EyeXception: Eye-gaze Estimation with Deep Learning

Peter Grönquist
petergro@student.ethz.ch

Xiaochen Zheng
xzheng@student.ethz.ch

ABSTRACT

Eye-gaze estimation has gained a lot of renewed interest in the past years, as new methods of visual classification and regression emerge in the field of neural networks more and more frequently. In this project report concerning the ETHZ Machine Perception course 2019, we explore different classification model adaptations stemming from deep learning, whilst combining it with methods from computer vision. We show that by using the combined methods and applying a statistical evaluation it is possible to obtain state of the art predictions in eye-gaze approximation.

1 INTRODUCTION

As the popularity of unconstrained eye-gaze estimation arises so do the available projects. And as such we were assigned the 'Eye-gaze approximation' Project. The task consists of predicting the general gaze angle of pitch and yaw in radians, from an unconstrained eye-gaze dataset called *GazeCapture*[10]. Novel in this work is that a lot of the data has already been conveniently preprocessed for us into useful features that we can select from, which in theory allows us to compete with the state of the art models.

Upon analysis of previous models and methods, we work out three most important parts to an approximation model:

- **Filtering:** In the filtering process it is our goal to facilitate the learning for our neural network. We do this by allowing it to focus solely on important features, namely edges. We place special importance on the recognition of the eyes, the iris and the pupil. The process can be compared to an automated worse but cheaper version of what has already been done with auxiliary learning in stacked hourglass structures[13].
- **Classification:** In the classification process the model has the task of efficiently learning and selecting out important features that it will then present to the regression layers for calculations.
- **Regression:** Finally, from the feature maps provided by the classification module, our model learns by connecting the different maps through several dense layers before ending in a last linear regression layer that outputs pitch and yaw angle as our predictions.

It is important to note that as we start late in the competition, a lot of observations were time-constrained. We would've liked to elaborate on them further, however we still manage to enter the top 2 on the leaderboard with our proposed solution.

2 RELATED WORK

Gaze estimation methods can be divided into model-based and appearance-based. Appearance-based methods directly use eyes image as input and require large amounts of user-specific training data. As such, many works immediately refer to using CNN architectures as a method of gaining information from the image structure[1][4][10][17]. Further work on[13] established that classification models in general are the direction to head towards, as it has also become clearer that for accurate estimations, physical models and neural network have to work hand in hand.

In *ITracker*[10], an end-to-end convolutional neural network for eye tracking is built which uses left eyes, right eyes, face images and face grid as inputs. Several convolutional layers whose size is based on *AlexNet*[9] are implemented with weights sharing between two eyes. Similar to *ITracker*, the structure of *RT-GENE*[4] is based on *VGG-16*[14] which proposes a real-time method considering the problem of robust gaze estimation in natural environments. It uses left eye, right eye, and face as inputs of three independent convolution blocks. The face image estimates the head pose which concatenates with the outputs of two eyes as the inputs of the next dense layers.

Besides all those well established convolutional network architectures such as *VGG-16*, *VGG-19*[14], *AlexNet*[9], and *ResNet*[6], the hourglass architecture[11][12] has been proven to be very effective in tasks such as human pose estimation and facial landmarks detection. The conventional gaze estimation models which learn a direct mapping from raw eye images to gaze direction can be more complex and less accurate. It has been shown[13] that such stacked hourglass structures can yield a better performance by applying a loss function on intermediate outputs of a network. In the paper it is shown that learning an intermediate image representation of the eye, a so-called *gazemap*, is possible. Based on these findings, the task of gaze estimation is then reformulated into two concrete tasks: regression from eye images to *gazemaps* through stacked hourrglass networks and secondly a regression from *gazemap* to gaze.

3 DATA AND MODEL

3.1 Data

The Data made available to us consisted of RGB images of the right and left eye, the eye region, the face, the head rotation as well as several facial landmarks as depicted on the face grid. While there exists interesting work on using facial landmarks[10] and using the eye-region alone might be interesting, the work that could be done was time limited and thus we had to reduce our selection to only using both eyes, the face and the head rotation.

