grayscale methods on a nature image.

Global binarization tries to find a single threshold that applies to the entire image. This method is efficient and it can be processed quickly. One limitation of global binarization is that the threshold might seem accurate for a single image, but it might be inaccurate for the majority of the dataset. Therefore, this method is mostly used for scanning documents or images where the backgrounds and objects being separated are very similar. One example of global binarization is Otsu's method which is an optimization problem that searches for a threshold that minimizes the variance between the background and the object [34].

Additionally, global binarization tends to be more inaccurate when there is more noise, or more changes in color, in the picture. Therefore, local binarization was created to estimate a different threshold for each pixel according to the values of the neighboring pixels. One issue is that this method is slower since it requires the computation of each pixel and its neighborhood of pixels. Additionally, as the size of the image increases, computation also increases for most methods. Binary methods that include local binarization includes the locally adaptive and Singh

methods. The methods were not implemented due to time constraints and can be found in the Appendix.

Figure 3.19: Binarization methods on GazeCapture participant.

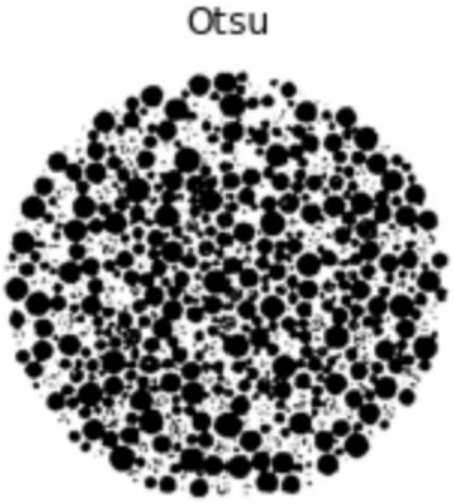


Figure 3.20: Binarization methods on a colorblind test image.

3.3.4 Edge Detection

Edge Detection in image processing localizes significant variations of colors and shadows which then helps with the identification of shapes, structures, and edges

Figure 3.21: Binarization methods on a nature image.

in an image. These edges provide important information regarding the physical, photometrical, and geometrical properties of the objects located within the image. Specifically, in eye gaze, this means the locating of the body and facial components' edges, which are crucial in helping determine the direction that a person is looking at.

The process of locating the edges in an image is simpler when the input image is a grayscale image, meaning that it is a 2D array consisting of pixels between the value of 0 and 255. One problem with edge detection is that the method can become very sensitive to noise. Therefore, the image must be smoothed to limit the number of edges that can be identified, otherwise, all changes in color will be shown. Sadly, this does mean that some of the information in the image is lost and some structures could be in slightly different locations on the image plane. To smooth the image, we use Gaussian smoothing, also known as Gaussian scale-space [29]. The Gaussian filter is denoted by G_σ with a standard deviation σ. If we consider a normalized grayscale image I with R rows and C columns as a function

$$I : \{1, ..., R\} \times \{1, ..., C\} \rightarrow [0, 1] \tag{3.40}$$

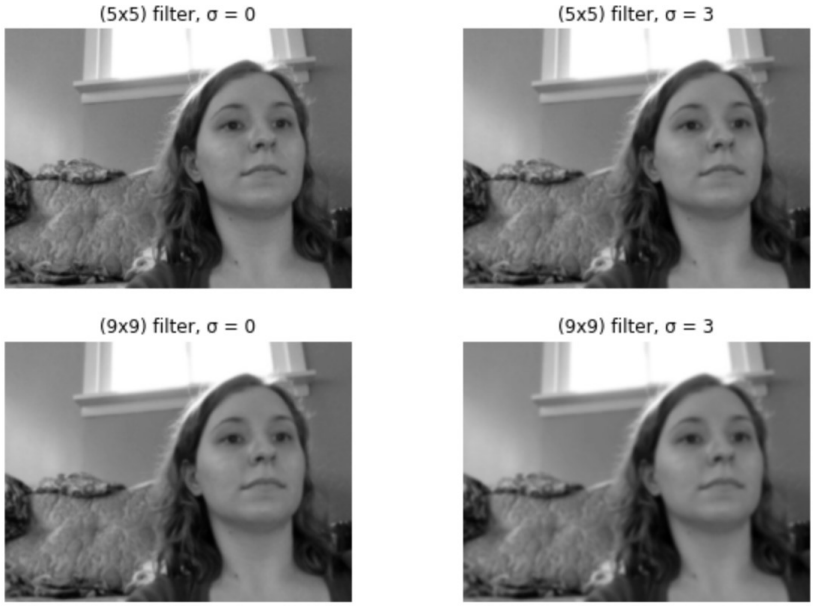


Figure 3.22: Gaussian Smoothing methods on GazeCapture participant.

then its projected Gaussian smoothing can be represented as

$$I_\sigma : \{1, ..., R\} \times \{1, ..., C\} \rightarrow [0, 1] \tag{3.41}$$

This means that $I_\sigma(x, y)$ is equal to the convolution of the image I with the filter G_σ, or $(G_\sigma) * I(x, y)$ [29]. With these equations as we increase the standard deviation σ the more smoothed the image becomes and the less noise there is in the image. This results in fewer edges in the edge detection images. Similarly, if you increase the size of the filter, then the image will also become more smoothed, meaning fewer edges. Based on Figures 3.22, 3.23, and 3.24, which show the Gaussian smoothing with a GazeCapture participant, colorblind test, and nature, our edge detection images will use a (5,5) filter with a $\sigma = 3$ because it eliminates noise, but it also loses less information from the image compared to the other values.

In this paper, I will investigate three different types of edge detection methods. Those methods include the Sobel operator, the Canny edge detector, and the Marr-Hildreth edge detector. The images shown in Figures 3.25, 3.26, and 3.27 illustrate

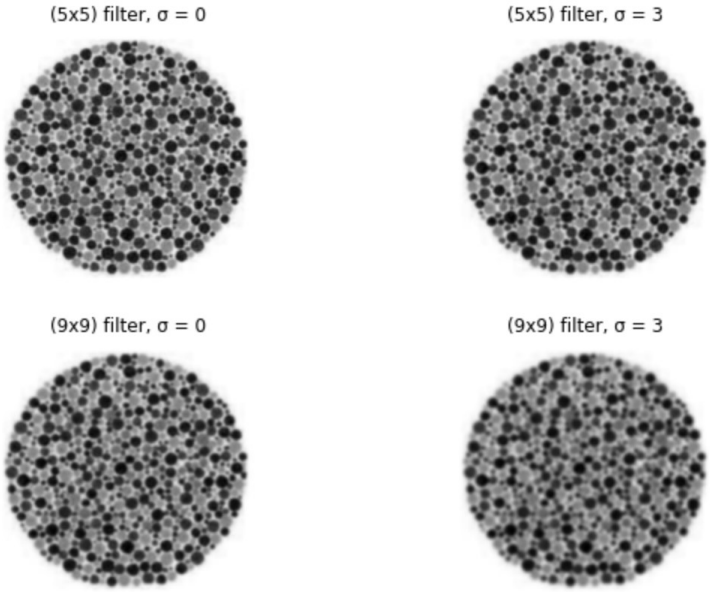


Figure 3.23: Gaussian Smoothing methods on a colorblind test image.

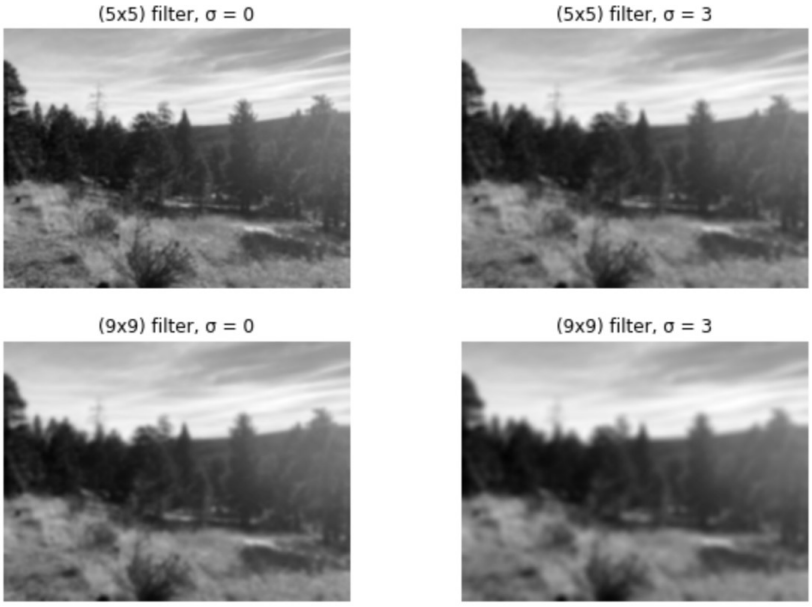


Figure 3.24: Gaussian Smoothing methods on a nature image.

Figure 3.25: Edge Detection methods on GazeCapture participant.

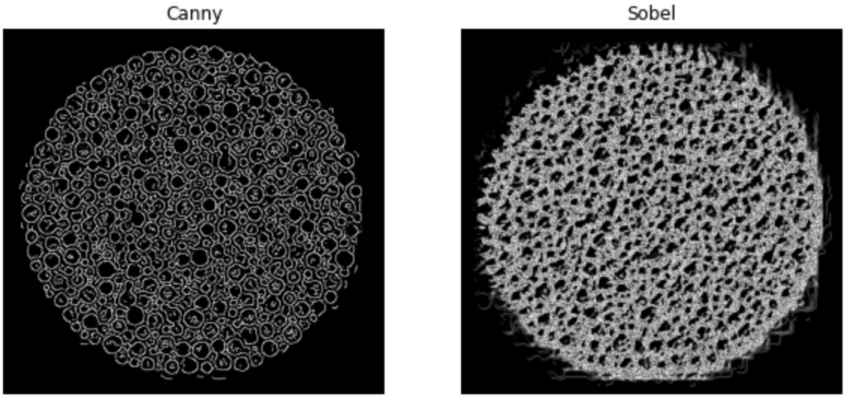


Figure 3.26: Edge Detection methods on a colorblind test image.

these techniques, by showing the edge detection methods on a GazeCapture participant, colorblind test, and nature.

First, the Sobel edge detector uses the concept of convolution, which as mentioned before is the dot product of two matrices where one matrix slides over each patch of an image. The first matrix is a convolution kernel that is applied to the second matrix, which is the image itself. The Sobel operator applies two 3×3 kernels [4]. The first is for detecting edges in the horizontal or x direction and is given as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.42}$$

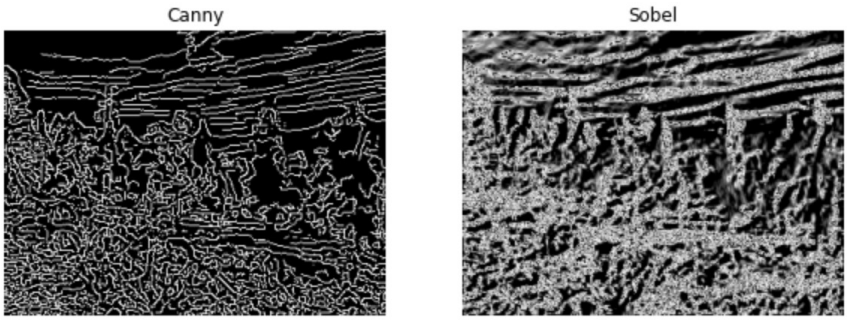


Figure 3.27: Edge Detection methods on a nature image.

And the other is for detecting edges in the vertical or y direction and is given:

$$G_y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.43}$$

The Sobel operator then convolves these kernels with the image, which involves sliding the image over each 3×3 patch in the image and then taking the dot product of the overlapping pixels. This results in two images, which are then combined to create a final image that discovers the gradient magnitude at each pixel, which helps identify if there is an edge in the image. This is found in the equation:

$$G = \sqrt{(G_x)^2 + (G_y)^2} \tag{3.44}$$

and the angle of the gradient, shown in Figure 3.28, is given:

$$Angle = \theta = arctan\left(\frac{G_y}{G_x}\right) \tag{3.45}$$

and the direction of the edge can be found by computing $\theta - 90°$.

The Sobel operator is often slower to compute, but its larger convolutional kernels help smooth the image so that it is less sensitive to noise.

