Practical Approaches towards Complete Real-time Gaze Tracking

Xin Li

4721101

Practical Approaches towards Complete Real-time Gaze Tracking

by

Xin Li

TU Delft

Practical Approaches $\overrightarrow{\text{towards}}$ Complete Real-time Gaze Tracking

Xin Li

Abstract

Visual context plays a key role in many computer vision tasks, and performance of eye/gaze-tracking methods also benefit from it. However, the size of contextual information (e.g. full face image) is very large w.r.t the primary input i.e. cropped image of the eye. This adds large computational costs to the algorithm and makes it inefficient, severely limiting its utility in real-time applications. In this paper, we perform a (computational) cost vs benefit analysis of various input types that include context, leaning towards an efficient gaze-tracking system. We further study the effect of an alternate ranking loss based training strategy. Finally, we demonstrate some practical calibration techniques that can convert gaze-vectors into points-on-screen, an important application that is often overlooked in literature. We examine how data-efficient these techniques are in terms of how well they utilise expensive calibration data.

1. Introduction

The task of gaze-tracking The task of camera based gaze tracking involves estimating where a subject's gaze is pointing based on the images captured. This is commonly in the form of a gaze vector, which determines the pitch and yaw of the gaze with respect to the camera [30]. A more complete form of gaze tracking further extends this by also computing at which specific point the subject is looking at on a screen in front of the subject [14, 29]. This is achieved by estimating the position of the said screen w.r.t. the camera (a.k.a. calibration), which is not precisely known beforehand. We present a study of core choices in the design of such gaze tracking methods in combination with calibration and training techniques, leaning towards an efficient real-time camera-to-screen gaze-tracking system.

Gaze-tracking based on context With the help of powerful deep learning based computer vision methods, there has been a significant improvement in the state-of-the-art accuracies in gaze tracking. A useful feature of deep learning (for computer vision) is its ability to infer information from large input image sizes to natively incorporate context from the whole scene. For example, subjective tasks like object classification are greatly benefited by the incorporation of such contextual information in an image background (cars are more likely to appear on roads, hence detecting roads in the background helps in recognizing cars) [26, 3]. The same effect is also seen to work for gaze estimation. The existing CNNs-based methods use not just the image of the eye(s) [18, 30], but also the whole eye region, the whole face [29], and even the whole image captured by the camera [14]. Such an improvement can be attributed to the correlation between the face appearance and gaze vector (e.g. a face facing left often also has its gaze towards the left). Thus, gaze tracking can clearly benefit from context.

Drawbacks of context dependence Depending on such contextual information has critical drawbacks for the task of gaze prediction. Unlike object classification tasks, estimating gaze is an objective and absolute measurement task: it is solely dependant on the elements that directly contribute in its creation (namely, the eye and pupil centre locations w.r.t. camera), and not on surrounding factors (like the state of the mouth, the facial expression, etc.). Therefore, while facial context can be helpful and correlated with a person's gaze, it is not the cause of the gaze itself and hence not vital for predicting gaze. Over dependence on such contextual factors can make generalizability harder to achieve and add bias into the model. We study the impact of various input types and sizes with varying amounts of facial contextual information to determine their value.

In addition, incorporation of such contextual data makes the overall system highly inefficient due to its computational load. This is because the relative size of the context here is much larger than the core input itself (e.g., the full face images are 20 times larger than image of eye crops). While such context improves accuracy, they severely limit the utility of such methods in real-time applications. For such practical applications, a good balance has to be sought between input size and computation speeds, and often heavy and expensive dependence on context is not affordable. Our computational cost vs benefit analysis can help practitioner find a good balance.

Calibration: gaze-vector to gaze-point Another aspect of gaze estimation that has received relatively little focus is