3.2 Data Preprocessing

The Data is originally from the *GazeCapture* dataset[1], and is made available to us in RGB h5 format(See *Tab. 1*). Not only has the data already been split into train, validation and test set for us, but the face image has already been preprocessed, more specifically the initial image was warped such that the subject is placed at a consistent distance to the camera, with the head rotated to be upright[16].

[1]http://gazecapture.csail.mit.edu

Table 1: Dimensions of available features.

Feature	Channels	Height	Width
face	3	224	224
eye-region	3	60	224
left-eye	3	60	90
right-eye	3	60	90
head	-	1	2
face-landmark	-	33	2
gaze	-	1	2

Now we want to take away as much work from our Neural Network as possible. For this we receive a code skeleton that would allow us to read in data from the h5 format but it is already doing some preprocessing for us. The code normalizes the data (between -1 and 1), also it is doing so in the process of loading it up, thus slowing down the training time. Furthermore a lot of the skeleton is dated and does not easily support functions we'd like to use, so we code one again from scratch. This allowed us to do a standardization: Taking the mean and the standard deviation from the training data. Subtracting that mean from train, validation and test set and dividing by the standard deviation. This seemed like a good compromise of highlighting differences while reducing the training time (by keeping the input values small). We also tried training on the normalized dataset, but got better validation values for our standardized one. To speed up the loading up of data by using functions such as preloading and shuffling we save our data in TFRecord[2] form.

3.2.1 Filtering. In the mindset of preprocessing our data further we looked into the realm of filters and their possibilities as they are at the basis for edge detection in Computer Vision. After adapting an initial edge detection process (using a Sobel filter) on our input (then concatenating our input to it) and finding success in better validation results, we wanted to look further into it. However being limited by time decided to look into speeding up this process with convolutions using a stride of one as they act as filters as well. Looking into edge detection, the filters applied are high pass filters, which can be effectively implemented by using spatially separable convolutions, which have a lesser computational toll. Thus we decide to use a simple inception module[15] to include a variety of trainable high pass filters (3x3,5x5,7x7) through a neural network (See *Fig. 1*). This allowed us to perform better on simpler neural networks such as a simplified version of the ITracker[10], but performed worse on deeper networks. We imagine the learning to be too hard for a structure that is too deep. It is also important to note that the inception module also add to the computational toll. Thus we decided to remain with the concatenated result of the Sobel edge detection and our standardized image.

As this work is limited as a homework project, we could not further delve into this topic but it is our impression that further work can be done into prefiltering the images for efficient edge detection by pretraining the inception module on edge detection. This could then be further used to simplify and maybe enhance the hourglass structures currently used in eye gaze estimation[13], which we were also not allowed to look further into in the scope of this work.

[2]https://www.tensorflow.org/tutorials/load_data/tf_records

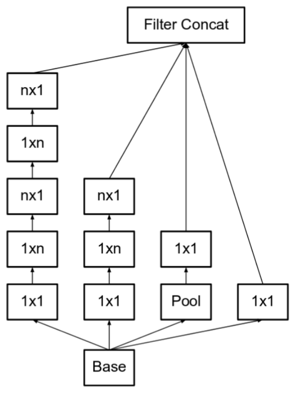

Figure 1: A simple inception module with spatially separable convolutions of size n. In our training it was applied to the provided head and eye images.

3.3 Model

For our model we quickly determined that classification followed by fully connected layers was the direction to turn to, already having great success in models such as hourglass structures[12][13]. Under this assumption we set out to try out many different classification structures concurrently in an effort to save time. Thus certain findings here can certainly be improved upon. We tried both 18-layer ResNet(*ResNet-18*)[6] and 50-layer ResNet(*ResNet-50*) with weights sharing for both eyes and found that in limited training time, structures with less layers perform better. Then we applied *SENet*[8] blocks into the *ResNet-18* to get a faster convergence and lower validation loss. We also experimented with the stacked hourglass[12] architecture between the inputs and ResNet to add more features related to eye gaze.