Next, the Canny edge detection also is executed by convolving the image with a

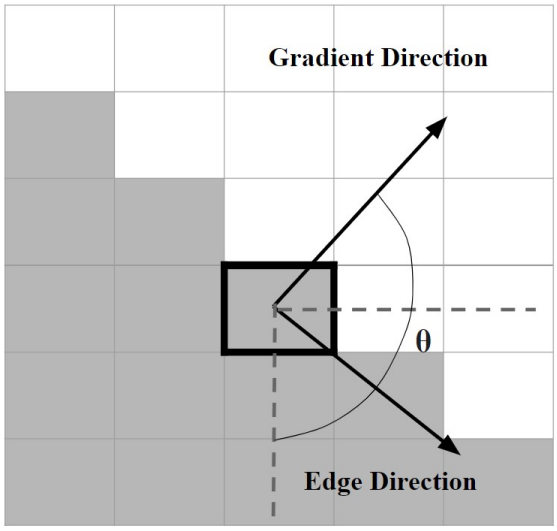


Figure 3.28: Illustration for the direction of the gradient, the direction of the edge, and the angle θ.

filter that can identify edges. Using a Gaussian-smoothed image we can find the first derivative of the 2D Gaussian G in a direction n where

$$G = e^{(-\frac{x^2+y^2}{2\sigma^2})} \tag{3.46}$$

and

$$G_n = \frac{\partial G}{\partial n} = n \cdot \nabla G \tag{3.47}$$

Even though n is unknown, it should is meant to be in the direction of the edge that is being detected. This is also considered finding the gradient orientation of the pixel [31]. The direction in which n travels can be estimated from the smoothed gradient direction such that:

$$n = \frac{\nabla(G * I)}{|\nabla(G * I)|} \tag{3.48}$$

where $G * I$ is the convolution of the filter and the image. Now that we have the direction of the edge we can discover if an edge point is at a local maximum. At a local maximum

$$\frac{\partial}{\partial n} G_n * I = 0 \tag{3.49}$$

and we can substitute for G_n from above and find the associating Gaussian convolution [1]. This also is known as computing the Laplacian along the gradient [31]. The Laplacian method searches for all the points of zero-crossings in the second derivative of the image to find edges, such that:

$$\frac{\partial^2}{\partial n^2} G * I = 0 \tag{3.50}$$

And at such a point in an image, these magnitudes suggest the strength of the edge located at the point. This magnitude is denoted as

$$|G_n * I| = |\nabla(G * I)| \tag{3.51}$$

And the amount of edges found in the image is based on the standard deviation σ that is used in the Gaussian smoothing process. A large σ value will result in fewer edges being detected. There are other edge detection methods, like Marr-Hildreth that also use the Laplacian, but due to time constraints, it was not implemented in my study.

All of these different types of image processing techniques are applied to the images in the Gaze Capture dataset. In the next section, we will go in-depth into this dataset and describe the transformations that were made to add the demographics of the participants to the dataset.

CHAPTER 4

Data Description

In this chapter, I analyze the GazeCapture dataset and discuss the data transformations that were made to create a final version of the dataset. I will also conduct an exploratory data analysis to gain a better understanding of the participants that are within the final version of the dataset.

4.1 GazeCapture Dataset

The dataset of interest, GazeCapture, includes 2,445,504 different images of participants looking at a random point on their mobile device, which was either an iPad or Phone. Additionally, the dataset consists of information about the position where the participant was looking, the device used, and the location of the face. The dataset consists of 1,474 subjects. The dataset has the second-largest number of participants out of other eye gaze datasets (GazeFollow with 130,339), but it has the second-largest number of images (NVGaze: Real World with 2,500,000). To develop a dataset on such a large scale, Krafka et al. used a platform called Amazon Mechanical Turk (AMT) to recruit subjects through crowdsourcing, meaning that they collected participants using the internet. Crowdsourcing is a simple alternative that can help collect data quickly and give a wide range of variations of genders and races, along with a variety of backgrounds, such as inside or outside and bright

or dim lighting. Many other datasets in the eye gaze field that researchers have collected are limited to a lab environment, which "leads to a lack of variation in the data and is costly to scale up" [23]. Additionally, other eye gaze datasets are focused on larger screens, instead of mobile devices. Therefore, another goal of the GazeCapute dataset was to create an approach to eye tracking that was used on mobile devices. They found that collecting eye-gaze data on mobile devices has three advantages: reliability, scalability, and variability.

Out of the 1474 subjects in the dataset, 1103 of the subjects' data came through AMT, another 230 came through in-class recruitment of the University of Georgia, and the final 141 subjects came through App Store downloads. 1249 subjects used iPhones resulting in approximately 2.1 million images, while 225 subjects used iPads, which resulted in 360 thousand images. Using these two types of mobile devices helped provide a wide range of variability in the different head poses, referring to where the head is in the image, the angle of the head, and gaze location, referring to where on the screen the participant is looking. Figure 4.1 shows the variability of the gaze locations relative to the participant's head pose. The darker red regions indicate parts of the screen that had more frequent gaze locations, while the lighter yellow regions indicate less frequent regions. The outline of these regions helps show the different shapes of mobile devices. iPhones were more frequently used in the dataset and therefore locations closer to the camera should be more easier to predict because the model has more data with those locations to learn from. Additionally, people tend to view an iPhone from only three different head poses, whereas they tend to view an iPad from four different head poses. This is because iPhones are only be viewed from three different orientations: one where the camera is on the top or portrait mode, one where it is on the left, and one where it is on the right, which is also known as landscape. In contrast, iPads are used in all three orientations as the iPhone and in portrait mode when the camera is on the bottom.

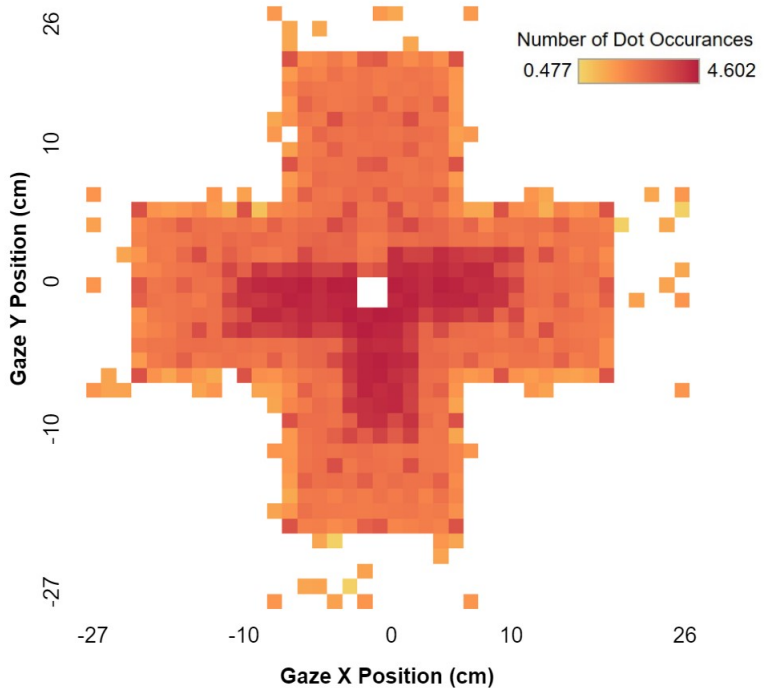


Figure 4.1: Gaze Locations in GazeCapture dataset relative to the Head Pose

One difficulty with the GazeCapture dataset is that there is a possibility of inaccurate data collection. Meaning that the participants might not be looking at the dot on the screen or their fingers could be blocking the face in the image. This is due to the fact that there was no human supervision to make sure the participant was completing the test correctly and honestly since it is a wild dataset. And with using mobile devices, the probability of the participant being distracted increased since they could receive a notification or a phone call. Therefore, to make the data more reliable many steps were taken to keep the participant focused. First participants were required to put their devices in airplane mode to make sure that they could not get a notification or a phone call. Next, instead of having a plain dot on the screen for the participant to focus on, they showed "a pulsating red circle around the dot ... that directs the fixation of the eye to lie in the middle of that circle" [23]. The dot then pulsated for 2 seconds before the image of the participant was taken. Then, at the end of the 2 seconds, a small letter L or R appeared for 0.05 seconds,

which would then require the participant to tap the side of the screen depending on which letter was shown, L for left and R for right. If the wrong side of the screen was tapped then the participant would have to repeat the gaze location again. Additionally, they used a face detection algorithm to make sure that enough of the participant's face was visible in the frame. Without the face being shown in the image it is impossible to predict where they are looking. Lastly, completing the process took each participant about 10 minutes because they were shown 60 dots for each orientation of the device. For iPhones, there are three orientations, and for iPads, there are 4. The dot locations were based on both random and 13 fixed locations helping add the variability of the dataset.

4.2 FairFace Demographic Transformations

To be able to conduct research on the gender and racial fairness of the models, both gender and race variables had to be known. Unfortunately, the GazeCapture dataset did not include these and no other eye gaze dataset includes this data. Therefore to determine the gender and race of the participants I had to classify them with another classification algorithm available. In this section, I will describe how their classifications were conducted and the training data used to make the predictions.

To obtain the demographic information I used the FairFace [17] demographic classification algorithm that predicts race and gender. The algorithm uses supervised learning, meaning it trains the model on images of people with their race and gender labels known to the computer. The model was trained on 108,501 images and was made to be balanced on race and gender. Many datasets lack the balance of demographics, including GazeCapture. As the model is trained on people that look more similar, patterns can become more easily recognized by the model. While this might be good for people of the more frequent demographic, it can also be harmful

to those that are represented less frequently. Therefore, I choose to use the FairFace algorithm because of its balance of demographics and the highly accurate results of correctly classifying other datasets.

The FairFace paper was published in 2021, meaning that its results are fairly recent. In their study, they compared many other demographic classification datasets. Their study was one of the few to include races other than Black, White, and Latino. In addition, the model included East Asian, Southeastern Asian, Middle Eastern, and Indian. Their results show that when training models on their dataset, the testing dataset, which was other demographic classifying datasets, correctly predicted the race of 94-97% on the White participants and 84-96% on the non-White participants, which is much higher than when training on other demographic datasets which predict non-Whites around 38-88%. Then, when looking at the gender predictions, again FairFace is the most accurate method. When predicting the genders of White participants it correctly predicts 92-98%, and for non-White participants, the model is even more accurate when predicting genders at 93-98%.

The FairFace model incorporates the ResNet-34 convolutional neural network model architecture. The ResNet-34 model has 34 layers and uses residuals to determine the output. A residual is a difference between the predicted and observed output. The model works similarly to any other convolutional neural network where most layers have 3×3 filters and maintain two simple design rules of "(i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer" [11]. Additionally, the convolution layers have a stride of 2 and the last layer in the network is a pooling layer that uses the softmax function. But, what makes the ResNet neural networks special is that they include a shortcut to use the residuals of the network. The model adds the input of the layer to the output before sending the result of the layer to the next layer. The residual

connections, therefore, transform the mapping into a residual function because it is connecting the input to the output of each layer. Mathematically, pretend we have an input image x, and in a normal convolution layer, the output y is found by performing the transformation with the convolution layer $F(x)$. When using the ResNet model the transformation from the convolution layer would now be $y = F(x) + x$ where x is added to the output of the layer. The structure of a residual building block for the ResNet-34 model is shown in Figure 4.2. Each time the residuals are connected to the output they skip over two layers. Therefore, this process happens 16 times in the model; not 17 times because the first layer and last layers do not use the residuals.

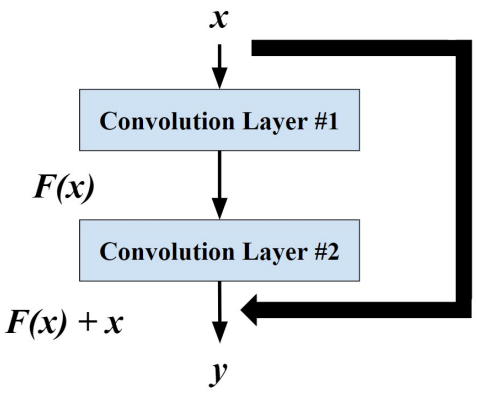


Figure 4.2: Residual learning block for the ResNet-34 model

To create demographic information for each participant, I selected one clear image that showed the best full view of the participant's face from the list of their images. This could create some bias since what I determined to be a clear view might be different from what the participant feels is a clear view. Some of the participants have around 2000 images and therefore, if I found an image that showed the entire face early, I would move on to the next participant. Many of the images include fingers in front of parts of the images such as Figure 4.3, images with only parts of the face within the frame like Figure 4.4, or are very blurred like Figure 4.5.

The blurriness was most likely due to the participant moving their head when the image was being taken. Those types of images were not used in the demographic classification because (i) information was missing, or (ii) the algorithm would not detect a face in the image to be classified. The model then uses these images to create gender and race predictions.

Figure 4.3: GazeCapture image where finger blocks the view of the face.

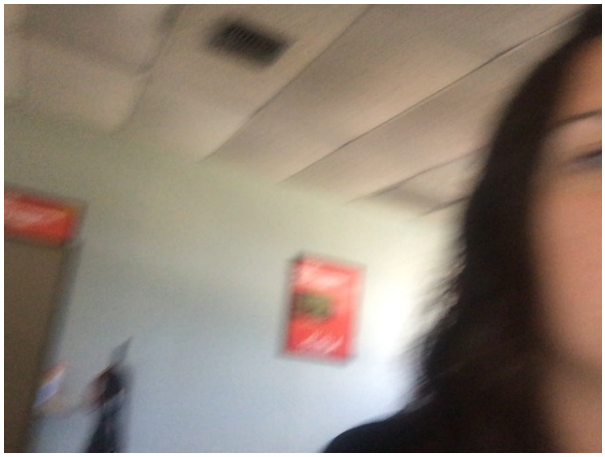


Figure 4.4: GazeCapture image where the face is only partially shown.

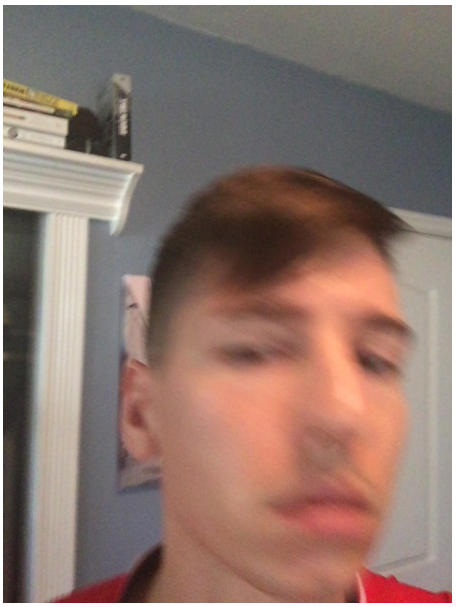


Figure 4.5: GazeCapture image where the image is blurred.