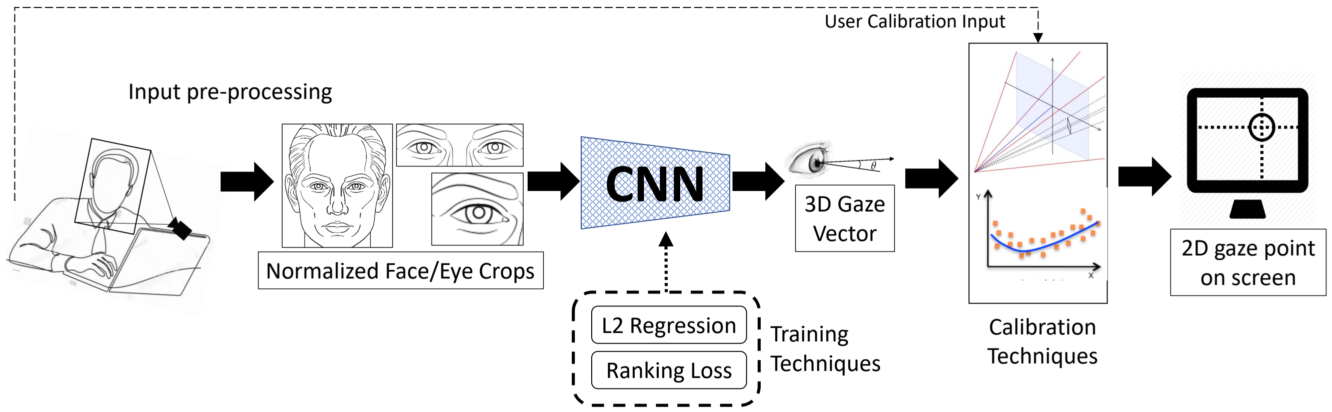


Figure 1: An overview illustration of our camera-to-screen gaze tracking pipeline. (Left to right) Images captured by the webcam are first pre-processed to create a normalized image of face and eyes. These images are used for training a convolutional neural network to predict the 3D gaze vector. Such training can be aided by a ranking loss based training scheme. With the cooperation of the user, the predicted gaze vectors can finally be projected on to the screen he/she is looking at using calibration techniques.

the task of predicting a gaze-point, the point on a screen in front of the subject where he/she is looking. In comparison with a gaze-vector, this gaze-point on screen is a more intuitive and directly useful result for gaze tracking based tasks. If the relative locations and pose of the camera w.r.t to the screen were exactly known, projecting the gaze-vector to a point on screen would be straightforward. However, this transformation is often not known in real-world scenarios, and hence must also be implicitly or explicitly estimated through an additional calibration step.

Multiple types of methods maybe applied to perform the $3D$ to $2D$ transformation. Geometry based modelling methods have the advantage that maximum expert/geometrical knowledge can be embedded into the system. One the other hand, such mathematical models are rigid and based on strong assumptions, which may not always hold. In contrast, machine learning based methods require no hand-crafted knowledge, although they may be more data dependant to learn the underlying geometry. In this paper, we present various calibration techniques including a hybrid approach between machine learning and geometric modelling. We examine the drawbacks and benefits of each technique.

Contributions In our work, we make the following contributions: **(i)** We shed light on the balance of gains from context-rich inputs vs their drawbacks. We study their individual impact on the system's accuracy w.r.t. their computational load to determine their efficiency and help practitioners find the right trade-off. (**ii**) We further study an alternate training strategy; **(iii)** Finally, we demonstrate practical screen calibration techniques that can convert the predicted gaze-vectors to points-on-screen, thereby performing the task of complete camera-to-screen gaze tracking.

2. Related Work

Appearance-based CNN gaze-tracking Appearance-based methods take the related eye images as input and make a mapping between images and gaze angles. As the deep learning methods have shown their potentials in many areas, some appearance-based CNN networks work efficiently for the task of gaze prediction.

Zhang *et al.* [28, 30] proposed the first deep learning model for appearance-based gaze prediction. This method utilized minimal context by using the grayscale eye image and head pose as input. Krafka *et al.* [14] presented a more context-dependant multi-model CNN to extract information from two single eye images, face image and face grid (a binary mask of the face area in an image). To investigate how the different face region contributes to the gaze prediction, a full-face appearance-based CNN with spatial weights was proposed [29]. Our work investigates the contribution of context in more detail by an in-depth ablation study.

Park *et al.* [18] proposed an hourglass [15] and DenseNet [10] combined network to take the advantage of auxiliary supervision based on the gaze-map, which is two 2D projection binary mask of the iris and eyeball. Cheng *et al.* introduced ARE-Net [5], which can be divided into two smaller modules. One is to find directions from each eye individually, and the other is to generates probability for the reliability of each eye. Deng and Zhu *et al.* [6] defines two CNN to generate head angle and gaze angle, which are aggregated by a geometrically constrained transform layer. Ranjan *et al.* [19] clustered the head pose into different groups and used branching structure for different groups. Chen *et al*. proposed Dilated-Nets [4] to extract high level features by adding dilated convolution. We build upon these foundations and join the body of recent work to obtain a better understand of gaze-tracking techniques and improve upon them.

Recently, a GPU based real-time gaze tracking method [7] for the natural environments was implemented using model ensemble fashion, taking two eye patches and head pose vector. This method achieves state-of-the-art in several datasets [7, 25, 30] for person-independent condition. In addition, [4, 16] have included some results about the improvements that can be obtained from different inputs. In our work, we perform a more in-detail evaluation add the dimension of computation load of each input type. Our insight in the cost vs benefit trade-off may help design efficient gaze tracking software that can run real-time on regular CPUs and not just GPUs.