3.3.1 Classification. The Xception model[2]*(See Fig.2)* was our final choice, as it achieves top scores on many classification tasks whilst being proportionally computationally easier to handle. We implemented it according to the architecture from the paper (See *Fig.2*), making adaptions on the downsizing, strides and the tail of the network. Ending up with different *light* networks for face and eye classification. We use the classification networks on the face as well as on both eyes, for which we decided to reuse the weights. This was decided because of two reasons. Firstly it would have gone over the time constraints of this project to introduce an auxiliary learning layer for the classifications as done in the hourglass network[13]. To counteract this, by sharing the weights in the eye classification models, our network is able to concentrate on forms that are shared by both eyes, meaning the pupil and the eye in general, as opposed to gathering more data from a second network, which is done in other work[1]. Secondly we are predicting the general gaze represented by a vector in the middle of the eyes, so we aim to obtain a similar gaze vector from both eyes. If we gain no information from one eye (being obstructed or such) the network then theoretically still has enough filters to be able to pass along a

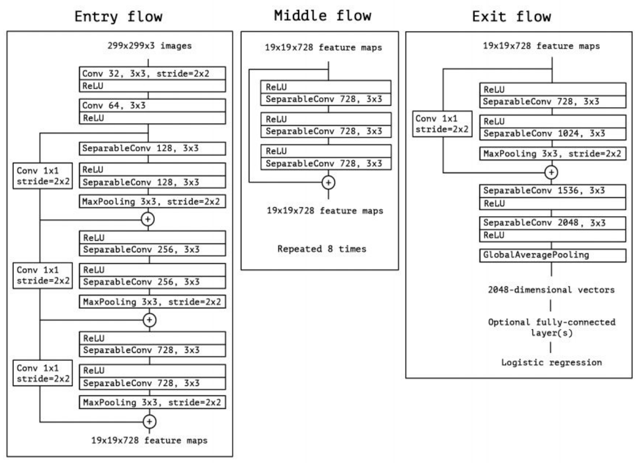

Figure 2: Basic Xception model structure[2].

value that reduces the effect of that specific eye classification, even with shared weights.

3.3.2 Dense Layers. In a second step we process the outputs of our classification models by flattening the filters of the face image and the eyes respectively. The head filters are then further reduced into less hidden units by dense layers, as to control the impact it plays on the final result. We do keep it at the same size as each single eye in the end though, as we found out in an accident of falsely loading the eye images, that our model classifies the eye gaze accurately past the easy baseline already with just the head as input. Thus the classification from the face can be used as a factor to optimize the eye gaze estimation. With the same idea in mind we add the head angle through a dense layer of half the regular size (60 hidden units), to further its impact on the prediction by using it as a factor. Initially we thought of directly adding it to the output of our eye gaze estimation network, but found that the values were unreliable and thus decided to just use it as a regulating factor. All these layers are then concatenated together, let through another similarly sized dense layer, before getting regressed into two final values, our eye-gaze estimation. We apply batch normalization and ReLU activation on all layers except the two last ones in an effort to encourage generalization in our model. The structure for the dense layers was obtained by training on lighter networks, such as our *ITracker* implementation and tuning them, to make the training faster.

As we were trying out predicting the offsets into x and y direction instead of predicting the gaze angle we once made the mistake of using the *tangent* function on our prediction rather than the *arctangent* to calculate the MSE, but upon evaluation we realized we still had a model over the easy baseline. After inspection of the prediction values themselves, we saw that one adapted better to smaller angle estimations while the other adapted better to larger angle estimation. Which made us explore the different options of output normalization/standardization as well, leading to decent results, however by combining and averaging both predictions we arrived at our best prediction yet by then. By then further experimenting on our top 10 validation results and taking their average we arrived at the top 2 on the leaderboard with substantial improvements. This led us to the conclusion that the random weight initialization of our models had a much heavier impact than predicted. In which case and as an easy solution given the time constraint, we decided to just train 10 of our top models and to average over the input. Additionally further work on normalizing or adjusting the output by a certain factor could be done here.