4.2.1 Exploratory Data Analysis

The FairFace model created a spreadsheet of the race and gender demographics that could be easily accessed in Visual Studio Code. The title assigned to the participants' images and data during the original download was a five-digit identification code. This code was utilized as an identifier throughout my analysis and it was added to both the metadata MAT file and the gender and race classifications spreadsheet. This allowed me to use these values to create race and gender metrics to compare the algorithm fairness of the GazeCapture model. It is important to recognize the possibility of falsely classified predictions of demographic information. Since no eye gaze dataset includes this information it had to be generated to conduct my study. In this section, we will discuss the results of these predictions.

The FairFace gender and race classifications were predicted based on one image for each participant. This process allowed for the determination of the race and gender variables. The FairFace algorithm found 803 male and 671 female participants. These corresponding results are shown in Table 4.1. Each participant

in the study was allowed to take as many images as they wanted. The most number of images by a single participant was 3,591 and the smallest number of images was 5. 1,358,819 of those images are from male participants, while 1,088,168 images are from female participants. Therefore, the gender variable is unequal in favor of the male group, which could lead to some bias in the data since it has more data to train itself on. Additionally, the FairFace algorithm found 822 White, 134 Black, 233 East Asian, 45 Southeast Asian, 29 Indian, 136 Latino, and 75 Middle Eastern participants. Again, there was an unequal number of images for each race. The White participants produced the largest amount of images at 1,401,073, which is at least 3.5 times more than any of the races. These corresponding results are shown in Table 4.2. The next highest number of images is for the East Asian group at 395,598 images. Then, 222,462 for Latino, 193,588 for Black, 119,457 for Middle Eastern, 80,407 for Southeast Asian, and 34,402 for Indian.

Table 4.1: Gender in the GazeCapture dataset

Gender	Number of Participants	Number of Images
Female	671	1,087,498
Male	803	1,358,006
Total	1,474	2,445,504

Upon further examination of the number of participants and images for each gender by race, a consistent trend emerges. The male groups have a greater number of participants and more images when compared to their respective female groups. And this trend is observed across all races in the dataset. These corresponding results are shown in Table 4.3.

The dataset was divided into a training, validation, and testing set. The training set contains 1274 subjects, the validation set contains 50 subjects, and the testing

Table 4.2: Race in the GazeCapture dataset

Race	Number of Participants	Number of Images
Black	134	193,453
East Asian	233	395,361
Indian	29	34,372
Latino	136	222,325
Middle Eastern	75	119,384
Southeast Asian	45	80,362
White	822	1,400,247
Total	1474	2,445,504

Table 4.3: Race and Gender in the GazeCapture dataset

Gender	Race	Number of Participants	Number of Images
Female	Black	81	73,441
	East Asian	126	177,981
	Indian	17	17,304
	Latino	83	84,652
	Middle Eastern	56	26,747
	Southeast Asian	33	19,701
	White	407	688,342
Male	Black	53	120,147
	East Asian	107	217,617
	Indian	12	17,098
	Latino	53	137,810
	Middle Eastern	19	92,710
	Southeast Asian	12	60,706
	White	415	712,731

set contains 150 subjects. Additionally, the validation and testing sets only contain participants that looked at all the gaze locations to help ensure that there was a uniform distribution in these sets. This allows the model to perform a "thorough evaluation on the impact of calibration across these subjects" [23]. This splitting was done in the preexisting code from GazeCapture.

Upon inspection of the number of participants within each of the training, testing, and validation datasets, there is a familiar pattern in that there are more males than females. This information can be seen in Table 4.4. Additionally, when looking at the race variables within each dataset the results are again heavily skewed for the

White groups. White participants make up 60% of the testing dataset, about 55% of the training dataset, and 66% of the validation dataset.

Table 4.4: Gender in the training, testing, and validation datasets

Dataset	Female	Male	Number of Participants
test	72 (48%)	78 (52%)	150
train	579 (45.45%)	695 (54.55%)	1,274
validation	20 (40%)	30 (60%)	50
Total	671	803	1,474

Table 4.5: Race in the training, testing, and validation datasets

Race	Test	Train	Validation	Total
White	91 (60.67%)	698 (54.79%)	33 (66%)	822
Black	11 (7.33%)	120 (9.42%)	3 (6%)	134
Latino	14 (9.33%)	119 (9.34%)	3 (6%)	136
Middle Eastern	5 (3.33%)	67 (5.26%)	3 (6%)	75
Indian	5 (3.33%)	24 (9.34%)	0	29
East Asian	22 (14.67%)	206 (16.17%)	5 (10%)	233
Southeast Asian	2 (1.33%)	40 (3.14%)	3 (6%)	45
Number of Participants	150	1,274	50	1,474

Lastly, the dataset was created for both iPhone and iPad users. For these two types of devices, there were 1,250 participants that used iPhones to create their images and 224 participants that used iPads. For iPhone users, there were 8 different iPhone models that could be used. The oldest of these models was the iPhone 4S, while the newest iPhone used in the dataset was the iPhone 6S Plus. When comparing the users within the dataset, male participants used the iPhone 6 most frequently, while female participants used the iPhone 5S the most frequently. All of the other iPhone devices were used around a similar percentage. These results can be seen in Figure 4.6. Upon further investigation of the usage of the iPhone devices by race, almost all groups used the iPhone 5S and iPhone 6 the most, with the exception of the Indian participants. They most frequently used the iPhone 4S, the oldest iPhone in the dataset. This could be problematic because the iPhone 4S

had a lower-quality camera compared to the newer devices, which could lead to a lower-quality image that might make it harder for the convolution neural network to make predictions. These results can be seen in Figure 4.7 where the white spaces show that no participants in that group used that device.

Then, when investigating the usage of iPads within the GazeCapture dataset I found that there were 7 different iPad models used. The iPad Mini is the most frequently used iPad within the dataset, while the iPad Pro is the least frequent iPad device. When looking at the usage between male and female participants, both males and females tended to use the iPad Mini most often and the iPad 3 and iPad Pro the least often. These results can be found in Figure 4.8. I find it interesting that the iPad 3 had fewer users than the iPad 2 since the iPad 3 was released more recently. Additionally, when looking at the usage between different races, which is shown in Figure 4.9, the most popular iPad changes for each race. White participants almost use each iPad equally, while Latino participants use the iPad Air 2 most often. Also, both Black, East Asian, and Indian participants use the iPad Mini most often, while Middle Eastern and Southeast Asian participants used the iPad 4 and iPad Air the most.

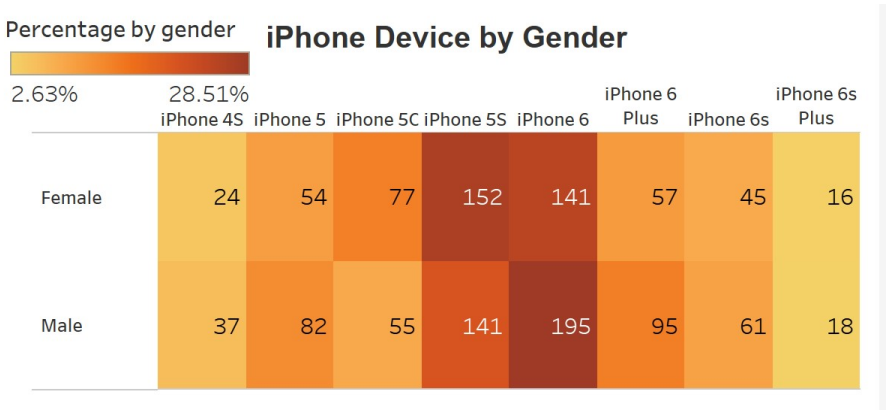
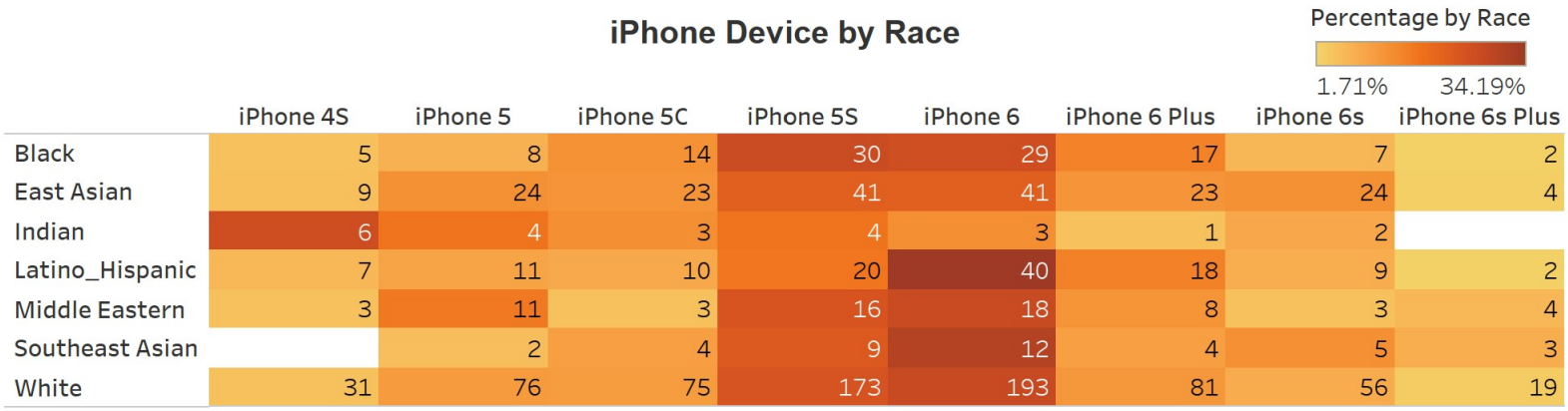


Figure 4.6: Number of participants in the GazeCapture dataset using an iPhone device by gender



	iPhone 4S	iPhone 5	iPhone 5C	iPhone 5S	iPhone 6	iPhone 6 Plus	iPhone 6s	iPhone 6s Plus
Black	5	8	14	30	29	17	7	2
East Asian	9	24	23	41	41	23	24	4
Indian	6	4	3	4	3	1	2	
Latino_Hispanic	7	11	10	20	40	18	9	2
Middle Eastern	3	11	3	16	18	8	3	4
Southeast Asian		2	4	9	12	4	5	3
White	31	76	75	173	193	81	56	19

Figure 4.7: Number of participants in the GazeCapture dataset using an iPhone device by race

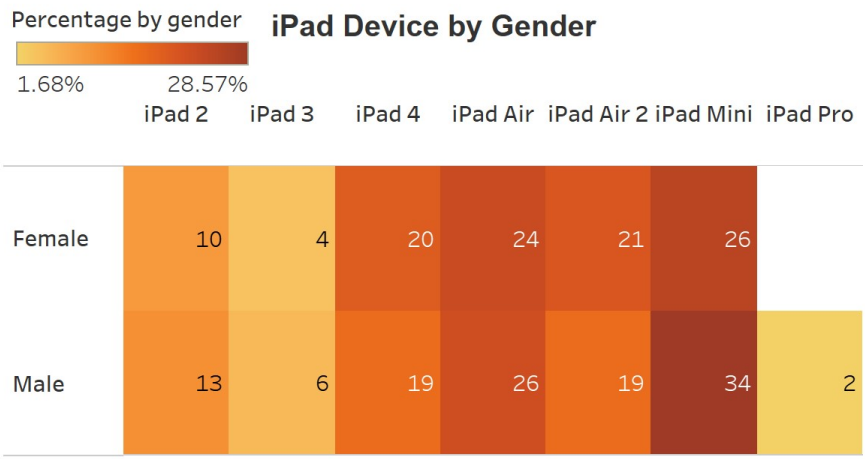


Figure 4.8: Number of participants in the GazeCapture dataset using an iPad device by gender

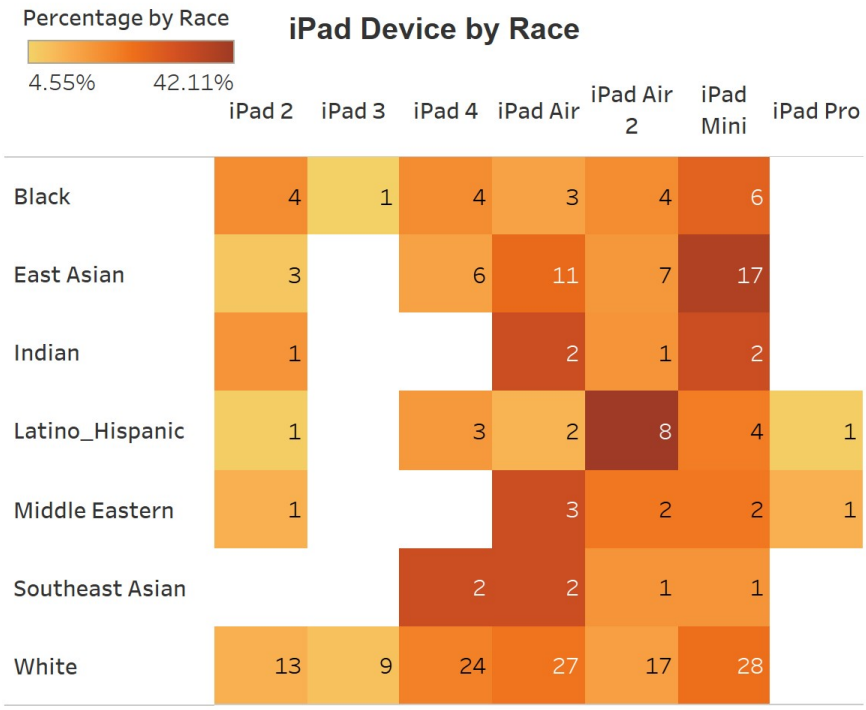


Figure 4.9: Number of participants in the GazeCapture dataset using an iPad device by race

The usage of these devices is important because they hold a hidden bias because this product cost a large amount of money. Therefore wealthier people, which often tend to be White, are more likely to purchase newer devices that have higher-quality

cameras. With a higher-quality camera, more information can be found within the images, which could lead to more accurate predictions. Therefore, it is crucial to be able to know the demographic information so that these types of biases can be discovered and discussed. In the next section, we will discuss the architecture of the GazeCapture model and how the metrics of the race and gender variables were made.

