MobileGaze: An efficient framework for mobile gaze tracking

Avoy Datta
Stanford University
Stanford, CA
avoy.datta@stanford.edu

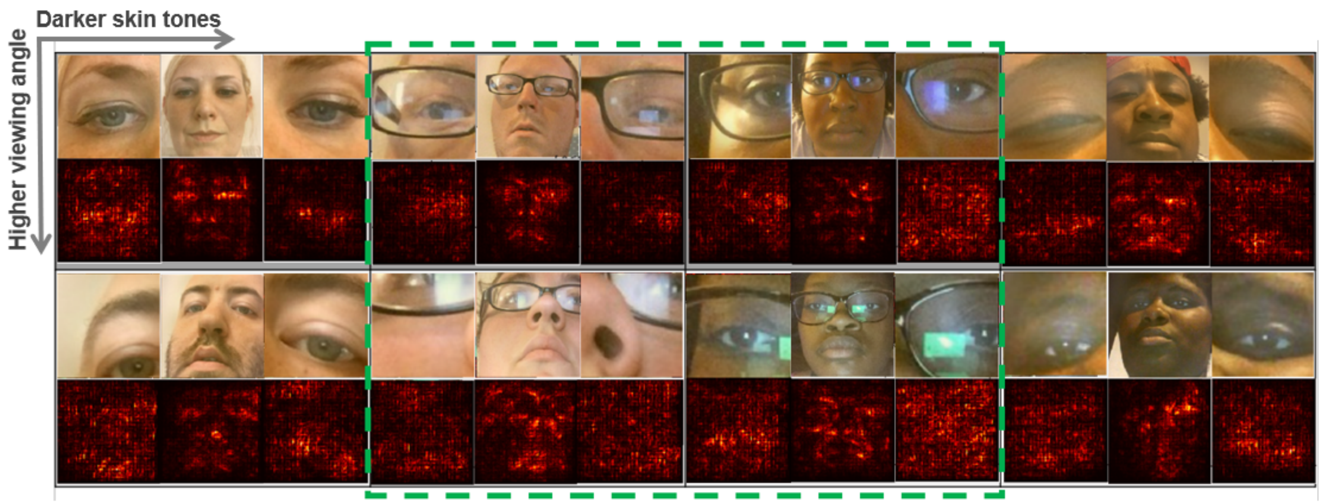


Figure 1: Tracking the Network's Gaze: What a neural network focuses on when making gaze predictions

Abstract

Gaze tracking on mobile devices demands optimization across prediction accuracy, space usage and computational efficiency. To this end, we propose an efficient and compact convolutional neural network, MobileGaze, that is primed towards accurate gaze predictions using a compact set of parameters. On the GazeCapture dataset, MobileGaze achieves a dot error rate of 2.12 cm without ensembling after 5 epochs of training, beating the existing state-of-the-art's 2.46 cm error rate. What's more astonishing is MobileGaze was able to achieve 2.22 cm dot error after just 3 epochs of training, compared to the 25 epochs of training for the state-of-the-art, showing that MobileGaze converges to a satisfactory solution significantly faster than its competitor. Alongside its memory and computational efficiency, this attribute also makes MobileGaze a more suitable model for deployment in conditions that require real-time machine learning.

We also perform experiments tracking the specific parts of inputs MobileGaze focuses on when making predictions. Comparing its saliency maps with those of the iTracker model, we find that the gradient flow in MobileGaze is considerably more conservative, which makes it more robust to noise in the facial crops that form a third of its image inputs. We conclude by establishing the connection between this robustness to noise and the superior performance over iTracker on the gaze prediction task.

1. Introduction

Eye tracking has been crucial to developing attentive user interfaces in both Augmented Reality(AR) and Virtual Reality(VR) systems. A reliable gaze detection system allows the computer to adaptively render visuals on the head mounted display (for VR) or the plethora of display technologies in AR (heads-up displays and mobile monitors, among others) based on the user's gaze. But perhaps where gaze tracking's biggest benefit lies is its ability to provide a hands-free means of human-computer interaction (HCI).

Infrared-based video-oculography[28] lies at the core of state-of-the-art eye-tracking systems used in both HMDs and HUDs today, mapping corneal reflections[8] to gaze vectors to estimate gaze positions on the display. However, one major shortcoming of infrared-based approaches is the necessity of an infrared source and sensor in close proximity to the cornea - infrared-based approaches fail as soon as the user sets down the wearable display. This makes IR-based oculography a poor approach to gaze tracking in **mobile Augmented Reality** applications, an emerging field where the device often only has a webcam available to perform any gaze-tracking task. This has fueled interest in gaze-tracking beyond IR-oculography.

Computer vision has been used as a tool in attempts to solve this problem, with architectures built on convolutional neural networks (CNNs) leading the race for a robust vision algorithm for gaze tracking [22]. We frame this research problem as a simultaneous object-segmentation and point of gaze (PoG)-regression task, and attempt to use a deep CNN to predict the POG locations of a subject on a mobile device placed in front.