4 RESULTS

4.1 Training

We train on the mean squared error of the angle error (in radians) for our regression as upon inspection of the dataset no outliers seemed evident. All layers are initialized with a truncated normal initializer with standard deviation of 0.000001 and a regularizer of the same scale (larger ones lead to worse training). We've also experimented with Xavier[5] and He[7] initialization, however this lead to worse results as well, maybe also due to our inexperience with such initializers. Additionally we do not use initialization for the separable convolutional layers as this again lead to MSE that were larger by a magnitude and also resulted in longer training times. We rather rely on batch normalization to introduce a difference in those weights. For training, after loading the TFRecords through a parser, the data is shuffled over with a shufflebuffer of size 15000. The larger the better, as there are 500 records per participant: just shuffling through anything less than 500 entries would be the same as not shuffling at all. We also had to use a lower batch size to allow our model to fit into the 10GB of GPU RAM made available to us, and as such we use a batch size of 16, but our assumption is that it would provide better gradient updates if a larger batch size was possible. We use early stopping on the validation error with a patience of two. Setting the maximum number of epochs to 10, any larger than that seemed to lead to overfitting. With our current batch size we already have 6251 updates per epoch. We use the Adam Optimizer[3] with an adaptive *alpha* value of 0.001, reducing it by half every second epoch. We've also experimented on different loss functions and factors but the MSE provided us with the best angular error values. Training one full model takes about 8h30 using 64GB of RAM, one CPU and a Nvidia GTX 1080 Ti GPU (provided to us by the ETH Zürich Leonhard cluster). For our best submission we run 10 of our models also on the validation data and take their average. It is important to note that the model is not fully fine-tuned and we are certain it can be improved upon with more time and adaptations.

4.2 Evaluation

Our model alone, without taking the mean of several predictions, evaluates to MSE values on the validation set in the ballpark of 4.6, which then evaluates to approximately 5.1 on the provided test dataset used for evaluation online. We can track these differences amongst others to a different gaze dataset being used for testing. We rule out overfitting in such a magnitude, as it is trained for few epochs and with a diminishing *alpha* for the AdamOptimizer. Additionally, to counteract other possible overfitting sources, we introduce more generalization methods such as batch normalization, also on dense layers, in an effort to compensate. Further adaptation

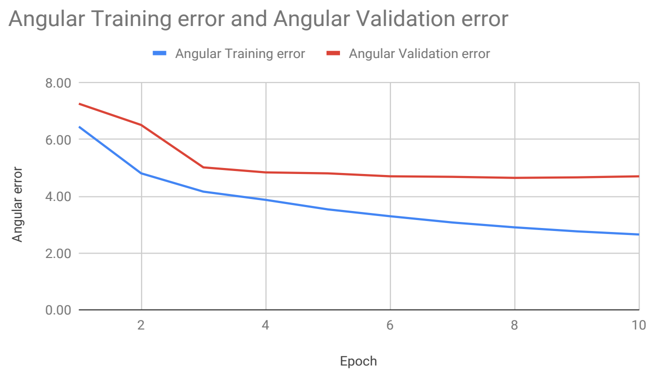

Figure 3: EyeXception model training of 10 epochs, the model starts to overfitting at epoch 9.

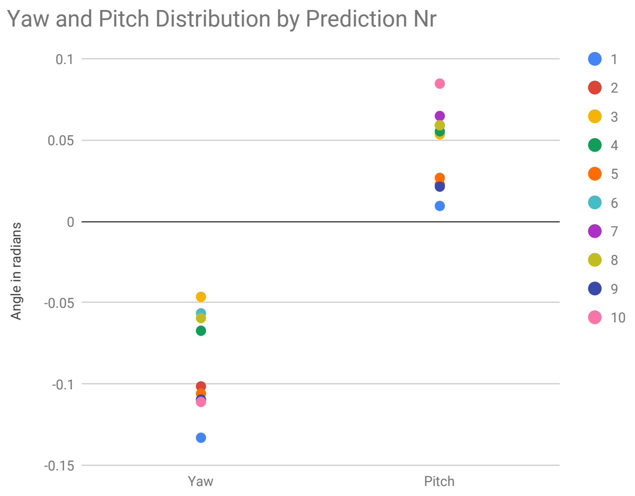

Figure 4: Distribution of 10 EyeXception models for 1 prediction.

