Convolutional Neural Networks for Eye Tracking Algorithm

Jonathan Griffin
Stanford University
jgriffi2@stanford.edu

Andrea Ramirez
Stanford University
aramire9@stanford.edu

Abstract

Eye tracking is an integral part of many VR and AR systems. Common eye tracking algorithms use infrared LEDs shown onto the eye, then compute the relative distance of that light to the pupil of the eye as a means to determine eye movement. We suggest that using a fast classification method, such as CNNs, on a relatively dense dataset and using interpolation methods could give similar speed and accuracy.

1. Introduction

Virtual Reality and Augmented Reality systems can alter the way we perceive our surroundings. These systems use a variety of techniques to create a comfortable new reality for the users. One of these techniques is eye tracking, a process where the eye gaze is measured to deduce where a person is looking and what element of their environment they are paying attention to. Today, there are a number of algorithms that are commonly used to implement eye tracking in a VR or AR system. However, we believe that there is room for exploration into other eye tracking techniques.

1.1. Motivation

In order to explore the possible alternatives to existing eye tracking algorithms, we decided to focus on building our own algorithm. We were interested in using neural networks to build our new algorithm. Neural networks are a popular method in different machine learning applications, but have not been popular in the eye tracking field. Therefore, we wanted to look into the application of neural networks in eye tracking to begin understanding how we could use them to build a reliable model for eye tracking. This paper explores the development and testing of our neural network based eye tracking model.

2. Related Work

Work on eye tracking algorithms dates back to the 1960s [7]. In this section, we aim to learn about some of these algorithms.

2.1. Infrared Eye Tracking

Infrared Eye Tracking is a very popular eye tracking algorithm. Eye trackers that use this technique have an infrared camera placed next to a display. The infrared light is aimed at the eye. The light does not disturb the user. This light creates reflections in the eye. Image processing software sees these reflections and keeps track of the pupil. The position of the reflections relative to the pupil is used to determine the point-of-regard, the position that the user is predicted to be looking at [9].

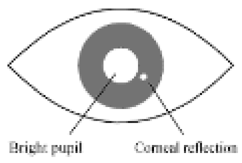

Figure 1. Pupil and reflection from infrared light aimed at eye for tracking [9].

2.2. Electro-oculography

Electro-oculography measures the electric potential between the front and back of the eye. An electric field exists near the cornea and retina. As the eyeballs rotate, the orientation of this electric field changes. Electrodes are placed at different points on the skin close to the eye. These electrodes measure the changes in the electric field due to eye movement. These measurements are then amplified and analyzed to deduce the eye position [5].

2.3. CNNs

Convolutional Neural Networks, or CNNs, are a type of neural network that have different convolutional layers used to train and build hidden levels of neurons that connect the input with output classes for classification. These neurons activate depending on the input stimuli. CNNs are very popular machine learning tools for image analysis. They are

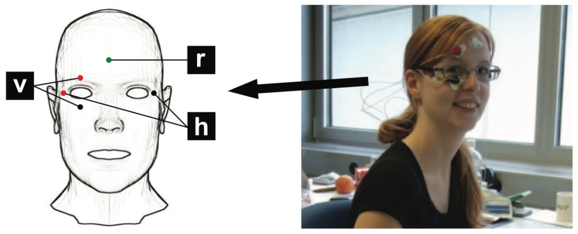

Figure 2. Example of electrode position for electro-oculography [1].

useful models with large learning capacity and are able to handle the complexity of tasks involving problems that are too large to be fully specified in any data set [8].

Using CNNs for eye tracking is a newer algorithm that is now being researched [10]. Because of how little has been done in this area, our project focuses on this eye tracking algorithm in order to add knowledge to the development of CNNs for eye tracking.

3. Methods

There were many processes to go through in order to test our eye tracking algorithm. Our dataset was not completely functional for our methods, so we had to perform a few preprocessing algorithms on it. These algorithms will be discussed below. We will also discuss what type of architecture we used to train our preprocessed dataset and why we chose that architecture. We used various optimizers in our training algorithm to develop the fastest, most accurate model. We tested how normalization affects the dataset and the model reacts when it is trained. Finally, we interpolated the results we got from the model to determine a more accurate result.