In addition to PoG accuracy, we also optimize the space efficiency of our model. Even the best gaze-tracker would be a poor fit for mobile AR if it couldn't be loaded onto memory on its host devices. In this endeavor, we propose MobileGaze, which is built on top of the SqueezeNet[12] architecture. SqueezeNet was groundbreaking in achieving state-of-the-art ImageNet classification accuracies with hundreds of times fewer model parameters than comparable competitors. Space efficiency in this scale will reflect on the breadth of mobile hardware that can load and run this model, vastly improving its scalability as a mass-distributed product.

2. Related Work

2.1. Origins of computer vision in gaze-tracking

The use of computer vision in gaze tracking has been surveyed since before most deep learning methods were introduced. [22] provides a survey of both intrusive (contact lenses and electro-oculography)[6] and non-intrusive image-based methods used to capture eye movement. The authors concluded the artificial neural networks (ANNs)[1] performed the worst at this task, with error margins in predicting the gaze position larger than video-oculography [28]. The ANN used consisted of a 3-layer deep feedforward neural network, with hidden units divided in some of the experiments to disjointly predict the x and y coordinates of gaze on a screen, with inputs to the network being raw pixel values. One of the major drawbacks of a feedforward network is the hidden neurons' lack of ability to learn *spatial context* given a set of pixels in an image. In the field of computer vision, Convolutional Layers, first proposed as the Neocognitron in [5] with backpropagation applied later in [17], are able to capture semantic meaning among groups of neighboring pixels by computing cross-correlations for each group. Deep convolutional networks consist of multiple convolutional layers stacked in series, and have been shown to improve inference-time performance of these networks on vision tasks given a sufficiently large training set.

2.2. Trends in publicly available datasets

To train a model that performs well in real-world conditions, we need data that can accurately and reliably capture the diversity of conditions in which the model will be put to the test. There have traditionally existed a number of publicly available gaze datasets in this field ([29], [21], [26], [31], [11]). Many of the earlier datasets ([29], [21], [26]) do not contain significant variation in head pose or have a coarse gaze point sampling density, while others have limited scale - especially in the number of participants ([31], [11]).

Developed at MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL), GazeCapture[15] tackles some of the shortcomings of its predecessors and currently stands the largest publicly available gaze-tracking dataset of human images. The data was crowdsourced using a custom-designed app (also called GazeCapture) through Amazon Mechanical Turk (AMT), making the data collection scalable to a large population. Crowdsourcing allowed the authors to build a dataset with roughly 30 times as many participants as the next largest dataset, while recording premissions enabled them to make the data public without occluding the faces of subjects. Steps were also taken to minimize distractions during the collection process (improving reliability), with specific instructions provided during the collection process that enhanced variability of data. The creators of the dataset also propose their own CNN-based architecture for this problem, *iTracker*, which achieves significant reduction in error over previous approaches when performing in real-time on a mobile device.

2.3. Recent approaches to mobile gaze tracking

In a recent EE267 project (Spring 2018)[7], the authors attempt to use a 3-layer deep CNN architecture to track gaze direction vectors. Theirs is a slightly different problem to tracking point of focus on a screen (the purpose of this work), but their quantitative results shine a light on the role quality data in driving forward research in this field.

A past CS 231A project at Stanford proposed the model **Gazelle** [20], which supplements the images in GazeCapture with Histogram of Oriented Gradients (HOG)[3] feature vectors. The authors used OpenCV to generate the HOG feature vectors, and given the large variation in gaze conditions in the GazeCapture data (figure 3) and computational limitations they could only generate feature vectors for 25 k samples out of over 1 million frames in the dataset. The functions they used to generate the HOG vectors failed to return satisfactory results for a majority of the images in

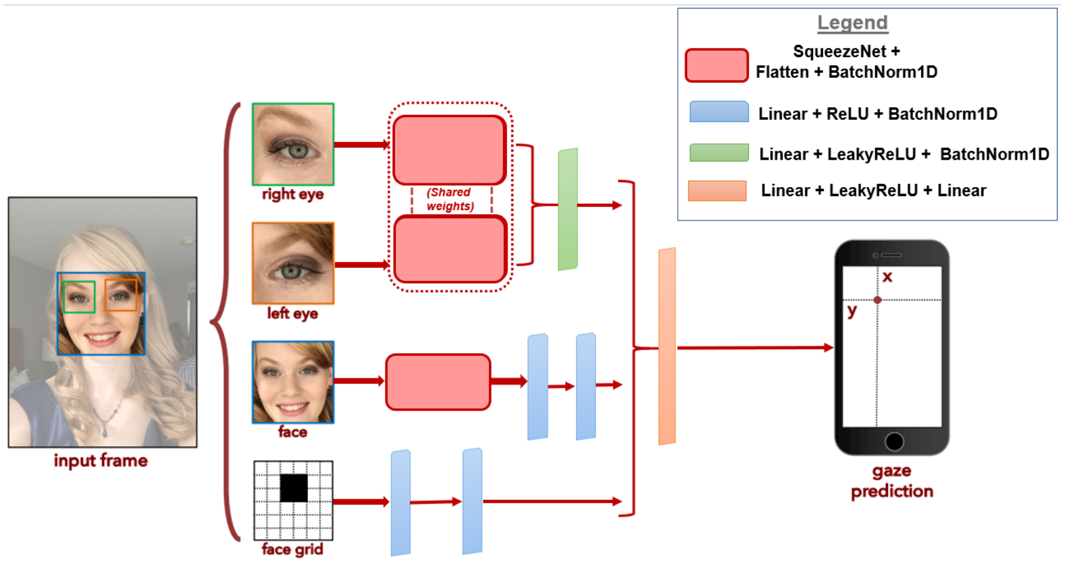

Legend
SqueezeNet + Flatten + BatchNorm1D
Linear + ReLU + BatchNorm1D
Linear + LeakyReLU + BatchNorm1D
Linear + LeakyReLU + Linear
right eye
left eye
(Shared weights)
face
face grid
input frame
x
y
gaze prediction