CHAPTER 5

Computational Framework

5.1 Implementation

The GazeCapture model requires many libraries and dependencies in order to run properly. First, the most important of these dependencies is the Python library. All of the code used for my study was written in Python. The remaining dependencies are within the frontend and backend groups of the code. The frontend includes the functions and the model that are written within the python code. This code can be found on my GitHub `https://github.com/ameyer23-m`. For example, my image processing functions that define how the images are converted are part of this frontend group. Then, the backend group consists of the libraries and functions used to write and perform the frontend code. For example, within my image processing functions, I use the OpenCV library to use a predefined function. Therefore, OpenCV is one of these backend requirements. Another key backend dependency used in this study is the PyTorch library. The PyTorch library performs the complex calculations needed to perform neural networks. Additionally, the PyTorch library is extremely useful when working with a graphic processing unit (GPU) because it is optimized for GPU acceleration, which makes it particularly efficient when working with deep learning. Without this library, I would have had

to write all the intricate calculations myself. But resourceful libraries like this help maximize productivity so that each programmer does not have to rewrite their own calculations for every project. The list of dependencies and packages needed to run the code can be found in the Appendix in Section 9.4.

Two of these dependencies, CUDA and NVIDIA GPU Driver are required for training the neural networks on the GPU rather than the computer's central processing unit (CPU). GPUs are specialized hardware designed to accelerate the types of calculations that neural networks require. CUDA, or Compute Unified Device Architecture, is a parallel computing platform that allows GPUs to process their parallel calculations. They are most popular for their capabilities in gaming because they can increase performance speeds, but they have recently become more popular in AI as the calculations have become more complex causing the time of these calculations to increase. CPUs do not have this accelerated calculation performance, therefore making them less efficient for complex calculations. Even with the GPU the full code takes about 3 days to run. Meaning that it imports the data, and runs the model through 5 epochs, meaning it runs through the entire dataset 5 times. Usually, neural networks have 25-50 epochs, but due to the time frame of this study, I had to limit that number. Using the CPU would approximately require about 10 days for the code to run the GazeCapute model.

5.2 GazeCapture iTracker Model

The GazeCapture iTracker model utilizes a complex convolutional neural network so it can produce highly accurate predictions. The model produces three images from each image in the dataset that it is examining and then uses four different types of layers throughout the network. In this section, I will describe the full process of how an image produces a predicted gaze location output. First, I will discuss the

preprocessing steps, including the different image processing algorithms and the changes made to the inputted images. And additionally, I will explain the complex architecture of the iTracker model.

In the preprocessing steps of the model, the images are first converted to either grayscale, binarization, or edge detection. When investigating the RGB images, they do not need to be converted because the images within the GazeCapture dataset are already in the RGB format which has 3 channels, one for the red color component, one for the blue color component, and one for the green color component.

The images are saved as a torch tensor object which is a multi-dimensional array that the PyTorch library utilizes. Tensors are an array, but they have additional features to work with the GPU for faster computation. Additionally, due to complications with the model I made sure the images that were inputted had three channels. Three channels were used instead of one channel due to time constraints. This does not change the type of image used because 3 channels with all the same value at each location in the image array have the same result.

After the image is converted to one of the types of image processing, the image utilized is reformatted to create three separate images: the face, left eye, and right eye. The face image is a 224×224 image that removes all data outside the face grid parameters. These face grid parameters are also given in the dataset. They are a "binary mask used to indicate the location and size of the head within the frame" [23], which is of size 25×25. Since the face image only shows information from within the face grid parameters, the model does not have to worry about other noise in the background, such as other faces. Additionally, there are images of each of the left and right eyes, which are also of size 224×224, because they incorporate the most significant information about where the participant is looking.

All three of the images are then inputted into the single image model where both eyes share weights. This model is shown in Figure 5.1. First, the model includes

a convolutional layer that requires 3 input channels and results in an output with 96 channels. 3 channels are the number of channels used in an RGB image, while grayscale and edge detection images usually tend to have only 1 channel. The non-RGB images needed to be computed in a manner that provided them with three channels. This method is a fast solution because it replicates the grayscale image three times, and it does not add any additional information. Yet, one problem is that it can lead to poor performance because the model will learn on redundant features [46]. Additionally, the layer has a kernel size of 11, meaning that the convolution filter is of size 11×11. The layer also has a stride of 4, meaning the filter shifts over 4 pixels for each convolution. And lastly, the layer has a padding of 0, meaning that it does not add any extra rows or columns around the image to affect the output size of the image. Then the image is convolved by the layer and the output is applied to the ReLU activation function so that negative values are not activated, and hence they do not go into the next layer.

The output from the ReLU activation function then serves as the input for the next layer which is a max pooling layer. The max pooling layer has a kernel size of 3, meaning that the filter is of size 3×3. Then, the layer also has a stride of 2. Therefore, the max pooling results in the max value of each 3×3 patch of the image when the patch is shifted by 2 pixels each time it is applied. After that, the output from the max pooling is utilized as the input for the local response normalization layer. The LRN layer has a size of 5, meaning that 5 adjacent layers are considered for the normalization process. Then, it has an alpha value of 0.0001, which refers to the level of normalization. Smaller values reduce that amount. Then the LRN layer has a beta value of 0.75 which also affects the amount of normalization such that a smaller value also decreases it. Lastly, the LRN layer has a k value of 1, meaning that the size of the normalization does not need to increase. Then, these steps are again repeated, with the output from the LRN going to a new convolution layer

that requires an image with 96 input channels. The new layer has a kernel size of 2, a stride of 1, and a padding of 2, meaning that it adds two levels of zeros around the border of the image to maintain the desired size. This convolution layer also incorporates groups of 2. The group parameter controls the connections between the input and the output channels, such that it divides the number of groups that the channels are divided into. This layer results in an even deeper output image with 256 channels. Figure 5.1 denotes this depth by having layers where the image has more channels to be wider. Additionally, the height of these layers in Figure 5.1 indicates the size of the image at each layer, such that the size is decreasing after each convolutional layer. The output from the convolution layer is then applied to the ReLU activation function which is then used for the input to another max pooling layer and LRN layer, which both have the same parameters as their previous versions. Following the LRN layer, the output is then fed into another convolution layer that requires an image with 256 channels. Additionally, this convolution layer has a kernel size of 3, a stride of 1, and a padding of 1. The layer then reformats the image to incorporate 384 channels. Again, this output is applied to another ReLU activation function, whose output is fed into one last convolution layer that requires an image with 384 channels. This layer also has a kernel size of 1, a stride of 1, and a padding of 0. The output of this layer is an image with 64 channels which is then again applied to another ReLU activation function.

Then the face image uses two linear layers that are both followed by ReLU activation functions. The first linear layer takes in 9216 input features and results in an output with 128 features. Then the next linear layer allows an input of 128 features and results in an output of 64 features.

The face grid parameters also run through the model just using two linear layers that both use a ReLU activation function. The first linear layer takes in 625 input

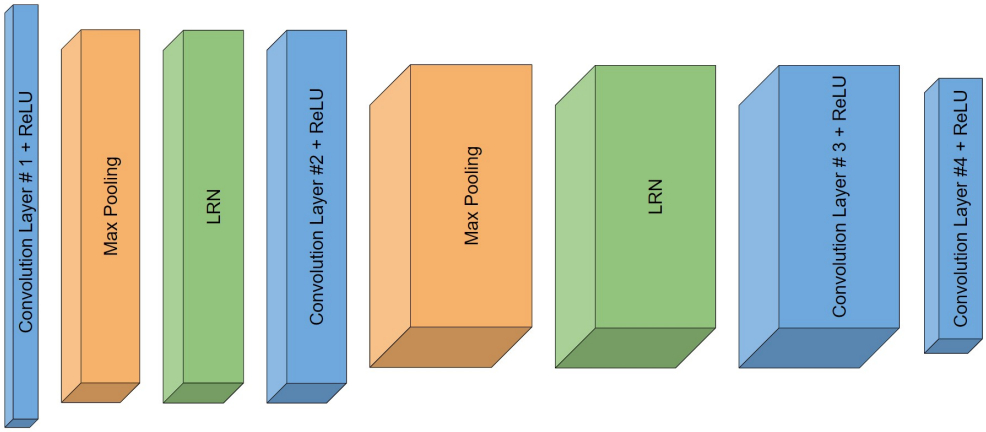


Figure 5.1: The convolutional neural network model that each image goes through before being combined for the full model

features and results in an output with 256 features. Then the next linear layer allows an input of 256 features and results in an output of 128 features.

After the two eye images completely run through the image model they are both combined using a linear layer that connects every input neuron to every output neuron. This linear layer requires an input of 18,432 features and results in an output of 128 features. This linear layer also uses a ReLU activation function so that only positive features activate.

Now that all the individual images and the face grid parameters have run through their individual models they are all combined using two more linear layers, but only the first layer uses a ReLU activation function. The first linear layer 320 features which is the combination of the output from the other three previous linear layers. Then this layer results in an output of 128 features. This output is then fed into one final linear layer that results in two features, an x and y position of the predicted gaze location. The full layout of this entire model is shown in Figure 5.2.

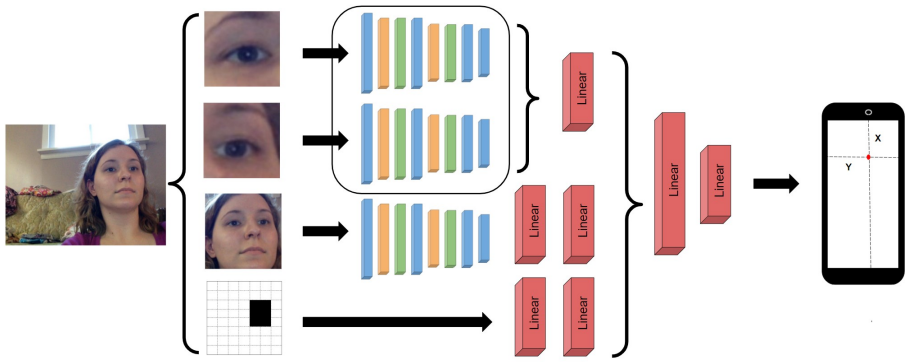


Figure 5.2: GazeCapture's iTracker convolutional neural network model

CHAPTER 6

Results

In this chapter, I examine the results of each type of image processing and how they affected the overall accuracy of the model. Additionally, I will also examine the results of the algorithm fairness for each gender and race group from the GazeCapture dataset.

Before performing the convolutional neural network, the dataset was divided into a training, validation, and testing set. The training set contains 1,274 subjects, the validation set contains 50 subjects, and the testing set contains 150 subjects. Additionally, the validation and testing sets only contain participants that looked at all the gaze locations to help ensure that there was a uniform distribution in these sets. This allows the model to perform a "thorough evaluation on the impact of calibration across these subjects" [23]. This splitting was done in the preexisting code from GazeCapture.

6.1 Accuracy

The model outputs an error value, which is the average Euclidean distance in centimeters from the actual gaze location to the predicted gaze location. Table 6.1 displays the error of each type of image processing. On average, the model deviates by 2.7172 cm when utilizing RGB images with color. Next, the grayscale methods

range from 2.7568 cm to 2.8757 cm. Then, when using Canny and Sobel edge detection the model on average deviates by 3.0457 cm and 3.4103 cm respectively. Lastly, when using the Otsu binary method the model on average deviates by 3.3723 cm.

Table 6.1: Overall accuracy for GazeCapture data

Image Processing	Method	Error (cm)
Color	RGB	**2.7172**
Grayscale	Intensity	2.7736
	Luminance	2.7780
	Luster	2.7568
	Value	2.8757
Binary	Otsu	3.3723
Edge Detection	Canny	3.0457
	Sobel	3.4103

6.2 Gender Fairness

When calculating gender fairness I used a two-sample t-test to determine if the average error for males was equivalent to the average error for females. When conducting t-tests there are three conditions that must be met. First, the samples must be independent, meaning that each of the subjects can only belong to one of the gender groups. Each participant was only classified as a male or female and they could not be classified as both. Therefore, this first condition is met. Secondly, the samples from each group should be normally distributed. The predictions appear to all be near the mean of their gender group and there do not appear to be any outlying predictions that are more inaccurate than the others. Therefore, the samples are normally distributed and the second condition is met. And third, the samples should have equal variance or at least approximately equal variances. I found that the standard deviations for both the male and female groups are similar for each

method. For example, the RGB method male group had a standard deviation of 1.0601 cm while the female group had a standard deviation of 1.2607 cm. Each image processing method showed similar values in that the male standard deviation was slightly smaller than the female deviation. These standard deviations along with the sample sizes for each group can be found in the Appendix in Table 9.1. Therefore, this third and final condition is met.