Alternate regression training strategy For head pose estimation, alternate loss functions have shown potential to make more accurate predictions [9, 21]. Based on the observation that face patterns change significantly with the extreme head pose, Ruiz *et al*. [21] proposed a pipeline using the expectation of predicted bin as the final angle prediction, using both L2 and cross-entropy losses during training. Results from Hsu *et al*. [9] have also benefited from the bin label and an alternate loss. Unlike [21], QuatNet [9] has two separate networks following the CNN, one for regression and the other for ranking. By predicting the ordinal bin labels, the ranking net helps the model provide better feature for regression. In our work, we evaluate the applicability of this ranking aided training technique for the task of gaze-vector prediction.

Calibration: gaze-vector to gaze-point In a classical geometry-based model, projecting any gaze-vector to a point on a screen requires a fully-calibrated system. Except for the camera matrix, the screen pose and the 3D eye location in the camera coordinate system should be known. Using a mirror-based calibration technique [20], the corresponding position of camera and screen can be attained. This method needs to be re-applied for different computer and camera setting, which is non-trivial and time-consuming. During human computer interaction, information like mouse click may also provide useful information for calibration[17].

Several machine learning models are free of rigid geometric modelling while showing good performance. Methods like second order polynomial regression [12] and Gaussian process regression [27] have been applied to predict gaze more universally. The WebGazer [17] trains regression models to map pupil positions and eye features to 2D screen locations directly without any explicit 3D geometry. This is, however, strongly based on the assumption that people are always looking at the mouse cursor during the click.

As deep learning has shown its potential in different areas, other inputs can be mixed with CNN-based feature for implicit calibration[14, 29]. CNN features from the eyes, face are used as inputs to a support vector regressor to estimate gaze-point coordinates. These methods take advantage of being free of rigid modelling and show good performance. On the other hand, training directly on CNN features makes this calibration system non-modular since it is specific to one gaze-prediction system. In our work, we explore and evaluate some modular calibration techniques that convert gaze-vectors to gaze-points based on geometric modelling, machine learning, and a mix of both.

3. Method

The proposed pipeline can be contains three parts (see Figure 1). The first part performs initial input pre-processing by finding and normalizing the facial images. The second part is a CNN that takes these facial images as input to predict the gaze vector. The last stage converts the gaze-vectors to points on the screen. During calibration, the system learns this mapping between the gaze vectors and points with the cooperation of the user.

3.1. Input pre-processing

The input to the system is obtained from facial images of subjects. Through a face finding and facial landmark detection algorithm [1], the face and its key parts are is localized in the image. During training, these face location and landmarks are obtained directly from the datasets. Following the procedure described by Sugano *et al*. in [25], the detected 2D landmarks are fitted onto a 3D model of the face. By comparing the 3D face model and 2D landmarks, the head rotation and translation matrix R, T and the 3D eye locations e are obtained in CCS. A standardized view of the face is now obtained by defining a fixed distance d between the eye centres and the camera centre and using a scale matrix $S = \text{diag}(1, 1, \frac{d}{||e||})$. The obtained conversion matrix $M = S \cdot R$ is used to apply perspective warping to obtain a normalized image without roll (in-plane rotation). For training, the corresponding ground truth vector g can similarly be transformed: $M \cdot g$.

3.2. Implementation details: training and inference

The heart of this eye-tracking pipeline is a deep convolutional neural network. This neural network is trained to predict the pitch and yaw angles of the gaze vector with respect to the camera from the normalized pre-processed images faces/eyes. For this work, a VGG16 [23] network architecture with BatchNorm [11] layers is chosen.

Training Following the prior work in[30], the network was pre-trained on image from the ImageNet [22] dataset (all classes). This was done in order to start with a good initialization of weights in the network and ensure faster convergence. For all the experiments conducted in this work, we set the following hyperparameters for the training of the network for gaze-vector prediction: (i) Adam optimizer with default settings [13]; (ii) A learning rate decay criteria with a patience of 5 epochs; (iii) learning rate of 10^{-5}, decaying by 0.1 if validation error plateaus (up to 3 times); (iv) Simple data augmentation with mirroring and gaussian noise ($\sigma = 0.01$).

Inference This trained deep neural network can now make prediction of the gaze vector. The predicted gaze-vector (in the form of pitch and yaw angles) are with respect to the 'virtual' camera corresponding to the normalized images. The predicted virtual gaze vectors can be transformed back to the actual gaze vector with respect to the real camera using the transformation parameters obtained during image pre-processing. These vectors can then be projected onto a point on screen after the calibration step.