Figure 2: MobileGaze Architecture. The same subject is used as [15] to avoid potential conflicts in regard to privacy of other subjects.

the dataset, possibly due to partial occlusion of some images, low image resolution or poor illumination. As a result, the final accuracies they obtained fell quite short of the metrics achievable through iTracker. However, the authors ran control experiments over their limited dataset with and without the HOG vectors and showed that applying HOG does indeed help the CNN model improve performance. This won't be a direction we pursue in this work, since we want to ideally utilize the full breadth of the data.

2.4. Efficient gaze-tracking

An ideal fully functional gaze-tracking model would need to perform inference on a user every second of time, and be versatile enough to fit in memory of a mobile device; algorithmic efficiency (both in space and time) is therefore a crucial consideration when building these models. The authors of GazeCapture found their model to perform slightly better than AlexNet[16], which set a benchmark on the ImageNet classification challenge for multiple years. More recently, the authors of SqueezeNet [12] achieved classification accuracies on ImageNet comparable to AlexNet, but with a model that contained roughly *50 times fewer parameters* and could fit in *less than 0.5 MB* of memory after compression.

Running inference on batches of image frames every second would also require the ideal gaze-tracking model to have minimal computational *inefficiency*. Recent work in optimizing time complexities of deep CNNs[[10], [24], [32], [19]] has focused on developing and improving modules that eliminate unnecessary convolutions in the channel dimension in the CNN architecture. MobileNet[10] performs spatial and channel convolutions in 2 steps via separable convolutions, the second step utilizing a 1x1 conv layer similar to that used in SqueezeNet. ShuffleNet[32] addresses the computational bottleneck of the 1x1 conv block by performing grouped convolutions before shuffling the channels. This incorporates stochasticity in the channel connections. MobileNet-v2 [24], despite its name, takes a very different approach to MobileNet in tackling the computational bottleneck - expanding and then compressing the number of channels with depth-wise convolutions in between.

3. MobileGaze

MobileGaze is built to primarily optimize one metric - memory usage. To this end, we propose an architecture that utilizes SqueezeNet([12]) as a feature extractor from input images of the face and eye crops. Table 2 in the appendix compares the model size (in number of parameters) and computational cost (in number of floating point operations or FLOPs) of popular architectures designed to run on mobile platforms.

Clearly, there is a trade-off to be made between model size, FLOP count and learning capacity, as measured by the ImageNet[4] top-1 accuracy. We choose **SqueezeNet-**1_1 not only because of its size, but also because it delivers a decent performance on the ImageNet Classification task with a computational overhead the lowest among the 'mobile' models that reached close to 60%.

The compact size of the MobileGaze architecture gives it the following advantages over iTracker[15] and Gazelle[20]:

- Can be more feasibly deployed onto FPGAs and hardware with limited memory
- Requires less bandwidth to deploy from the cloud onto a mobile device
- Requires less communication among servers during real-time training

We segment the architecture into three main modules, based loosely off the skeleton of that used in [15]:

- An **eyeModel** module tasked with extracting specific information regarding orientation of the eyeball from cropped images of **each individual eye**. This A SqueezeNet feature extractor is combined with Batch Normalization to extract facial features from both eyes, separately. The outputs from images of both eyes backpropagate their gradients to this module.
- A **faceModel** tasked with extracting postional features of the eyes *relative* to other facial landmarks from a **square crop of a face**. We want this module to have*translational invariance* to these landmarks for high-level features, such as detecting the presence of eyes in an image, but we also want this module to have some understanding of inter-pupillary distance (i.p.d.) that can help predict gaze position. We therefore use a feature extractor similar to the eyeModel described above, but with linear layers at the end to help inject translational variance into this module.
- A **faceGrid** module tasked with extracting the location composed of a series of linear layers to process the binary face mask signaling the location of the face. We forego the use of a convolutional layer for this purpose because convolutional layers inject some degree of their translational invariance to inputs, which is exactly what we *don't* want for this module. We instead choose a series of linear layers with Batch Normalization before each.

We forego the use of any feature augmentation for MobileGaze. Feature augmentation such as **Histogram of Oriented Gradients (H.O.G.)** add a computational overhead at test-time, and as the authors behind Gazelle([20]) have shown, drastically reduce the size of the set of examples in GazeCapture that the model can be trained on. We instead rely on the depth of SqueezeNet to extract lower level features such as edges accurately from each image, eliminating the need for this extraneous computational load.