The male and female average errors and their corresponding t-statistic and p-value are shown in Table 6.2. When using Canny edge detection the p-value is higher than 0.05, meaning that I do not have significant evidence to conclude that the alternative hypothesis, $\mu_{males} \neq \mu_{females}$, is true. Canny edge detection finds an average error of 3.0457 cm for males and 3.0453 cm for females. The corresponding p-value is 0.9470, meaning that the probability of obtaining a result as extreme or more extreme, assuming the null hypothesis is true, is 94.7% of the time. Thus showing that this model is fair for the gender variables.

Additionally, the Intensity grayscale image processing method also produces a p-value greater than 0.05. It finds an average error of 2.7785 cm for males and 2.7691 cm for females. The corresponding p-value is 0.0946, meaning that the probability of obtaining a result as extreme or more extreme, assuming the null hypothesis is true, is 9.46% of the time. But, if one was to use a p-value threshold of 0.1 then the results would be significant enough to conclude that the alternative hypothesis is true. However, for this study, I am using the 0.05 threshold.

The most accurate overall model was found when using RGB images. When using the colored RGB images, the model on average deviates by 2.6927 cm for males and 2.7384 cm for females. The t-test results in a p-value of approximately 0, meaning that there is significant evidence to reject the null hypothesis that the two groups are fair and the model is more accurate for males.

Sobel edge detection and the luminance grayscale methods both found similar

results to the RGB method. They had more accurate predictions for the male groups compared to the female groups. For Sobel, I found an average error of 3.3541 cm and 3.4618 cm for males and females respectively, which resulted in a p-value of approximately 0. Then for luminance, I found an average error of 2.7696 cm and 2.7857 cm for males and females respectively. These values are closer than some of the other predictions, but they did have a significant enough p-value to show that there is a difference in the two averages.

I found interesting results for the luster and value grayscale methods. Both of these methods found more accurate predictions for the female groups. For luster, I found an average error of 2.7639 cm and 2.7503 cm for males and females respectively, which resulted in a p-value of approximately 0.0142. This p-value is not above the 0.05 threshold, so I cannot conclude that the two averages are equivalent enough to be fair. Then for the value grayscale method, I found an average error of 2.8894 cm and 2.8632 cm for males and females respectively, which resulted in a p-value of approximately 0.

Lastly, when using the Otsu binary method the model results in an average error of 3.3480 cm and 3.3946 cm for males and females respectively. This method also had a large enough difference in the average errors, therefore I cannot conclude that this method is fair for the gender variable.

Table 6.2: Gender fairness for GazeCapture data

Image Processing	Method	Male Error (cm)	Female Error (cm)	T-Statistic	P-Value
Color	RGB	2.6942	2.7384	-8.0612	< 0.0000
Grayscale	Intensity	2.7785	2.7691	1.6716	**0.0946**
	Luminance	2.7696	2.7857	-2.8607	0.0042
	Luster	2.7639	2.7503	2.4519	0.0142
	Value	2.8894	2.8632	4.7397	< 0.0000
Binary	Otsu	3.3480	3.3946	-6.7968	< 0.0000
Edge Detection	Canny	3.0457	3.0453	0.0664	**0.9470**
	Sobel	3.3541	3.4618	-16.1141	< 0.0000

6.3 Racial Fairness

When calculating racial fairness I used an ANOVA test to determine if the average error for each of the seven races were equal. When conducting ANOVA tests, it follows the same three conditions as the t-test. First, the samples must be independent, meaning that each of the subjects can only belong to one of the racial groups. Each participant was only classified as one race, White, Black, Latino, Middle Eastern, Indian, East Asian, or Southeast Asian, and they could not be classified as both. Therefore, this first condition is met. Secondly, the samples from each group should be normally distributed. The predictions again appear to all be near the mean of their racial groups and there do not appear to be any outlying predictions that are more inaccurate than the others. Therefore, the samples are normally distributed and the second condition is met. And third, the samples should have equal variance or at least approximately equal variances. I found that the standard deviations for all racial groups are similar. These standard deviations and sample sizes for each racial group can be found in the Appendix in Tables 9.2 and 9.3. Therefore, this third and final condition is met.

The race average errors and their corresponding f-statistic and p-value are shown in Table 6.3. According to the table, the analysis of the image processing methods did reveal significant differences in the average errors between the different racial groups. For the RGB images, I found an average error of 2.8250 cm for White participants, 2.2677 cm for Black participants, 2.7685 cm for Latino participants, 2.2528 cm for Middle Eastern participants, 2.5575 cm for Indian participants, 2.6803 cm for East Asian participants, and 2.5811 cm for Southeastern participants. I obtained a p-value of around 0, indicating that there is significant evidence to conclude that the alternative hypothesis that the means of at least one racial group are not equal is true.

Next, when looking at the grayscale methods I found similar results. All of

these methods had a large enough range of values for the racial groups to lead to a p-value of approximately 0, meaning that the null hypotheses are rejected and we can conclude that the alternative hypothesis, the means of the racial groups are not equal, is true.

Then, when looking at the Otsu binary method I again found the same results. I found an average error of 3.4783 cm for White participants, 2.6766 cm for Black participants, 3.2789 cm for Latino participants, 2.8124 cm for Middle Eastern participants, 2.8043 cm for Indian participants, 3.5377 cm for East Asian participants, and 3.3017 cm for Southeastern Asian participants. With this wide range of average error prediction, I found a p-value of approximately 0, meaning that the model is not fair for the race variable.

Lastly, when looking at the edge detection methods I found the same recurring results. For the Canny images, I found an average error of 3.1926 cm for White participants, 2.5299 cm for Black participants, 2.8994 cm for Latino participants, 2.4812 cm for Middle Eastern participants, 2.5530 cm for Indian participants, 3.0560 cm for East Asian participants, and 2.9074 cm for Southeastern Asian participants. And then for the Sobel images, I found an average error of 3.5789 cm for White participants, 2.7425 cm for Black participants, 3.3423 cm for Latino participants, 2.7469 cm for Middle Eastern participants, 2.9198 cm for Indian participants, 3.4175 cm for East Asian participants, and 3.1618 cm for Southeastern Asian participants. Both of these two edge detection methods found ranges of average error predictions. Therefore, I found a p-value of approximately 0, meaning that the model is not fair for the race variable.

Table 6.3: Racial fairness for GazeCapture data

Image Processing	Method	White Error (cm)	Black Error (cm)	Latino Error (cm)	Middle Eastern Error (cm)	Indian Error (cm)	East Asian Error (cm)	Southeast Asian Error (cm)	F-Statistic	P-Value
Color	RGB	2.8250	2.2677	2.7685	2.2528	2.5575	2.6803	2.5811	734.5318	< 0.0000
Grayscale	Intensity	2.8969	2.3846	2.7335	2.2138	2.6106	2.7301	2.7172	814.8035	< 0.0000
	Luminance	2.9100	2.3409	2.6549	2.3387	2.6848	2.7541	2.5814	759.5421	< 0.0000
	Luster	2.8804	2.2811	2.7097	2.2350	2.3226	2.7496	2.6697	882.2332	< 0.0000
	Value	3.0051	2.4565	2.7114	2.4482	2.5574	2.8697	2.7319	769.2345	< 0.0000
Binary	Otsu	3.4783	2.6766	3.2789	2.8124	2.8043	3.5377	3.3017	879.1980	< 0.0000
Edge Detection	Canny	3.1926	2.5299	2.8994	2.4812	2.5530	3.0560	2.9074	956.1833	< 0.0000
	Sobel	3.5789	2.7425	3.3423	2.7469	2.9198	3.4175	3.1618	1115.0969	< 0.0000

CHAPTER 7

Discussion

In this chapter, I discuss the key findings from the previous chapter. First, I will compare the accuracies of the different image processing models. Next, I will compare the algorithm fairness of models for both the gender and racial groups. Then, I will discuss how these results are similar to other research. And lastly, I will discuss the limitations of this study and potential areas for future investigation.

7.1 Accuracy

The RGB model found the most accurate predictions. This was assumed because the RGB image contains the most information out of all the images. After all, it has three different channels with information on the red color component, green color component, and blue color component. The average error was 2.7172 cm. To get a better understanding of this distance Figure 7.1 shows an iPhone 6 with a dark red predicted gaze location and a larger light red circle with a radius of 2.7172 cm. For the predicted location, I expect that the actual gaze location is somewhere near the border of the large circle. While this might not seem accurate, the model shows significant growth from the first few predictions. The first batch in the first epoch of training tended to create a prediction error that was off by 22 cm, which

is larger than the dimensions of any iPhone and some iPads. This shows that the convolutional neural network can learn significantly over time.

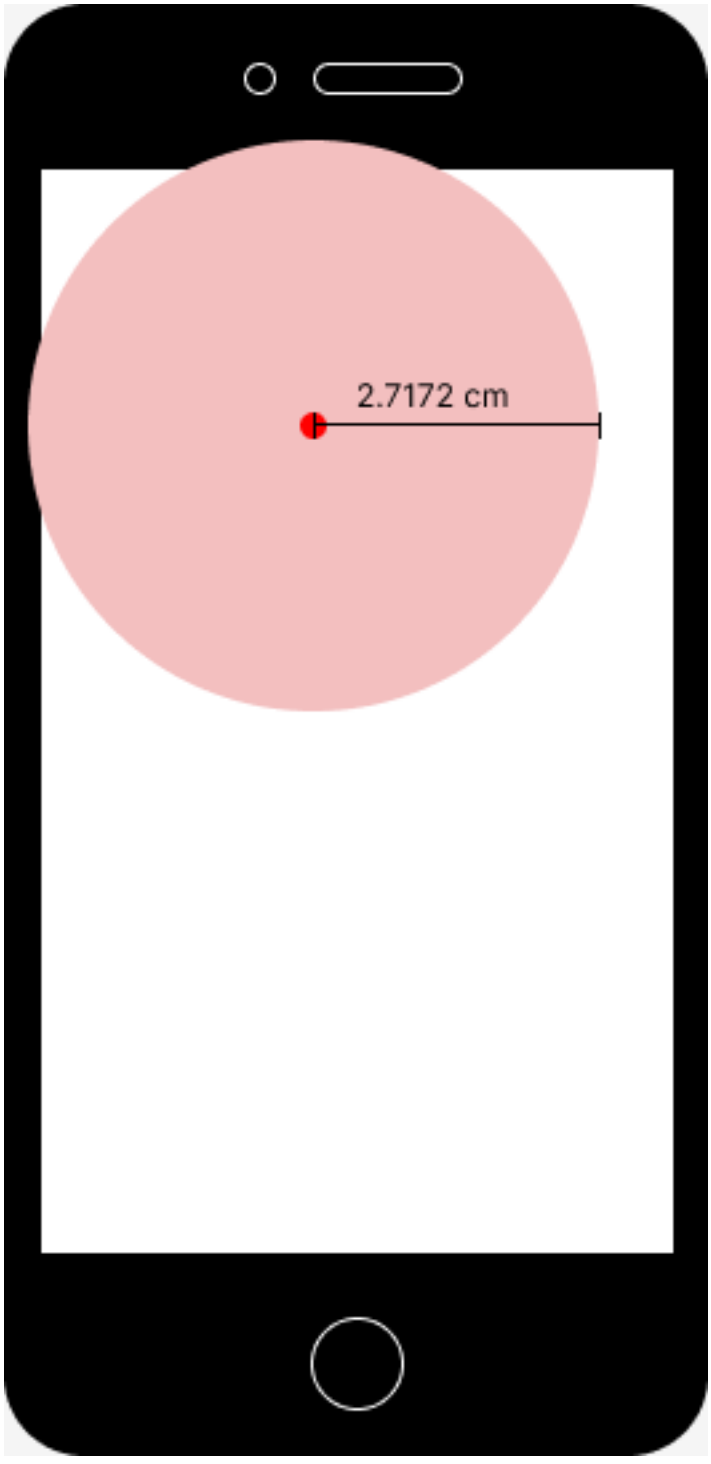


Figure 7.1: RGB results compared to an iPhone 6

The other types of image processing methods still had three channels, but each of the channels had the same value, which was just repetitive for the model to train on. After the RGB, the grayscale methods were the next most accurate type. Luster was the most accurate grayscale, followed by intensity, followed by luminance, and then the value method was the least accurate grayscale with an average error distance of 2.8757 cm. All these average prediction errors were within 0.1585 cm of the RGB predictions. To get a better understanding of how changing the image processing affected the predictions, Figure 7.2 shows an iPhone 6 with a dark red predicted gaze location and a larger light blue circle with a radius of 2.8757 cm. Figure 7.3 shows the four different types of grayscale methods. I find it interesting that luster was the most accurate out of these methods when it gave the faces an unnatural look. To the human eye, the skin tone of the participant is distorted by being darker except when near the eyes, nostrils, mouth, and jaw. Additionally, the value method looks very similar to the luminance and intensity methods, but it has a significantly higher average error distance than the rest.