3.1. Dataset

We did not create our own dataset due to time constraints. Therefore, we used a dataset that provided the most accurate and useful information for our needs. We used the dataset from [12] because it was a large enough dataset that contained varying eye shapes, sizes, eye colors, and light conditions. The dataset also had ground-truth directional vectors that pointed in the direction of where the eye was looking. The way the authors of [12] obtained their data was by performing dense 3D head scans, retopologizing the scans for animation, adding displacement maps for detail, manually adding iris and eyelid detail, and rendering the image. The process is shown below.

The ground-truth directional vectors are length three vectors that represent the x, y, and z angles of rotation in degrees. The way the authors of [12] obtained this information was by placing the camera of their rendering software in the location of the pupil of the eye and took the measurements of the camera.

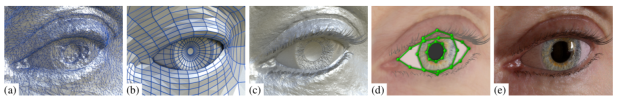

Figure 3. (a) 3D head scans (b) retopology (c) skin details added through displacement maps (d) iris and eyelid details added manually (e) rendering [12].

Figure 4. Example image of an eye in the dataset [12].

The dataset has 11382 samples of images in varying conditions with varying attributes. The dataset also has 11382 different directional vectors that the eyes would be looking in. This is an issue because for convolutional neural networks to work efficiently, there needs to be some dimensionality reduction. This problem is addressed and explained in the Preprocessing section below.

3.2. Preprocessing

Due to the issue described above in the Dataset section, we needed to reduce the dimensionality of the dataset. The way we did this was by computing the distance of each directional vector with all other directional vectors and summing them together. In Equation (1), $dist_i$ is the summation of the distances from vector v_i to all other vectors.

$$dist_i = \sum_{i \neq j} \|v_i - v_j\| \tag{1}$$

After computing the total distances for all directional vectors, we took the directional vectors that have the largest total distances to be the most unique vectors and kept them and got rid of the rest of the directional vectors to reduce the dimensions. We kept the C most unique directional vectors, where $C = 11382 * 0.005 = 56$. Therefore, we were able to reduce the dimensions of our dataset from 11382 samples to 56 classes.

Then, we had to relabel the samples to have one of the 56 classes instead of the original directional vector it was. We did this by using a simple L2 norm function on the original directional vector with all of the 56 unique vectors.

3.3. Architecture

We developed the convolutional architecture similar to that of [2]. [2] uses an architecture that first detects the face of a person, and then localizes to the eyes. This is a simple deformable parts model, but we do not need to implement this first step because we already have images of eyes. The model consists of three convolutional layers with a single rectified linear unit (ReLU) and max-pooling layer in between. The input to the model is an image from the dataset, of shape 120 x 80 x 3. The first convolutional layer consists of a 32 filters, each with shape 7 x 7, with a stride of 1 and padding of 3. The padding size and stride length were chosen so that the height and width of the input would remain the same as the output. The height and width of the output of the convolutional layer is defined below, where $H^{'}$ and $W^{'}$ are the output height and width, H and W are the input height and width, pad is the padding size, stride is the stride length, and F is the filter size.

$$H^{'} = 1 + \frac{H + 2pad - F}{stride} \tag{2}$$

$$W^{'} = 1 + \frac{W + 2pad - F}{stride} \tag{3}$$

Following the convolutional layer is the ReLU layer, which is defined as:

$$f(x) = \max(0, x) \tag{4}$$

where x is the input the the ReLU layer.

After the ReLU layer, we run a 2 x 2 max-pooling layer, with stride of 2. Max pooling will take the maximum value of a 2 x 2 section of the input layer and keep only that value, thus reducing the size of the input layer in half.

We repeat the same process with 32 filters with shape 5 x 5 and then a layer of 64 filters with shape 3 x 3, with the ReLU and max-pooling layers in between each. Afterwards, we flatten the output layer to be of shape 1 x 150 and perform a fully connected layer to produce an output vector of length 56.