The architecture for MobileGaze has been illustrated in detail in Figure **??**. There were deliberate design choices made to serve as improvements over the existing state-of-the-art in [15], and they are briefly discussed below.

3.1. Batch Normalization

Internal Covariate Shift is the redistribution of activations between layers in a neural network caused by the layers' parameters being continuously updated during training. As the outputs of one layer feed into the next, the weight initializations in a deep network have to be *carefully* tuned such that the activations of each layer in the network lie within the same relative order of magnitude, as the network would otherwise suffer from vanishing/exploding gradients if sufficiently deep. This phenomenon also negatively impacts networks with saturating non-linearities, as it increases the likelihood of the activations constantly falling outside the 'linear' range of the activation, and thus the gradients for those forward passes being extremely small for that layer. Both these problems used to require small learning rates and careful hyperparameter tuning to solve. **Batch Normalization** ([13]) attempts to solve them both by renormalizing the activations to each layer based on the mean and variance of each input neuron computed *across the batch*. Larger batch sizes favor this technique due to better estimation of global activation statistics for each layer.

We propose using 1-dimensional BatchNorm prior to each linear layer preceding the final linear layers. To implement the impact of this normalization at test-time, the algorithm keeps a running mean and variance of the activations of each neural layer. One added difficulty of using BatchNorm is setting the hyperparameter that regulates the running average and bias statistics computed at train-time. The results of this hyperparameter tuning are discussed in section 6.

The redistribution of BatchNorm injects stochasticity during training, as this process is hinged on non-deterministic batch statistics if the batches are shuffled. This acts as a regularizer for the network, and allows us to forego the use of **Dropout**([27]) outside the SqueezeNet for our purposes. As discussed in [18] these two techniques often conflict with each other as Dropout tends to inject the same internal covariate shift that BatchNorm eliminates.

3.2. Terminal Leaky ReLUs

ReLUs (Rectified Linear Units) became popular due to the common belief that sparsity in a neural network was good for regularization, preventing codependence of neurons during training. Recent research[30] has exposed some of the weaknesses of the ReLU activation, particularly at the earlier stages of backpropagation (i.e. terminal stages of network) where a negative activation can cause the gradients to an entire branch of the network to be cut off, which in subsequent iterations can prevent that subpart from learning at all (the **dying ReLU problem**). We thus place LeakyReLUs at the terminal branches for our network to pass a small negative gradient in the event subsequent negative activations arise for a neuron in a terminal layer.

4. Methodology

4.1. Loss Function:

The objective of the gaze-prediction task is to predict the (x,y) coordinates of the user's gaze, thus it would intuitively make sense to penalize the model on the *linear* distance between the ground-truth gaze and predicted gaze coordinates. The **Huber Loss** function was also experimented with, but offered no superior benefits over the **mean-squared-error M.S.E.** Loss, which was simpler to compute. The loss function the learning algorithm tries to minimize at each step is thus:

$$L_{MSE} = \frac{1}{|B|}\sum_{i\in B}(y_i - \hat{y}_i)^2$$

where,

B == minibatch

$$\hat{y}_i = \text{MobileGaze}(\{\text{Images}\}_i, \text{FaceGrid}_i)$$

4.2. Experimental setup

A few key implementation details should be mentioned for future attempts to replicate this work:

- **Initializations:** A SqueezeNet pre-trained on the ImageNet classification task with accuracy mentioned in Table 2 was used for this task. The architecture was **not fine-tuned** on GazeCapture but rather trained jointly with the rest of the network. **Kaiming initialization**[9] was used to initialize all other weights, and zeros used for all other biases.

- **Optimizer:** **Adam** optimization was used for all parameter updates with default values set in the original paper [14].

- **Learning Rate:** The learning rate turned out to be an important hyperparameter to tune for this task (see Section 6 for further discussion). An initial learning rate of **0.001** yielded promising results at the start, with the rate decayed to **0.0001** after 4 epochs.

- **Overcoming slow data loader**: This project made use of the unedited Pytorch Dataset class used in [15] [15]. An unresolved issue with loading the data led to the algorithm slowing down significantly during the middle of each epoch. To circumvent this, only *half* the dataset was looked at each 'epoch', with two such loops counting as one *full* epoch or pass through the dataset (with the dataset shuffled randomly at the start of each loop). The shortcoming of this strategy is increased exposure to a subset of the samples and higher likelihood of samples, but the author placed sufficient faith in the vastness of the dataset and regularization power of BatchNorm to overcome this.

- **Hardware:** Experiments were run on 4 x Nvidia Tesla K80 GPUs and Intel Xeon 16-core CPUs on Google Compute Engine. A batch size of 64 samples per GPU was used at training time and 16 samples per GPU at test time.