possibilities would've been data augmentation: Even though the datasets are already very noisy, adding more variance might have helped for further generalization. Additionally it is important to pay attention to the randomness of the initialization and the subsequent training, which allowed us to have an MSE interval on the test set of around 5.0 to 5.3. As such it is very much possible that when reproducing our results by taking the average of several predictions it will be possible to either get better or worse results in the ballpark of an MSE of 4.7.

5 DISCUSSION

All in all, we learn from our models that with our current set up, our structure does not have enough capabilities to learn enough about the eye gaze. It is limited by the amount of filters and the random initialization amongst other things. We then end up with an ensemble of predictions of which we can take the mean as our final prediction and the standard deviation as measure of uncertainty in our prediction. For additional improvements we speculate that as an easy step, adding more preprocessing and edge detection filters will take away from the work our model has to do, additionally we can use more filters in the model itself, which is currently running with a base number of filters of 16. This however will be severely limited by GPU RAM. For further optimization we can definitely fine-tune the model hyper parameters further as we cut it short due to time constraints. Training inception module filters with an auxiliary loss is something else that we have hopes for. Finally, probably the easiest but more computationally intensive solution for better predictions would be to train more models, such as to gain a larger distribution set.

REFERENCES

[1] Yihua Cheng, Feng Lu, and Xucong Zhang. 2018. Appearance-Based Gaze Estimation via Evaluation-Guided Asymmetric Regression. *The European Conference on Computer Vision (ECCV)* (September 2018).
[2] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016). http://arxiv.org/abs/1610.02357
[3] Jimmy Ba Diederik P. Kingma. 2015. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)* (2015). https://arxiv.org/abs/1412.6980
[4] Tobias Fischer, Hyung Jin Chang, and Yiannis Demiris. 2018. RT-GENE: Real-Time Eye Gaze Estimation in Natural Environments. (September 2018), 339–357.
[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Vol. 9. 249–256. http://proceedings.mlr.press/v9/glorot10a.html
[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR* abs/1502.01852 (2015).
[8] Jie Hu, Li Shen, and Gang Sun. 2017. Squeeze-and-Excitation Networks. *CoRR* abs/1709.01507 (2017). http://arxiv.org/abs/1709.01507
[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. (2012), 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[10] Petr Kellnhofer Harini Kannan Suchendra Bhandarkar Wojciech Matusik Antonio Torralba Kyle Krafka, Aditya Khosla. 2016. Eye Tracking for Everyone. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (December 2016). https://doi.org/10.1109/CVPR.2016.239
[11] Alejandro Newell and Jia Deng. 2016. Associative Embedding: End-to-End Learning for Joint Detection and Grouping. *Computing Research Repository (CoRR)* abs/1611.05424 (2016). arXiv:1611.05424 http://arxiv.org/abs/1611.05424
[12] Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked Hourglass Networks for Human Pose Estimation. *Computing Research Repository (CoRR)* abs/1603.06937 (2016). arXiv:1603.06937 http://arxiv.org/abs/1603.06937
[13] Seonwook Park, Adrian Spurr, and Otmar Hilliges. 2018. Deep Pictorial Gaze Estimation. *The European Conference on Computer Vision (ECCV)* (September 2018).
[14] K. Simonyan and A. Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2015).
[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. http://arxiv.org/abs/1409.4842
[16] Xucong Zhang, Yusuke Sugano, and Andreas Bulling. 2018. Revisiting Data Normalization for Appearance-based Gaze Estimation. , Article 12 (2018), 9 pages. https://doi.org/10.1145/3204493.3204548
[17] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2016. It's Written All Over Your Face: Full-Face Appearance-Based Gaze Estimation. *Computing Research Repository (CoRR)* abs/1611.08860 (2016). arXiv:1611.08860 http://arxiv.org/abs/1611.08860