Next, we compute the loss of the model. We use cross entropy loss, which is the same loss function that [2] uses. cross entropy loss is defined as:

$$L(f(x), y) = -y \ln(f(x)) - (1 - y)\ln(1 - f(x)) \tag{5}$$

where x is the output vector of length 56 and y is a label of either 0 or 1, depending on if the classification if incorrect or correct, respectively.

Finally, we classify the image to a corresponding directional vector by taking the label with the maximum score:

$$class = \arg\max(score) \tag{6}$$

3.4. Optimizers

We tested three different optimizers to see which would give the best results for our model. The three optimizers were adaptive moment estimation (Adam), RMSprop, and stochastic gradient descent (SGD). Each of the optimizers have their own advantages and disadvantages that we will discuss below.

SGD updates the parameters for every training example. It is fast due to these frequent updates, but his causes higher variance in the loss and make it more challenging to to converge due to these fluctuations [13]. To reduce the number of oscillations that occur in SGD, we add an additional variable for momentum, which will accelerate the descent [13]. The equations for SGD that we used are shown below:

$$v = m * v - lr * dx \tag{7}$$

$$x = x + v \tag{8}$$

where v is the velocity, m is the momentum, lr is the learning rate, dx is the gradient of x, and x the the position.

RMSprop is an adaptive learning rate method that uses the root mean square of the gradients to update the weight of our model [3]. It updates the learning rate individually for each parameter and divides the learning rate using an exponentially decaying average of square gradients [3]. RMSprop is actually unpublished but is widely used due to the accuracy it gives in most models. The equations for RMSprop that we used are shown below:

$$cache = \alpha * cache + (1 - \alpha) * dx^2 \tag{9}$$

$$x = x - lr * \frac{dx}{\sqrt{cache} + eps} \tag{10}$$

where α is a smoothing coefficient.

Adam is another adaptive learning rate method, like RMSprop, but incorporates momentum changes for each parameter [6]. It uses both the mean and variance of the gradients to update the parameters of the model. Adam was made to address some of the drawbacks of SGD and RMSprop, so it is by far the more robust optimizer. The equations for Adam that we used are shown below:

$$m = beta_1 * m + (1 - beta_1) * dx \tag{11}$$

$$m_t = \frac{m}{1 - beta_1^t} \tag{12}$$

$$v = beta_2 * v + (1 - beta_2) * (dx^2) \tag{13}$$

$$v_t = \frac{v}{1 - beta_2^t} \tag{14}$$

$$x = x - lr * \frac{m_t}{\sqrt{v_t} + eps} \tag{15}$$

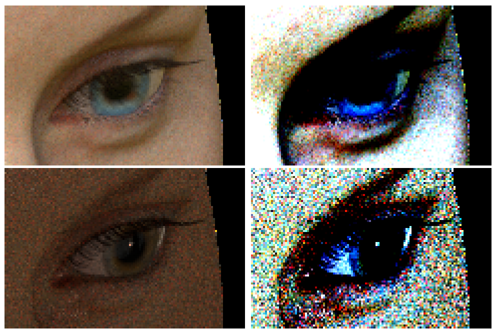

Figure 5. Top Left, Bottom Left: unnormalized images. Top Right, Bottom Right: normalized images.

where m is the smoothed version of the gradient, t is an iterator that allows for bias correction, and $beta_1$ and $beta_2$ are coefficients for averaging the mean and variance.

For testing, we used standard values for learning rate, alpha, betas, and momentum of 0.001, 0.9, 0.5 and 0.999, and 0.9, respectively.

3.5. Normalization

We also tested how normalization affects the run time and the accuracy of our classify. To normalize our dataset, we took each image and subtracted its mean and divided by its standard deviation. The equation is defined as:

$$x^{'} = \frac{x-\mu}{\sigma} \tag{16}$$

where μ is the mean and σ is the standard deviation.

The affect of normalization on an image is shown in Figure 5.

Here, normalization is used to correct for various lighting conditions and to keep all images within the same range of values, with a mean of 0 and a standard deviation of 1.