4.3. Evaluation

A rudimentary baseline for any gaze prediction task is to predict the average accuracy of a model that *deterministically* predicts the *center* of the screen for all samples. Based on the conditions in which our dataset was created, the average error would be deterministic for such a model, and is recorded in Table 1 as a simple baseline. A much more sophisticated baseline is one where facial images are passed through **AlexNet**([16]) and the activations at various layers used to perform **Support Vector Regression**([2]) to predict gaze coordinates. Note, the metrics provided in Section 6 were obtained from [15], in which specific numbers regarding computational cost were not provided. However, as the authors of [33] note, the computational cost of SVR is $O(n^3)$ in the number of input dimensions, and provided the *final* convolutional layer of AlexNet passing activations to a 2048-dimensional linear layer, which is the terminal point where activations can be extracted for the SVR before all translationally-invariant features are lost, a quick computation reveals a *lower bound* of **8.5 G** floating operations for this approach.

For quantitative metrics of evaluation, we use the **Euclidean (L2) distance** for comparison between models. In their CVPR submission, the authors in [15] used *temporal averaging* to improve results by considering frames taken over a few time steps to temporally average results and remove noise. For the sake of fairness, we compare our model to theirs without this feature, as it is not included with their Pytorch implementation. Instead, we compare our metrics directly against the results of their Pytorch implementation, which also doesn't fine-train on Mobile / Tablet data but aggregates data from both devices into the same dataset.

Alongside the L2 Error, we list the model size in number of model parameters, as well as the computational cost of each model in passing a **224x224** input through it. Note that the AlexNet+SVR approach only takes a single 224x224 face image unlike the other methods that also take in crops of each eye.

4.4. Saliency Maps

Saliency Maps provide an excellent way of visualizing the impact the input features to a neural network have on the final loss function(s). Introduced in [25], these maps are a single-channel visualization of the *absolute value of* *gradients* backpropagated all the way to the *inputs*. For a fully-trained network that takes images as inputs, the result is a single-channel image (of the same dimensions as its corresponding input during the forward pass) showing in bright the pixels that had the largest impact on the network's predictions. We can extend this technique to MobileGaze to show which parts of its inputs our network focuses on at inference time, and analyze any differences in saliency maps produced by our network and iTracker. Our Saliency Maps are generated on images from the validation set, to fairly analyze the model's ability to generalize to unseen examples.

5. Dataset

5.0.1 GazeCapture

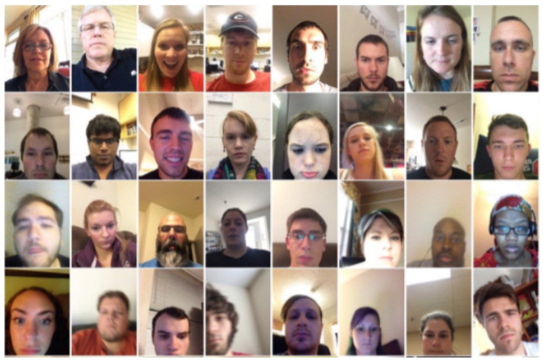


Figure 3: Samples from GazeCapture. Adapted from [15]

GazeCapture[15] consists of roughly 2.5M images along with gaze coordinates on a tablet screen captured from 1450 subjects. The data was collected through Amazon Mechanical Turk (A.M.T.) by asking users to look at a dot on the screen, and the coordinates of this ground-truth gaze then mapped to a prediction space (Figure 7b in the appendix) with the camera placed at the center. Figure 7a (appendix) shows a large degree of variation in the yaw angle of samples in the dataset, with comparable variations in pitch.

Figure 3 shows a set of unprocessed image samples from GazeCapture, demonstrating high degrees of variation in subjects' pose and environmental illumination. This helps the model learn to operate in a diverse set of real-world conditions, not just handcrafted ones in a laboratory setting. The dataset split between training, validation and test sets were:

- **Training:** 98.13 %, $\sim$ 12.51 M samples,
- **Validation:** 0.47 %, $\sim$ 58.48 k samples,
- **Testing:** 1.41 %, $\sim$ 179.5 k samples,

5.0.2 Data pre-processing

GazeCapture uses Apple's built-in facial segmentation libraries to crop out images of the eyes and face. For all experiments conducted on the dataset, these crops are processed *before training and testing*. While we do acknowledge a computational overhead associated with the crops, we also recognize that segmentation is a far more well-solved problem than gaze-prediction, and since the goal of this project is to optimize the latter, we use the same pre-processed crops that other projects on this dataset have used.

6. Results and Discussion

6.1. Quantitative Evaluation

Model	**Parameter Count (M)**	**# FLOPS (G)**	**Average L2 Error (cm)**
Center (baseline)	N/A	N/A	7.54
AlexNet (fine-tuned) + SVR	62.4	>8.5	3.09
iTracker (25 epochs)	6.29	1.41	2.46
MobileGaze (3 epochs)	2.05	1.16	2.22
MobileGaze (5 epochs)	2.05	1.16	**2.12**

Table 1: Comparison of quantitative results. All models had inputs: face, left eye, right eye images of shape 224 x 224 and binary face masks of shape 25 x 25. The number of FLOPs were computed for a single-frame sample.

MobileGaze outperforms the state-of-the-art iTracker on the test set *after just 3 epochs* of training (Table 1). After 5 epochs of training, MobileGaze achieved accuracies of **1.89** cm on the validation set and **2.12** cm on the test set. All these results were obtained *without ensembling* or *calibration*, both of which have been shown to improve inference time metrics on this dataset [15]. MobileGaze also achieves these metrics with a model that is both *smaller* in size and *faster* in terms of computational cost.