Then, Canny's edge detection was also less accurate than RGB with a difference of 0.34 cm, while Sobel was 0.7 cm less accurate than RGB. Their average errors can be seen in Figure 7.4 where they are compared with an iPhone 6. The orange circle indicates the radius for the Canny average error distance and the green circle indicates the radius for the Sobel average error distance. In Figure 7.5 Sobel exhibited a higher number of edges than the Canny method. Based on these two images, the presence of fewer edges in the Canny image, allowed the Canny model to generate more precise outcomes. But these were still less precise than the RGB method.

Lastly, I was surprised that the Otsu binary method was not found to be the least accurate image processing method. This was surprising since the image contains the least amount of information. Additionally, having a predefined threshold could

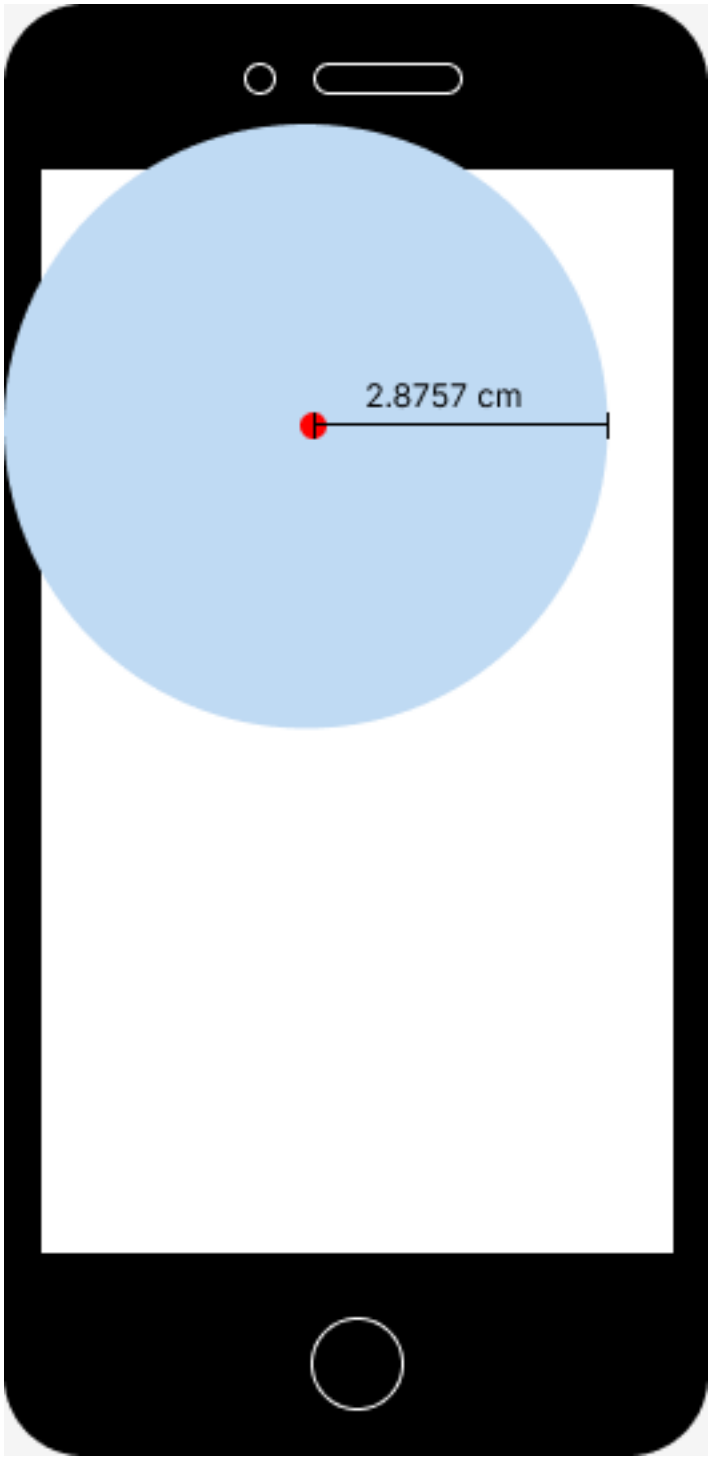


Figure 7.2: Value grayscale results compared to an iPhone 6

cause an entire image to be either all black or all white if all the pixels in the image are above or below the threshold value. The Otsu method found an average error

Figure 7.3: Grayscale methods on a GazeCapture participant

of 3.3723 cm. To get a better understanding of this distance Figure 7.6 shows an iPhone 6 with a dark red predicted gaze location and a larger light purple circle with a radius of 3.3723 cm.

The results from the Otsu binary method show how powerful convolutional neural networks are. To the human eye the Otsu image, which is shown in Figure 7.7, is almost unrecognizable. Yet the neural network can find underlying patterns within an image that is unrecognizable to the human eye to create fairly accurate predictions.

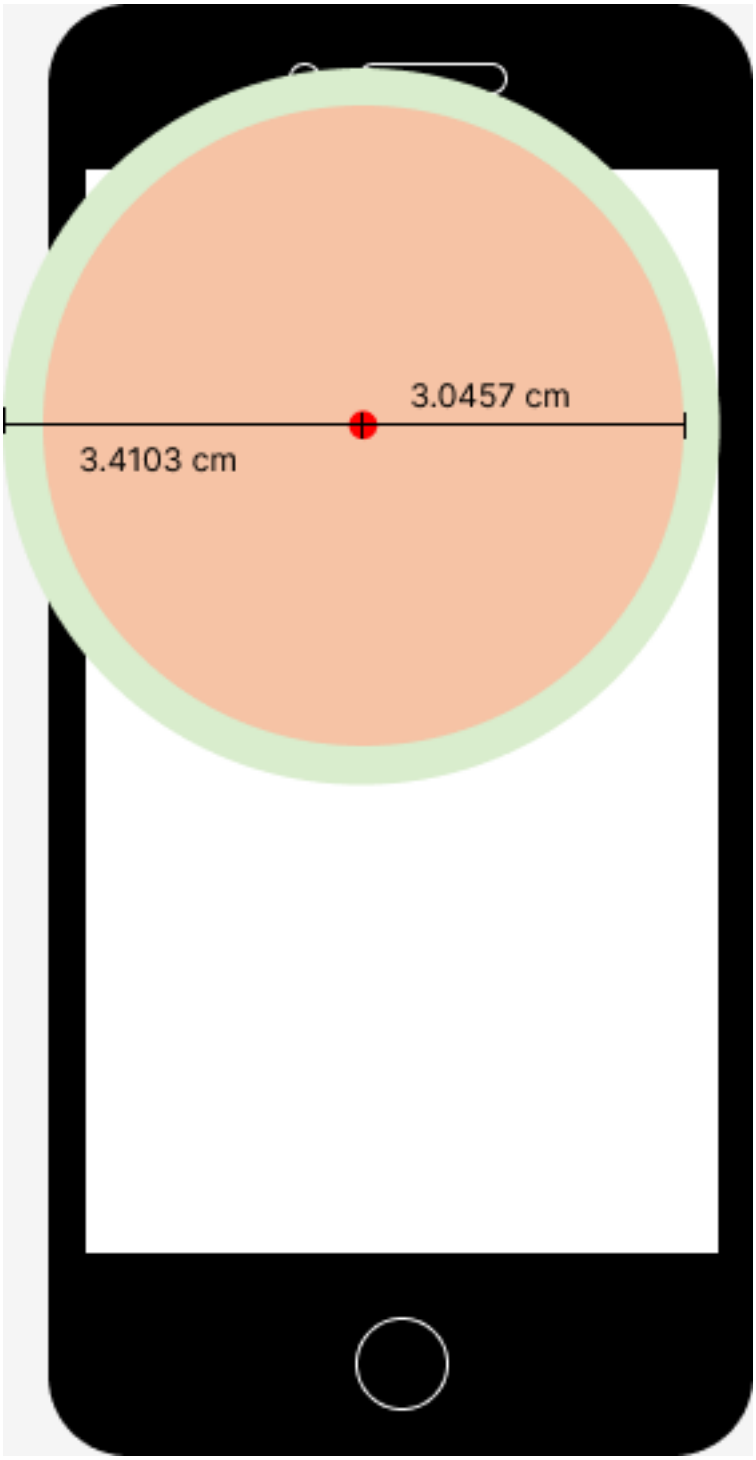


Figure 7.4: Edge Detection results compared to an iPhone 6

Figure 7.5: Edge Detection methods on GazeCapture participant

7.2 Algorithm Fairness

The results of the gender fairness metrics found insightful conclusions, especially for Canny edge detection. When performing the model on Canny I found a p-value of 0.947, meaning that the probability of obtaining a result as extreme or more extreme, assuming the null hypothesis is true, is 94.7% of the time. Changing the image from RGB to Canny made a fair gender algorithm. Similarly, intensity grayscale resulted in the conclusion that the model would be fair 9% of the time since it returned a p-value of 0.0946. While this value is not as large as the Canny p-value, it is still significant enough for the model to be fair when the p-value threshold is at 0.05. Additionally, both models predicted the female group slightly more accurately than the male group. This is surprising since there were more male participants within the dataset. In the past, research tended to find that the more frequent group would produce more accurate results, yet this does not appear to be the case here. Instead, the equal numbers within the testing dataset produce exactly equal prediction accuracies.

These results from the Canny and intensity methods are similar to the results found in [8]. Givens et al. found that gender did not affect the model's fairness, meaning that there was no bias towards male or female participants.

Canny edge detection and intensity grayscale produces slightly more accurate

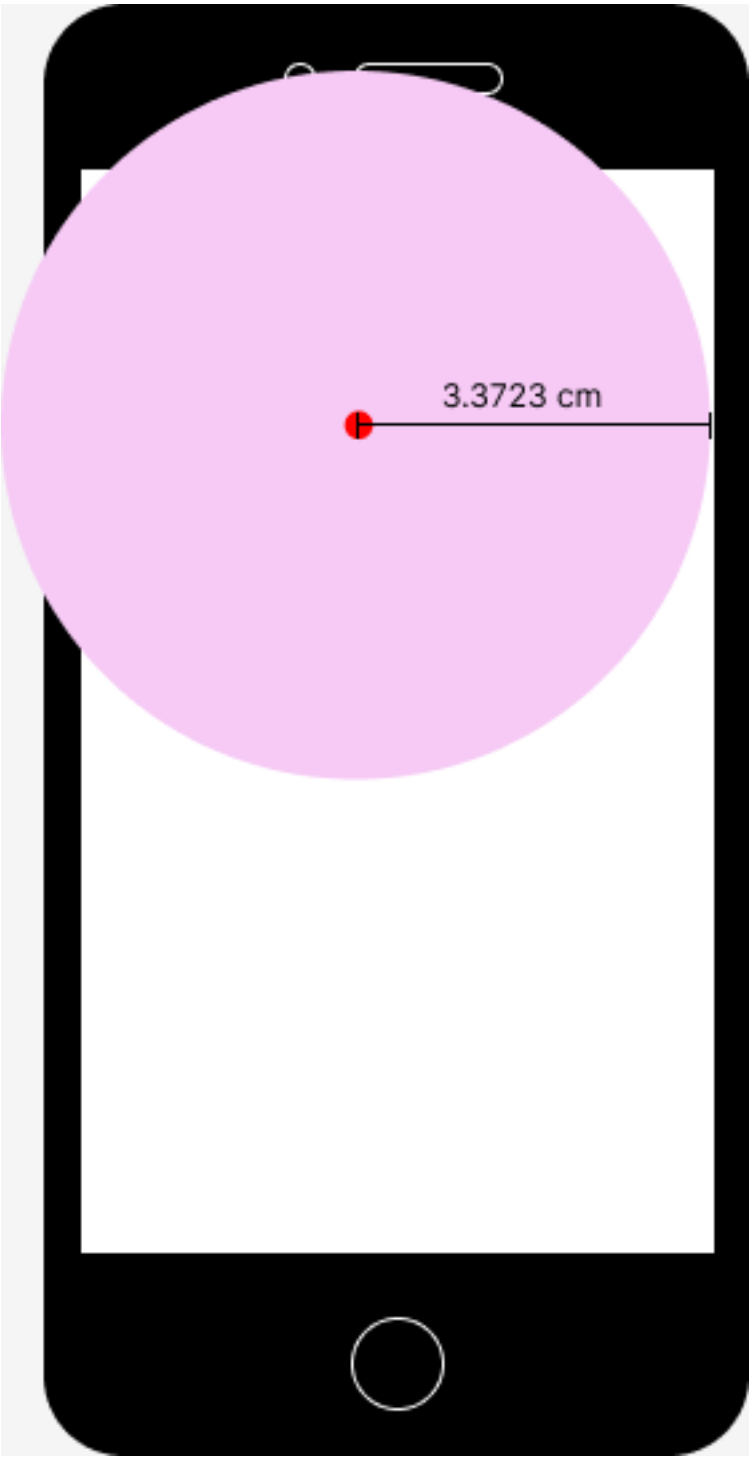


Figure 7.6: Otsu binary method results compared to an iPhone 6

results for females, but they were not the only methods to produce more accurate results for the female participants. Luster and value grayscale methods were also

Figure 7.7: Otsu image processing

more accurate for the female group, but the differences in their average errors were statistically significant enough to be unfair. All the other image processing models resulted in unfair gender algorithms that were in favor of the male group. Therefore, if technology uses these unfair image processing models, then the less accurate gender could have more frequent mistakes which would limit their ability to use this type of technology.