3.6. Interpolation

The last step in our algorithm is to interpolate between the 56 most unique directional vectors to get the most accurate representation of the ground-truth vector. To do this, we use the probability distribution that softmax loss uses.

$$f(score_i) = \frac{e^{score_i}}{\sum e^{score_j}} \tag{17}$$

This can cause a problem though if the divisor in Equation 17 is sufficiently small. Therefore, we subtract the maximum value from the scores to ensure that the new maximum value is 0, meaning the divisor will at least be 1. Now, all the sum of all $f(score_i)$ from Equation 17 will be 1 and each individual $f(score_i)$ will be between 0 and 1. Call these values p.

From here, we multiply p by the 56 most unique directional vectors, and sum them together to make a representation of the most likely directional vector to the ground-truth directional vector. Lastly, we must normalize this vector so that the length is 1.

$$f(p,y) = \frac{\sum p*y}{\|\sum p*y\|} \tag{18}$$

4. Evaluation

We evaluated our algorithm based on the accuracy in which we were able to classify each image into the 56 unique directional vectors and then interpolate between those directional vectors to obtain the ground-truth direction.

4.1. Qualitative

There were no qualitative measurements to perform evaluation on.

4.2. Quantitative

Our quantitative measurements were based on the accuracy of classification and accuracy of interpolating between the classifications.

4.2.1 Classification

We tested our model with three different optimizers, with normalization and without normalization. Without normalization, all of our models performed below chance, with Adam producing the best result of accuracy 35%. The losses for the models without normalization did not change much at all, which suggest that very little learning occurred in the model. This is probably why the accuracy suffered. The results for the models without normalization is shown in Figure 6.

With normalization, Adam and SGD improved dramatically, as can be seen in Figure 7. Adam increased from 35% to 70% and SGD improved from 0% to 60%. The reason that Adam and SGD did improve is most likely due to how normalization places all images in the same range of values with 0 mean and unit standard deviation. The loss for Adam suggests that there was an improvement because the loss slightly decreases the longer the model is trained. SGDs loss fluctuated a lot, but that is expected due to its rapid updating as mentioned before. RMSprop, on the other hand, did not improve at all. The accuracy remains the same, but the loss behaves differently. It has a cycle which suggests that there was a local minimum that RMSprop could not get through, most likely due to the lack of momentum in the equation.

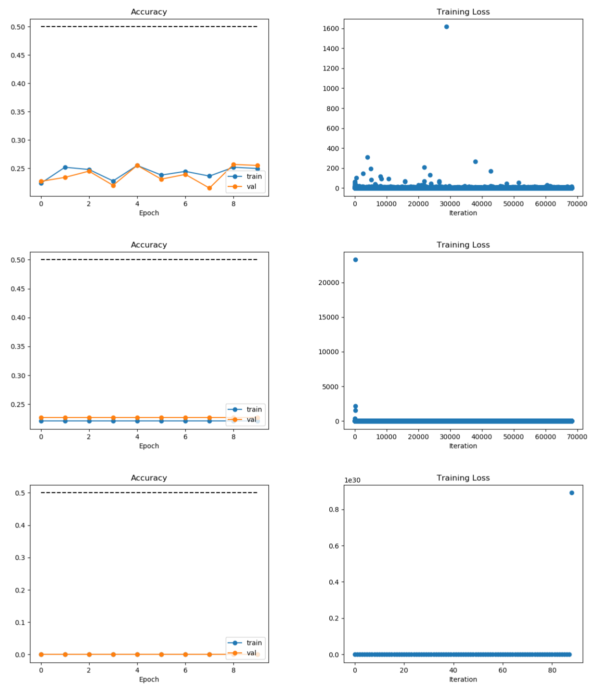

Figure 6. Top: Accuracy and Loss for Adam. Middle: Accuracy and Loss for RMSProp. Bottom: Accuracy and Loss for SGD. All models did not use normalization.