Figure 4 shows the progress of validation error rates with training epochs. Training had to conclude prematurely due to the computational and time constraints on this project, but as the trend shows, MobileGaze could still improve on the validation set with more training.

6.2. Analyzing the *network*'s gaze

Figure 1 shows a composite image constructed of saliency maps produced by the iTracker Model. We notice iTracker is generally good at locating the positions of eyes

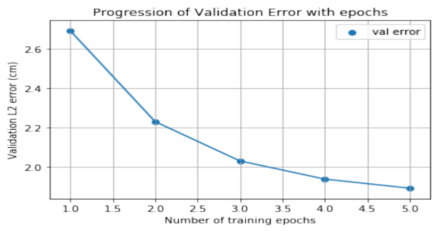


Figure 4: Progression of L2-error on the validation set over number of training epochs completed

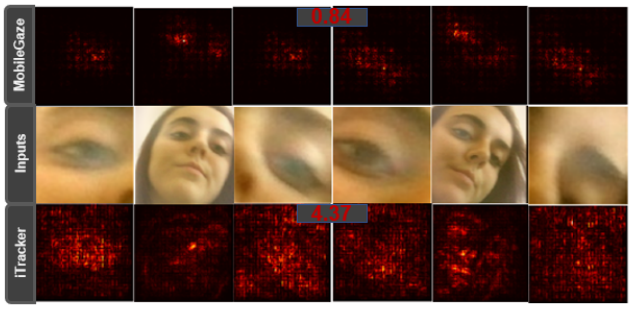
across an array of conditions, failing partly only in cases where the subject is wearing spectacles with sharp specular reflections (subjects at the center of the composite). We now want to compare these maps with those produced by the more accurate MobileGaze.

In order to compare the differences between saliency maps produced by the two models, we compare maps produced for the *same* inputs to both models for specific edge conditions, with the MobileGaze trained for 5 epochs and the iTracker trained for 25. We also list the average L2 error for the minibatch of which the sample was a part (using minibatch sizes of 2 to fulfill the minimum requirement for Pytorch batch normalization). It should also be noted that these examples are handpicked, and one should **not** assume superiority of one network over another for these specific conditions based on these examples alone. These are purely to give the reader an intuition of where the strengths and weaknesses of each network lie.

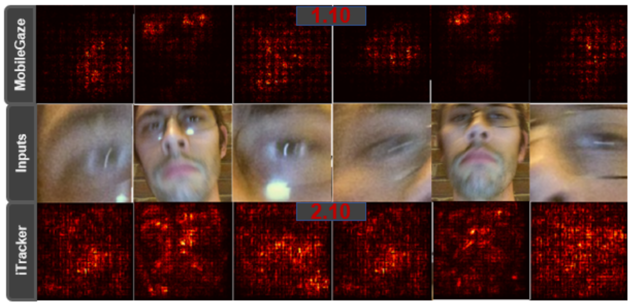
Figure 5 reveals clearly that the saliency maps for iTracker are brighter than those of MobileGaze. This indicates that the gradients flowing back to inputs are larger in iTracker than those in magnitude than those in MobileGaze. This has important implications for different input conditions as the analysis below reveals.

- **Subject with varying head orientation:** Large changes in head orientation reveal facial features other than eyes to the camera. As shown in the facial crop saliency map for iTracker in Figure 5a, iTracker hones in on the lips of the subject in one of the two samples when the left eye is obscured. Being more conservative in its gradient flow, MobileGaze only focuses on the only visible eye in that facial crop. This results in MobileGaze ending with a far lower error (0.84 cm) than iTracker (4.37 cm).

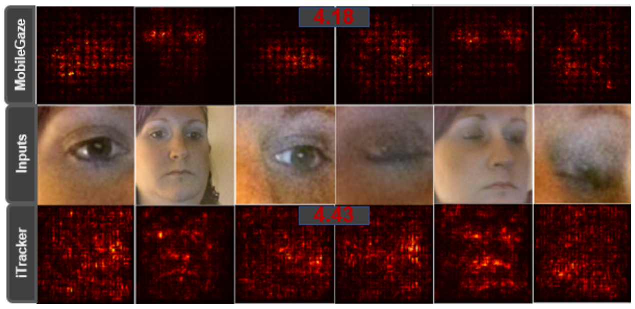
- **Blurry images:** Blurry images test one of the most realistic challenges faced by any computer vision network in practice. The conservativeness of MobileGaze's gradients help it distinguish the locations of the eyes from the noise better in the two facial crops in Figure 5b. This shows a robustness to noise injected by the gradient flow in MobileGaze.



(a) Saliency maps for a subject with varying head orientation



(b) Saliency maps for a blurry image



(c) Saliency maps for a blinking subject

Figure 5: Comparison of Saliency Maps for MobileGaze (top row of each subfigure) and iTracker (bottom row of each subfigure). The center row shows the raw image inputs to the networks, and each triplet of columns represents a single data sample.