Next, when investigating the results of the racial fairness metrics no image processing model was fair. All of them produced a p-value of approximately 0, meaning that the probability of achieving fair results is 0% of the time. This is highly concerning because so many models have been racially unfair in the past. And these unfair models have then led to certain groups being affected negatively. While I can think of no dangerous repercussions of an unfair eye gaze model, it would certainly be inconvenient and frustrating for less accurate groups.

It is interesting to note that despite the models being trained and tested mostly on White participants, 55% for the training dataset and 60% for the testing dataset, all models except for Otsu have less accurate average predictions for White participants compared to all other races. The most accurate predictions for all models, except Canny and Otsu, are for Middle Eastern participants. They made up 3% of the training participants and 5% of the testing participants. Additionally, the next best

accurate predictions were for Black participants, who made up 7% of participants in the training dataset and 9% in the testing dataset. Next, Indian participants were the third most accurate group for most models. Indian participants made up only 3% of all participants in the training set and 2% in the testing dataset. The Southeastern Asian, Latino, and East Asian participants were more inconsistent between the models, but only in the Otsu method the East Asian participants had the worst predictions. This result is interesting since East Asians and Latinos had the second and third highest percentage of participants and number of images respectively within the dataset.

These results are similar to [8], in that Givens et al. found that race did affect algorithm fairness. Additionally, they found that some of their models were more accurate for non-white participants. Givens et al. and my results are inconsistent with other research because many models in the past have been racially biased in favor of White participants. While my results might be good for non-White participants, it is still unfair for White participants. This is still just as much a problem because the goal is to create machine learning models that are fair for all racial groups. Therefore, this suggests that there is still much work needed to achieve a racially fair result.

7.3 Limitations

First, my research has been conducted using the GazeCapture dataset that was collected in 2016. This indicates that these findings from my research can only be applied to those participants within the dataset. Additionally, this dataset was created in a wild environment, meaning that there was no lab assistance walking the participant through the creation process. Therefore, there could be errors within the dataset, like fingers over cameras, glares in the background, or blurred images.

This could affect the accuracy of the models since these mistakes could withhold the model from making accurate predictions. Additionally, this dataset only used the iPhone 4S, iPhone 5, iPhone 5C, iPhone 5S, iPhone 6, iPhone 6 Plus, iPhone 6S, iPhone 6S Plus, iPad 2, iPad 3, iPad 4, iPad Air, iPad Air 2, iPad Mini, and iPad Pro devices. Therefore, these results can only be assumed for these older iPhone and iPad devices. Technology has advanced significantly in the past few years. As newer models of the iPhone and iPad come out, the cameras can take images with more pixels. These higher-pixel images contain more information than the images from these older devices, which essentially makes these results only useful for those who still use the older devices.

As mentioned earlier in the study, my convolutional neural network was limited to using 5 epochs instead of the 25 epochs that Krafka et al. originally used in the GazeCapture study. Most convolutional neural networks that use large datasets like the one used in this study tend to use larger amounts of epochs so that the model can create more accurate predictions. Therefore, my final results, at least for the RGB images, are less accurate than the finding of Krafka et al in [23] because their model was trained on more epochs. Due to computational time, I had to change the number of epochs to 5. Even with decreasing the number of epochs to 5, each model took 3 days to run. A larger epoch value, such as 25, could have increased this run time to at least 15 days.

One last limitation of my study is that I had to use a preexisting deep learning algorithm to create my gender and race predictions. There is a possibility that these demographic predictions are not entirely correct. Additionally, these predictions only grouped people into two genders, male and female, and seven races: White, Black, Latino, Middle Eastern, Indian, East Asian, and Southeast Asian. Therefore, these results cannot be generalized for those that identify as other genders or another race.

7.4 Next Steps

First, my study has only examined data that was created on iPhone and iPad models that were created before 2016. Therefore, future work should involve developing a new dataset with updated technology. Additionally, this new dataset should include a more balanced group of participants for the race variable to determine if the bias of the unequal dataset was a key contributor to the unfair race predictions. Many deep learning algorithms in the past have been found to be unfair to both different groups of races and genders. So, it is important to discover fair algorithms that produce similar results for everyone so that a specific group does not feel the harmful effects of that algorithm. Additionally, this future algorithm should also use data with race and gender demographics so that this information does not have to be predicted using a preexisting demographic prediction model. Then, each of the image processing models should be trained and tested on a larger epoch value so that the model can create more accurate predictions and hopefully fair algorithms. Also, I had intentionally planned on testing this model on more image processing methods including Saravanan grayscale, Marr-Hildreth edge detection, and basic adaptive and Singh binarization. The Saravanan grayscale method uses a more complex formula to create the grayscale image, which could possibly be easier for the model to predict. Marr-Hildreth edge detection uses the Laplacian and 2D Gaussian functions to determine if there is an edge. The Laplacian is also used by the Canny edge detection method, so I would expect that it results in similar prediction accuracy and fairness. Finally, the basic adaptive and Singh binarization methods use a local threshold to determine if a given pixel in the image is black or white. It could be possible that this difference in determining the threshold could lead to more accurate predictions. More information on these image processing methods can be found in the Appendix. Future studies should also investigate if these methods produce fair results.

CHAPTER 8

Conclusion

I have utilized pre-existing code from GazeCapture to utilize their supervised deep-learning algorithm on 8 different image processing methods and tracked the algorithm fairness of each of these methods. I have achieved both of my objectives. First, using convolutional neural networks, I created eye gaze location prediction of the x, y location on an iPhone or iPad that a person is fixated on. For this, prediction is the average Euclidean distance from the actual gaze location to the predicted gaze location. Next, I created algorithm fairness metrics for race and gender variables to determine if I can deem the algorithm and the image processing methods of RGB, grayscale, binarization, and edge detection to be fair.

I have examined that the predictions from the RGB model are the most accurate in the study with an average prediction error of 2.7172 cm. Additionally, I found that the Canny edge detection model is fair between the male and female genders in predicting a gaze location with a probability of obtaining equivalent results 94% of the time. This model's average error is 3.0457 cm, which is significantly less accurate than the RGB error. I also found that the intensity grayscale method had equivalent results for 9% of the time. Next, I did not find any evidence for a racially fair model when using any of the different types of image processing. Interestingly, even with White participants making up the majority of the dataset, their predictions were the least accurate. While the Middle Eastern and Black participants produced the most accurate average prediction. These trends are like those found by Givens et al. in [8].

While these results are good for non-White participants, the goal of the model is to be fair for all races. Therefore, this suggests that there is still much work needed to achieve a racially fair result. Some future work could include creating a dataset that has an equal number of participants for each gender and race group. Furthermore, when this new dataset is created race and gender demographic information should be collected so that the racial and gender metrics can be calculated more easily. Finally, the model used on this new dataset should incorporate a more appropriate number of epochs so that the model can create more accurate predictions.

CHAPTER 9

Appendix

9.1 Github

The code and data files used in my study can be found at `https://github.com/ameyer23-m/Algorithm-Fairness-and-a-Deep-Learning-Approach-to-Eye-Gaze`

9.2 Other image processing techniques

9.2.1 Saravanan Grayscale

Saravanan's grayscale method first calculates the luminance and chrominance values of the images and then approximates the RGB components. Then Saravanan's method calculates the average of the four values that are shown in the equation below and are denoted as R4, G4, B4, and UV. The algorithm is written as:

$$Y = (0.299xR) + (0.587xG) + (0.114xB)$$

$$U = (B - Y)x0.565$$

$$V = (R - Y)x0.713$$

$$UV = U + V$$

$$R1 = R * 0.299$$

$$R2 = R * 0.587$$

$$R3 = R * 0.114$$

$$G1 = G * 0.299$$

$$G2 = G * 0.587$$

$$G3 = G * 0.114$$

$$B1 = B * 0.299$$

$$B2 = B * 0.587$$

$$B3 = B * 0.114$$

$$R4 = (R1 + R2 + R3)/3$$

$$G4 = (G1 + G2 + G3)/3$$

$$B4 = (B1 + B2 + B3)/3$$

$$I1 = (R4 + G4 + B4 + UV)/4$$

9.2.2 Locally Adaptive Binary

Locally adaptive thresholding finds a threshold $T(x, y)$ such that:

$$b(x, y) = \begin{cases} 0, & \text{if } I(x, y) \leq T(x, y) \\ 1, & \text{otherwise} \end{cases} \tag{9.1}$$

where $b(x, y)$ is the binarized image and $I(x, y)$ is the intensity of the pixel at a location (x, y) of the image I. In the basic adaptive equation, the threshold $T(x, y)$ is a weighted sum of the neighborhood of (x, y) minus a constant C [33]. Each pixel is then weighted and determined by the Gaussian function, which gives more significant weight to pixels that are closer to the center of the filter. This results in a smoothing of the image. This makes the image look more grayscale because the black-and-white consistency within an area of the image changes so that one part has more black pixels than other areas. The algorithm uses a convolutional filter to the image using the Gaussian weights. The resulting values are then used to calculate a threshold for the neighborhood.

9.2.3 Singh Binary

In Singh adaptive threshold, Singh calculates the local threshold, $T(x, y)$, computing the local mean, $m(x, y)$, and mean deviation, $\partial(x, y)$ such that:

$$T(x, y) = m(x, y)\left[1 + k\left(\frac{\partial(x, y)}{1 - \partial(x, y)} - 1\right)\right] \tag{9.2}$$

where $\partial(x, y) = I(x, y) - m(x, y)$ is the local mean deviation and k is a bias that controls the level of adaptation varying threshold value in a range from 0 to 1 [44].

9.2.4 Marr-Hildreth Edge Detection

Marr-Hildreth edge detection uses a differential operator to be able to compute the first and second derivatives at every point in the Gaussian smoothed image [30]. Then, it uses the Laplacian, which results from convolving a 3×3 kernel over each patch of the image to find points of zero-crossings in the second derivative, which identifies edges. The Marr-Hildreth differs by identifying changes in the horizontal, vertical, and diagonal directions [37]. These zero crossings occur when the intensity of the pixel change over a wide range of scales. The Marr-Hildreth uses the same Laplacian and 2D Gaussian functions as was used in the Canny edge detection. They find the zero values using the equation:

$$\nabla^2 G(x, y) * I(x, y) \tag{9.3}$$

where $G(x, y)$ is the 2D Gaussian function, ∇^2 is the Laplacian, and $*$ is the convolution operator, such that:

$$\nabla^2 G = \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} = \Big(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\Big) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{9.4}$$

9.3 Addition Fairness Values

The standard deviations (SD) and sample sizes (SS) are shown in Table 9.1 for the gender fairness metrics. Then, Table 9.2 shows the SD, and Table 9.3 shows the SS for the racial metrics.

Table 9.1: Gender fairness t-test data

Image Processing	Method	Male SD	Female SD	Male SS	Female SS
Color	RGB	1.0601	1.2607	85,869	93,600
Grayscale	Intensity	1.0473	1.3284	85,869	93,600
	Luminance	1.0597	1.3195	85,869	93,600
	Luster	1.0633	1.2836	85,869	93,600
	Value	1.0659	1.2736	85,869	93,600
Binary	Otsu	1.3220	1.5797	85,869	93,600
Edge Detection	Canny	1.1106	1.4313	85,869	93,600
	Sobel	1.2859	1.5425	85,869	93,600

Table 9.2: Racial fairness standard deviations data

Image Processing	Method	White	Black	Latino	Middle Eastern	Indian	East Asian	Southeast Asian
Color	RGB	1.2433	0.7420	0.9468	0.6206	0.5779	1.2474	0.9747
Grayscale	Intensity	1.3083	0.7205	0.9230	0.5547	0.5882	1.2261	0.9594
	Luminance	1.3101	0.7931	0.9103	0.6277	0.5244	1.2171	0.8769
	Luster	1.2844	0.6594	0.8852	0.5717	0.4140	1.2221	0.9718
	Value	1.2949	0.7175	0.9216	0.6203	0.3682	1.1323	0.9829
Binary	Otsu	1.5678	0.9192	1.0625	0.5729	0.5861	1.5006	1.6119
Edge Detection	Canny	1.4241	0.7873	0.9532	0.5370	0.6301	1.2575	0.8998
	Sobel	1.5510	0.8153	1.2199	0.6921	0.5913	1.3555	1.1860

Table 9.3: Racial fairness sample sizes data

	White	Black	Latino	Middle Eastern	Indian	East Asian	Southeast Asian
Sample Size	102,700	10,200	16,796	10,900	1,200	30,600	7,100

9.4 Dependencies and Packages

Here is the entire list of dependencies and packages needed to run the code.

- ArgParse: 1.1
- CUDA: 12.0
- NumPy: 1.24.1
- NVIDA GPU Driver: 528.02
- OpenCV: 4.7.0
- PIL: 9.4.0
- Python: 3.10.10
- PyTorch: 1.13.1 + cu117
- Re: 2.2.1
- SciPy: 1.10.0
- Torchvision: 0.14.1

References

[1] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 679–698. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1986.4767851 (page 66).

[2] Jacqueline G. Cavazos et al. "Accuracy comparison across face recognition algorithms: Where are we on measuring race bias?" In: *IEEE transactions on biometrics, behavior, and identity science* 3.1 (Jan. 2021), pp. 101–111. ISSN: 2637-6407. DOI: 10.1109/TBIOM.2020.3027269. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7879975/ (visited on 01/15/2023) (page 15).