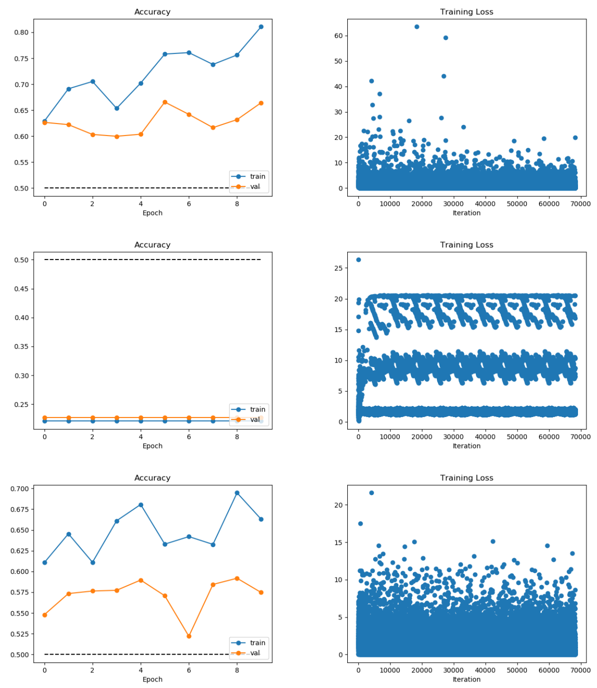

Figure 7. Top: Accuracy and Loss for Adam. Middle: Accuracy and Loss for RMSProp. Bottom: Accuracy and Loss for SGD. All models used normalization.

4.2.2 Interpolation

Our interpolating algorithm yielded the worst results. For all models, both with and without normalization, produced an accuracy of 0%. There is likely something wrong with how we have decided to perform the interpolation because the results of the classification step is accurate. The way we measured accuracy was based on if the interpolated direction we produced was within 10^{-5} of the ground-truth direction. With this measure of error, we had 0% accuracy. If we look at the average error though, it was 0.59 degrees. This is not a large error in reference to the human field of view. a 0.59 degree error is only 0.27% of the human horizontal field of view and 0.39% of the human vertical field of view. With this standard, our method seems fairly accurate, but not as accurate as we would like. Our algorithm is also fast. It takes, on average 0.005 seconds, or 5 milliseconds. This is significantly faster than the average humans visual reaction time of approximately 400 milliseconds [11].

5. Comparison to Previous Work

There is a wide variety of eye tracking algorithms, all of which focus on different aspects of the eye. Infrared eye tracking focuses on corneal reflections to define the eye position, while electro-oculography focuses on electric potential and eye rotations. Both of these methods have been shown to have a good correspondence between each other [4]. However, there has not been a comparison between these two methods and the use of CNNs. Perhaps this is due to the relatively recent saliency of CNNs which were not as used before when compared to infrared eye tracking and electro-oculography. CNNs are not only a more recent algorithm that is still being researched, but it is also an algorithm that, unlike the other two methods, employs machine learning techniques. However, because our CNN method is new, it still has to be thoroughly tested and optimized in order to provide reliable eye tracking that is comparable to the infrared eye tracking and the electro-oculography.

Our CNN eye tracking model is similar to other CNN eye tracking methods proposed by others [10]. Our convolutional architecture was inspired by the architecture proposed in [2]. Unlike the testing done in [2], which was done after training for 1000 epochs, we only trained for 10 epochs due to computational limitations from our available computers. This could be a reason why, despite using similar architecture, the model in [2] reached 92% accuracy compared to our 60% to 70% accuracy. The model in [2] was also using a smaller training set of 450 images compared to our 6829 training images. They also tested on only 50 images while we tested on 2276 images. Their smaller validation set could be another reason why our accuracies were different.

6. Discussion

The results of our classification showed that normalization is a key factor when building an eye tracking model using CNNs. All of our optimizers exhibited very low accuracy when classifying using unnormalized data. Adam had the highest accuracy of the three, but this accuracy was not high enough to have a reliable eye tracking system. Normalized data allowed our model to achieve good accuracies, especially when considering that we only ran training for 10 epochs. We also noticed that the initial accuracy after the first training epoch is significantly higher for normalized data versus unnormalized data, which is probably a factor that defines how high our accuracy could get.