- **Blinking subjects:** Although GazeCapture is largely free of samples with blinking subjects, there are cases in the validation set where subjects are indeed blinking. Due to the sparsity of these examples, both networks understandably perform very poorly for this edge case. Although both architectures correctly identify the locations of the eyes from the facial crops, the hidden cornea prevents any reasonable degree of information from being extracted from the eye crops, as the maps in Figure 5c reveal. MobileGaze only performs marginally better, but there is no solid reason to believe this advantage would necessarily extend to a large set of examples of this edge case.

The saliency maps reveal that the gradient flow back through the network is far more conservative for MobileGaze than it is for iTracker. While this does mean one has to be more careful with hyperparameter tuning to protect against the **vanishing gradients** problem, this also makes the MobileGaze architecture more robust to noise in its input facial crops. Indeed, of the three image inputs fed into the network (left/right eye crops and facial crops), it is in the last input where MobileGaze truly has the edge, as the saliency map comparison shows. We can conclude that this is perhaps one of the key reasons for its superior performance over iTracker.

6.3. Comparison of batch statistics within and between training and validation sets

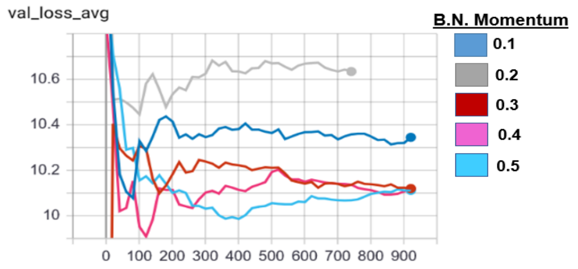


Figure 6: Change in average validation loss with number of iterations for the 1st training epoch. Note how the loss goes very high for some values of the momentum parameter

Changing the momentum parameter (Pytorch syntax) for batch normalization (which controls the weight new samples are given in the running average) yielded varying results. A range between 0.1 and 0.5 was tested with, and after the first epoch of training it was clear the default value (0.1 in Pytorch) was not a suitable candidate for the task6. We instead choose **0.3** as a momentum parameter. Although this suggests a large degree of variation in samples *within each epoch* during training, warming up the BatchNorm running means and variances on a small subset of the validation set (without backpropagation) was actually found to *worsen* performance on the validation set. This is indicative of the training having large internal variation overall, but a large degree of alignment in batch statistics to the validation set. This can be explained by the images being mean-centered, but not entirely normalized (resulting in large internal variation) and the training and validation sets being sampled from distributions that are largely aligned overall.

6.4. On making predictions based on facial location alone

In order to see the impact of the binary mask input on the gaze predictions, we can train the network as a whole for an entire epoch, then freeze all the weights but those in the faceGridModel(Section 3). This was implemented for us free of charge because of an initial learning rate (0.05) too high that caused the gradients after the first epoch to explode in all but the layers in the faceGridModel. The training L2 error plateaud at around 3.93 after 5 epochs of training, while the validation error plateaud around 4.25 cm. This shows the faceGrid binary mask, which only signals the position of the facial crop in the original frame, is a very poor predictor of gaze.

7. Conclusion and Future Directions

In this work, we propose MobileGaze, a compact neural network that outperforms the existing state-of-the-art architecture in the task of mobile gaze tracking. Our proposed model beat the S.O.T.A. iTracker after just 3 epochs of training on the GazeCapture dataset (L2 error of 2.22 cm compared to 2.46 cm), achieving average L2 errors as low as 2.12 cm after 5 epochs. With further training, the network could have achieved even lower error rates, given the declining trend in validation errors. We have also utilized saliency maps to analyze the parts of its inputs MobileGaze focuses on when making predictions, contrasting these maps with those obtained using the iTracker model proposed by the authors of the GazeCapture paper. The saliency maps have revealed to us that the magnitude of dependency of predictions of MobileGaze are a lot less reliant on individual pixels than those in iTracker, especially in the case of processing facial crops. We conclude this as one of the key reasons behind MobileGaze outperforming iTracker so early in its training stages.

Given more time and computational resources, the author would want to obviously finish training MobileGaze for the full 25 epochs and analyze performance on the basis of saliency maps and quantitative metrics. For a model that converges to satisfactory results after just 3 epochs, it would be interesting to see how the saliency maps for a small set of validation set inputs evolve with training. Given the interesting differences we have observed between how the two network's process inputs, it would be worthwhile to run a more comprehensive survey over saliency maps for a larger set of input conditions. We could also look at adversarial examples with ‘fooling images’ to test the robustness of MobileGaze - how much can we perturb a well-recognized input before we start getting high error rates? Fooling images could enhance our understanding of the set of conditions that could make a well-performing network like MobileGaze obtain low accuracies in real-world conditions, and help us prepare for those contingencies before deploying this model.