[3] Benedetta Cesqui et al. "Gaze Behavior in One-Handed Catching and Its Relation with Interceptive Performance: What the Eyes Can't Tell". In: *PLoS ONE* 10.3 (Mar. 2015). Publisher: Public Library of Science, pp. 1–39. ISSN: 19326203. DOI: 10.1371/journal.pone.0119445. URL: https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=101837515&site=ehost-live (visited on 11/24/2022) (page 7).

[4] Siyu Chen et al. "Innovation of aggregate angularity characterization using gradient approach based upon the traditional and modified Sobel operation". en. In: *Construction and Building Materials* 120 (Sept. 2016), pp. 442–449. ISSN: 0950-0618. DOI: 10.1016/j.conbuildmat.2016.05.120. URL: https://www.

sciencedirect.com/science/article/pii/S095006181630856X (visited on 12/21/2022) (page 63).

[5] Tal Feldman and Ashley Peake. *End-To-End Bias Mitigation: Removing Gender Bias in Deep Learning*. arXiv:2104.02532 [cs]. June 2021. URL: http://arxiv.org/abs/2104.02532 (visited on 01/16/2023) (page 16).

[6] Tobias Fischer, Hyung Jin Chang, and Yiannis Demiris. "RT-GENE: Real-Time Eye Gaze Estimation in Natural Environments". en. In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 339–357. ISBN: 978-3-030-01249-6. DOI: 10.1007/978-3-030-01249-6_21 (page 8).

[7] Laura Florea et al. "Can Your Eyes Tell Me How You Think? A Gaze Directed Estimation of the Mental Activity". In: Jan. 2013, pp. 60.1–60.11. ISBN: 978-1-901725-49-0. DOI: 10.5244/C.27.60 (pages 10–11).

[8] G. Givens et al. "How features of the human face affect recognition: a statistical comparison of three face recognition algorithms". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2. ISSN: 1063-6919. June 2004, pp. II–II. DOI: 10.1109/CVPR.2004.1315189 (pages 15, 105, 108, 111).

[9] Tobii Dynavox Global. *Success Stories Becky*. en. URL: https://www.tobiidynavox.com/pages/cerebral-palsy-success-story-becky (visited on 11/25/2022) (page 2).

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (pages 35, 39–40).

[11] Kaiming He et al. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. Dec. 2015. URL: http://arxiv.org/abs/1512.03385 (visited on 02/09/2023) (page 71).

[12] Barbara Illowsky and Susan Dean. *Statistics*. OpenStax, Mar. 2020. URL: https://openstax.org/books/statistics/pages/1-introduction (pages 21–22).

[13] Aine Ito and Pia Knoeferle. "Analysing data from the psycholinguistic visual-world paradigm: Comparison of different analysis methods". en. In: *Behavior Research Methods* (Nov. 2022). ISSN: 1554-3528. DOI: 10.3758/s13428-022-01969-3. URL: https://doi.org/10.3758/s13428-022-01969-3 (visited on 11/25/2022) (page 7).

[14] Gareth James et al. *An introduction to statistical learning : with applications in R*. New York : Springer, [2013] ©2013, 2013. URL: https://search.library.wisc.edu/catalog/9910207152902121 (pages 29, 31, 43–44).

[15] Christopher Kanan and Garrison W. Cottrell. "Color-to-Grayscale: Does the Method Matter in Image Recognition?" In: *PLoS ONE* 7.1 (Jan. 2012), e29740. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0029740. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3254613/ (visited on 12/16/2022) (page 55).

[16] Anuradha Kar and Peter Corcoran. "A Review and Analysis of Eye-Gaze Estimation Systems, Algorithms and Performance Evaluation Methods in Consumer Platforms". In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 16495–16519. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2735633 (page 7).

[17] Kimmo Kärkkäinen and Jungseock Joo. "FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age". en. In: (Aug. 2019). DOI: 10.48550/arXiv.1908.04913. URL: https://arxiv.org/abs/1908.04913v1 (visited on 02/09/2023) (page 70).

[18] Petr Kellnhofer et al. "Gaze360: Physically Unconstrained Gaze Estimation in the Wild". In: (Jan. 2019) (pages 9, 11).

[19] Joohwan Kim et al. "NVGaze: An Anatomically-Informed Dataset for Low-Latency, Near-Eye Gaze Estimation". en. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, May 2019, pp. 1–12. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300780. URL: https://dl.acm.org/doi/10.1145/3290605.3300780 (visited on 12/01/2022) (pages 8, 10).

[20] Ahmad F. Klaib et al. "Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and Internet of Things technologies". en. In: *Expert Systems with Applications* 166 (Mar. 2021), p. 114037. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2020.114037. URL: https://www.sciencedirect.com/science/article/pii/S0957417420308071 (visited on 11/25/2022) (pages 1, 7).

[21] Alina Köchling and Marius Claus Wehner. "Discriminated by an algorithm: a systematic review of discrimination and fairness by algorithmic decision-making in the context of HR recruitment and HR development". en. In: *Business Research* 13.3 (Nov. 2020), pp. 795–848. ISSN: 2198-2627. DOI: 10.1007/s40685-020-00134-w. URL: https://doi.org/10.1007/s40685-020-00134-w (visited on 01/12/2023).

[22] Kristin M. Kostick-Quenet et al. "Mitigating Racial Bias in Machine Learning". eng. In: *The Journal of Law, Medicine & Ethics: A Journal of the American Society of Law, Medicine & Ethics* 50.1 (2022), pp. 92–100. ISSN: 1748-720X. DOI: 10.1017/jme.2022.13 (page 2).

[23] Kyle Krafka et al. "Eye Tracking for Everyone". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (pages 9–10, 68–69, 76, 85, 91, 109).

[24] K. S. Krishnapriya et al. *Characterizing the Variability in Face Recognition Accuracy Relative to Race*. arXiv:1904.07325 [cs]. May 2019. DOI: 10.48550/arXiv.1904.

07325. URL: http://arxiv.org/abs/1904.07325 (visited on 01/15/2023) (page 15).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (page 49).

[26] Omkar Kulkarni et al. *Accuracy and Fairness in Pupil Detection Algorithm*. Pages: 24. Nov. 2021. DOI: 10.1109/BigMM52142.2021.00011 (pages 2, 14–15, 18).

[27] Tarun Kumar and Karun Verma. "A Theory Based on Conversion of RGB image to Gray image". en. In: *International Journal of Computer Applications* 7.2 (Sept. 2010), pp. 5–12. ISSN: 09758887. DOI: 10.5120/1140-1493. URL: http://www.ijcaonline.org/volume7/number2/pxc3871493.pdf (visited on 11/16/2022) (page 52).

[28] Meng-Lung Lai et al. "A review of using eye-tracking technology in exploring learning from 2000 to 2012". en. In: *Educational Research Review* 10 (Dec. 2013), pp. 90–115. ISSN: 1747-938X. DOI: 10.1016/j.edurev.2013.10.001. URL: https://www.sciencedirect.com/science/article/pii/S1747938X13000316 (visited on 11/25/2022).

[29] C. Lopez-Molina et al. "Multiscale edge detection based on Gaussian smoothing and edge tracking". en. In: *Knowledge-Based Systems* 44 (May 2013), pp. 101–111. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2013.01.026. URL: https://www.sciencedirect.com/science/article/pii/S0950705113000464 (visited on 12/21/2022) (pages 60–61).

[30] David Marr and Ellen Hildreth. "Theory of edge detection". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207.1167 (1980), pp. 187–217 (page 116).

[31] Shree K. Nayar. *Canny Edge Detector | Edge Detection*. Mar. 2021. URL: https://www.youtube.com/watch?v=hUC1uoigH6s (visited on 12/21/2022) (pages 65–66).

[32] Michael A. Nielsen. "Neural Networks and Deep Learning". en. In: (2015). Publisher: Determination Press. URL: http://neuralnetworksanddeeplearning.com (visited on 11/16/2022) (pages 24, 27).

[33] *OpenCV: Miscellaneous Image Transformations*. URL: https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#ggaa42a3e6ef26247da787bf34030ed772caf262a01e7a (visited on 12/20/2022) (page 115).

[34] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (Jan. 1979). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, pp. 62–66. ISSN: 2168-2909. DOI: 10.1109/TSMC.1979.4310076 (page 58).

[35] Trishan Panch, Heather Mattie, and Rifat Atun. "Artificial intelligence and algorithmic bias: implications for health systems". In: *Journal of Global Health* 9.2 (), p. 020318. ISSN: 2047-2978. DOI: 10.7189/jogh.09.020318. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6875681/ (visited on 01/15/2023).

[36] D. Panchuk and J. N. Vickers. "Gaze behaviors of goaltenders under spatial–temporal constraints". en. In: *Human Movement Science* 25.6 (Dec. 2006), pp. 733–752. ISSN: 0167-9457. DOI: 10.1016/j.humov.2006.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0167945706000583 (visited on 11/24/2022) (page 7).

[37] Mahesh R. Panicker. *Edge Detection Using Marr Hildreth Algorithm*. Nov. 2021. URL: https://www.youtube.com/watch?v=KdQ-wJlViR4 (visited on 12/22/2022) (page 116).

[38] Primesh Pathirana et al. "Eye gaze estimation: A survey on deep learning-based approaches". en. In: *Expert Systems with Applications* 199 (Aug. 2022), p. 116894. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2022.116894. URL: https://www.sciencedirect.com/science/article/pii/S0957417422003347 (visited on 11/25/2022) (page 7).

[39] Marcelo O. R. Prates, Pedro H. C. Avelar, and Luis Lamb. *Assessing Gender Bias in Machine Translation – A Case Study with Google Translate*. arXiv:1809.02208 [cs]. Mar. 2019. URL: http://arxiv.org/abs/1809.02208 (visited on 01/16/2023) (pages 2, 14, 18).

[40] Adria Recasens et al. "Where are they looking?" In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015. URL: https://papers.nips.cc/paper/2015/hash/ec8956637a99787bd197eacd77acce5e-Abstract.html (visited on 12/01/2022) (page 10).

[41] Nerisanu Remus et al. "CEREBRAL PALSY AND EYE-GAZE TECHNOLOGY. INTERACTION, PERSPECTIVE AND USABILITY. A REVIEW". In: (Nov. 2017) (page 1).

[42] Chandran Saravanan. "Color Image to Grayscale Image Conversion". In: *2010 Second International Conference on Computer Engineering and Applications* 2 (2010), pp. 196–199 (pages 53, 56).

[43] Terence Shin. *Real-life Examples of Discriminating Artificial Intelligence*. en. June 2020. URL: https://towardsdatascience.com/real-life-examples-of-discriminating-artificial-intelligence-cae395a90070 (visited on 01/12/2023) (pages 2, 14).

[44] T. Romen Singh et al. “A New Local Adaptive Thresholding Technique in Binarization”. In: (Jan. 2012). arXiv:1201.5227 [cs]. DOI: 10.48550/arXiv.1201.5227. URL: http://arxiv.org/abs/1201.5227 (visited on 12/18/2022) (page 115).

[45] J. N. Stember et al. “Eye Tracking for Deep Learning Segmentation Using Convolutional Neural Networks”. In: *Journal of Digital Imaging* 32.4 (Aug. 2019), pp. 597–604. ISSN: 0897-1889. DOI: 10.1007/s10278-019-00220-4. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6646645/ (visited on 11/25/2022) (page 7).

[46] Maisarah Mohd Sufian et al. “COVID-19 Classification through Deep Learning Models with Three-Channel Grayscale CT Images”. In: *Big Data and Cognitive Computing* 7.1 (2023). ISSN: 2504-2289. DOI: 10.3390/bdcc7010036. URL: https://www.mdpi.com/2504-2289/7/1/36 (page 86).

[47] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. “Learning-by-Synthesis for Appearance-Based 3D Gaze Estimation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2014, pp. 1821–1828. DOI: 10.1109/CVPR.2014.235 (page 8).

[48] Wenlong Sun, Olfa Nasraoui, and Patrick Shafto. “Evolution and impact of bias in human and machine learning algorithm interaction”. en. In: *PLOS ONE* 15.8 (Aug. 2020). Publisher: Public Library of Science, e0235502. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0235502. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0235502 (visited on 01/12/2023).

[49] Petroc Taylor. *Smartphone subscriptions worldwide 2027*. en. Jan. 2023. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (visited on 01/23/2023) (page 1).

[50] Aston Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021) (pages 43, 45–46).

[51] Xucong Zhang et al. "Appearance-based gaze estimation in the wild". en. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 4511–4520. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7299081. URL: http://ieeexplore.ieee.org/document/7299081/ (visited on 12/01/2022) (page 10).

[52] Xucong Zhang et al. "ETH-XGaze: A Large Scale Dataset for Gaze Estimation Under Extreme Head Pose and Gaze Variation". en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Vol. 12350. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 365–381. ISBN: 978-3-030-58557-0 978-3-030-58558-7. DOI: 10.1007/978-3-030-58558-7_22. URL: https://link.springer.com/10.1007/978-3-030-58558-7_22 (visited on 09/16/2022) (pages 8, 10).

[53] Xuebai Zhang et al. "Eye Tracking Based Control System for Natural Human-Computer Interaction". In: *Computational Intelligence & Neuroscience* (Dec. 2017). Publisher: Hindawi Limited, pp. 1–9. ISSN: 16875265. DOI: 10.1155/2017/5739301. URL: https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=126841209&site=ehost-live (visited on 11/25/2022) (page 7).