An interesting outcome from our model was seeing how the RMSProp optimizer had virtually zero change between tests of normalized and unnormalized data. This suggests that RMSProp was not a well suited optimizer for this particular set of data, and that it probably converged to a sub-optimal local minimum when updating parameters. Further testing could help understand what would be the optimal parameters and condition to get better accuracy from RMSProp.

SGD and Adam optimizers both reached a very good accuracy with normalized data. SGD had the highest performance boost after switching to normalized data. It was probably doing worse with unnormalized data compared to Adam because SGD suffers from high fluctuations that keep it from converging quickly and from reaching the minimum. Adam addresses these drawbacks which probably explain its better performance, both in the unnormalized and normalized data. Adam is meant to be faster and more robust, in exchange for added complexity, which is why we expected Adam to be the best optimizer.

We can also see the benefits of Adam when evaluating the training losses for the optimizers. SGD and Adam both had similar accuracies, but the training losses were much noisier for SGD. This is again due to the high variance SGD has, which creates many oscillations that are reflected in SGD's training loss. Adam was designed to not suffer from these oscillations, which is reflected in its less variable training loss data.

Despite the good performance that both SGD and Adam showed, we still encountered issues when trying to interpolate data. Ideally, we would be able to interpolate the exact direction each image was looking in after calculating the probabilities of each of our direction classes. However, we had no accuracy from our interpolation data. There are many reasons why we could have had issues with interpolation. One of them is that we do not have the best database available for classifying with CNNs. Since our data set had as many classes as it had samples, we had to reduce the classes we used to build our CNN. An ideal data set would be built such that we do not have to reduce the amount of classes, it would have significantly more samples than classes. Another issue is that we are not familiar with the exact distribution of the data we used. Therefore, we probably made mistakes when trying to describe the probability distribution used to interpolate data.

In order to improve on a CNN based eye tracking algorithm, there are many steps we could take. In the future, it would be better to build a new database that is more dense than our current database. We could also work on analyzing the data distribution to create a proper probability distribution that can be used to interpolate data accurately. Once the testing has improved and we can simulate a more reliable eye tracking model, we could test the model on a real VR or AR system with real people. This could help us understand what should be taken into account when performing eye tracking on people with different visual senses. Varying inter-pupillary distance, nearsightedness and farsightedness are only a few things that could affect how well our eye tracking model works when used to deliver a service to a user.

Our CNN based eye tracking model demonstrates that CNNs are a good tool when developing eye tracking systems for VR and AR. There is a lot of work that needs to be done in this area to be able to take full advantage of the benefits of CNNs.

References

[1] Bulling, Ward, Gellersen, and Gerhard. Eye movement analysis for activity recognition using electrooculography. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[2] A. George and A. Routray. Real-time eye gaze direction classification using convolutional neural network. *IEEE International Conference on Signal Processing and Communication*, 2016.

[3] G. Hinton. Overview of mini-batch gradient descent.

[4] W. Iacono and D. Lykken. Two-year retest stability of eye tracking performance and a comparison of elecro-oculographic and infrared recording techniques: Evidence of eeg in the electro-oculogram. *Psychophysiology*, 1981.

[5] Kaufman, Bandopadhay, and Shaviv. An eye tracking computer user interface. *IEEE Research Properties in Virtual Reality Symposium*, 2002.

[6] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2017.

[7] C. Kleinke. Gaze and eye contact: A research review, 1986.

[8] Krizhevsky, Sutskever, and Hinton. Imagenet classification with deep convolutional neural networks. *International Conference on Neural Information Processing Systems*, 2012.

[9] A. Poole and L. Ball. Eye tracking in hci and usability research, 2006.

[10] V. Sitzmann, A. Serrano, A. Pavel, M. Agrawala, D. Gutierrez, B. Masia, and G. Wetzstein. How do people explore

virtual environments? *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[11] Thorpe, Fize, and Marlot. Speed of processing in human visual system, 1996.

[12] Wood, Baltrusaitis, Zhang, Sugano, Robinson, and Bulling. Rendering of eyes for eye-shape registration and gaze estimation. *IEEE International Conference on Computer Vision*, 2015.

[13] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. *Proceedings of the twenty-first international conference on Machine learning*, 2012.