8. Acknowledgements

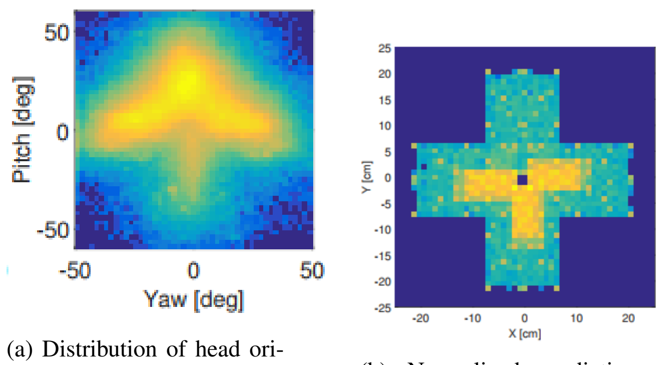
The author thanks the creators of the GazeCapture dataset - Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, Antonio Torralba. Large neural networks need a lot of reliable data to fuel them, and their hard work has made projects tackling the gaze prediction problem with deep neural networks possible. A special thanks goes to Petr Kellnhofer for adapting the original iTracker code to Pytorch, which helped the author get started on this project. The author also acknowledges the development team at Facebook responsible for creating and maintaining Pytorch[23] - their work has opened innumerable doors to Deep Learning researchers worldwide, including the author of this work.

The author thanks Gordon Wetzstein and Robert Konrad for their constant feedback and motivation for this project. Working alone on a paper can be a daunting task with self-repeating patterns of thought, and Gordon and Robert provided the voices that stirred this ship towards its destination when things were shaky at the start.

Last, but certainly not the least, I personally thank Stanford's CS231N TAs Winnie Lin, Owen Wang, Danfei Xu and William Shen for generously lending their time and expertise to a project that was technically not a part of 231N. Generosity of this magnitude reminds me everyday that there is nothing artificial about artificial intelligence.

9. Appendix

The code for this project is available privately in a GitHub repository. Please contact the author for access to the code base



(a) Distribution of head orientations for subjects tested in GazeCapture, weighted according to their number of samples in the dataset.

(b) Normalized prediction space for samples in the dataset.

Figure 7: Heat maps illustrating distribution of pose and the prediction space in GazeCapture. Brighter regions correspond to a higher density of samples. Source: [15]

Model	**Parameter Count (M)**	**# FLOPS (G)**	ImageNet Challenge top-1 Accuracy (%)
AlexNet	61.10	0.71	60.3
densenet-169	14.15	3.33	76.2
resnet50	25.56	3.53	75.3
GoogLeNet	6.8	1.55	69.8
0.25-MobileNet-224	0.5	0.041	50.6
1.0-MobileNet-224	4.2	0.569	70.6
ShuffleNet (0.25x)	0.833	0.013	47.3
squeezenet-1_0	1.25	0.70	57.5
squeezenet-1_1	**1.24**	**0.34**	56.34

Table 2: Comparison of model size, computational cost in number of floating-point operations (FLOPs) and ImageNet top-1 accuracy rates for architectures considered for this project. All comparisons were for standard input sizes used in these networks (mostly 224x224), with single-crop inputs used for all models.

References

[1] S. Baluja and D. Pomerleau. Non-intrusive gaze tracking using artificial neural networks. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 753–760. Morgan-Kaufmann, 1994.

[2] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[6] A. J. Glenstrup and T. Engell-nielsen. Eye controlled media: Present and future state. Technical report, 1995.

[7] J. Griffin and A. Ramirez. Convolutional neural networks for eye tracking algorithm. EE 267 final project, 2018.

[8] A. Haro, I. Essa, and M. Flickner. A non-invasive computer vision system for reliable eye tracking. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, pages 167–168, New York, NY, USA, 2000. ACM.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[11] Q. Huang, A. Veeraraghavan, and A. Sabharwal. Tabletgaze: Unconstrained appearance-based gaze estimation in mobile tablets. 2015.

[12] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$1mb model size. *CoRR*, abs/1602.07360, 2016.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[15] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba. Eye tracking for everyone. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[18] X. Li, S. Chen, X. Hu, and J. Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

[19] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[20] N. N. Matthew Kim, Owen Wang. Convolutional neural network architectures for gaze estimation on mobile devices. CS 229 final project, 2017.

[21] C. D. McMurrough, V. Metsis, J. Rich, and F. Makedon. An eye tracking dataset for point of gaze detection. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 305–308, New York, NY, USA, 2012. ACM.

[22] M. R. M. Mimica and C. H. Morimoto. A computer vision framework for eye gaze tracking. In *16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003)*, pages 406–412, Oct 2003.

[23] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[24] M. B. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[25] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[26] B. A. Smith, Q. Yin, S. K. Feiner, and S. K. Nayar. Gaze locking: Passive eye contact detection for human-object interaction. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 271–280, New York, NY, USA, 2013. ACM.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[28] J. van der Geest and M. Frens. Recording eye movements with video-oculography and scleral search coils: a direct comparison of two methods. *Journal of Neuroscience Methods*, 114(2):185 – 195, 2002.

[29] U. Weidenbacher, G. Layher, P. . Strauss, and H. Neumann. A comprehensive head pose and gaze database. In *2007 3rd IET International Conference on Intelligent Environments*, pages 455–458, Sep. 2007.

[30] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[31] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Appearance-based gaze estimation in the wild. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4511–4520, June 2015.

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[33] P. Zhong. Training robust support vector regression with smooth non-convex loss function. *Optimization Methods & Software - OPTIM METHOD SOFTW*, 27:1039–1058, 12 2012.