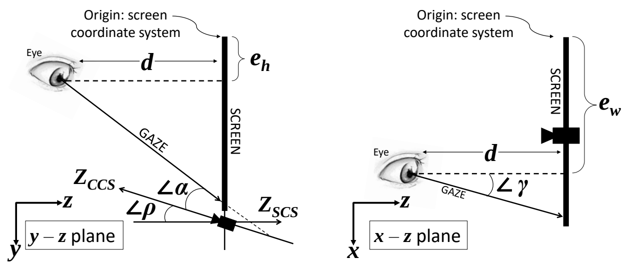
3.3. Ranking aided deep network training

The prediction of gaze-vector involves estimating the gaze pitch and yaw angles. Since pitch and yaw are both continuous scalar quantities, this training of the model can be performed via regression. Such a regression task can be converted to a classification task via binning. This approach has shown promise for head pose estimation [9, 21], which is a similar task. For this training, we follows the same pipeline as Hsu *et al.* in [9], and apply an additional ranking loss. To achieve this, the architecture of our deep network is modified by parallelly connecting an additional 256-dimension fully connected layer to the last fully connected hidden later of the network. This addition layer is trained for bin classification via the ranking loss. This ranking loss is defined as the sum of the cross entropy losses for predicting the bins of pitch and yaw. That is, $L_{\text{rank}} =$

$$\frac{-1}{N}\sum_{i}^{N}[\sum_{k}^{K_{\text{pitch}}}\sum_{j}^{m} h_{\text{pitch}}^{i,j,k}\log(g_{\text{pitch}}^{i,j,k}) + \sum_{k}^{K_{\text{yaw}}}\sum_{j}^{m} h_{\text{yaw}}^{i,j,k}\log(g_{\text{yaw}}^{i,j,k})], \tag{1}$$

where h_{pitch}, h_{yaw} denote the ordinal bin labels for the pitch and yaw angles, g_{pitch}, g_{yaw} denote the ordinal bin predictions for the pitch and yaw angles, and K_{pitch}, K_{pitch} denote the total number of bins for pitch and yaw. N represents the batch size. The final loss function is a sum of this ranking loss weighted by a parameter α, and the typical L2 regression loss (mean squared error):

$$L_{\text{total}} = L_{\text{regression}} + \alpha \cdot L_{\text{rank}}. \tag{2}$$



(a) (b)

Figure 2: A illustration of the geometric setup between the eye and the screen. (a) the $y - z$ plane between the eye and the screen showing the pitch angles; (b) the $x - z$ plan between the eye and the screen showing the relevant yaw angle.

3.4. Screen calibration to gaze-points

To project the predicted gaze vectors to gaze-points on a screen, the geometric transformations between the camera and the screen, the geometric transformation between image and camera must be known. The geomtric transformation can be attain by using a chess board pattern and calibration toolbox of OpenCV[2]. However, the relative location and pose of the camera w.r.t to the screen is not precisely known in most real-world scenarios. Because we focus on the task of webcam based eye-tracking, we may make some assumptions: (i) the roll angles between the camera and the screen is 0, and the yaw angles is 180 $^{\circ}/\pi$ rads, (ii) the camera matrix parameters are known, and (iii) the 3D location of the eyes is known w.r.t the camera (estimated by face modelling step). With these assumptions in place, we can design user-aided calibration techniques where the user cooperates by looking at predefined positions on the screen.

Geometry-based Calibration The information required to perfectly project a gaze-vector to a point on screen is given by the rotation matrix R and the translation vector T between the camera and the screen. With our assumptions in place, we only need two parameters to complete the transformation matrix: the pitch angle ρ between the camera and the screen norm, and the 3D eye location in the screen coordinate system Based on this, we can build a geometric model and estimate the transformation. This is shown in Figure 2.

By asking the user to look perpendicular at the screen plane and click on the point of gaze $\{x_{\texttt{screen}}^{\texttt{calib}}, y_{\texttt{screen}}^{\texttt{calib}}\}$ on the screen, the x and y value of 3D eye location $\{e_w, e_h\}$ can be roughly estimated. For calibration, the user is asked to sit at a fixed preset distance from the screen during calibration, so we can assume that the z value of 3D eye location is fixed and known.

We can define the gaze vector in the camera coordinate system (CCS) as

$$V_{\texttt{gaze}} = \begin{bmatrix} X_{\texttt{gaze}} & Y_{\texttt{gaze}} & Z_{\texttt{gaze}} \end{bmatrix}^T, \tag{3}$$

The pitch ρ angle along the y-axis between the camera and the screen can be determined by

$$\rho = \arctan(\frac{-e_h + y^{\texttt{calib}}_{\texttt{screen}}}{d}) - \alpha \tag{4}$$

where α is the pitch angle of the gaze in the camera coordinate system given by

$$\alpha = \arctan(-Y_{\texttt{gaze}}, Z_{\texttt{gaze}}). \tag{5}$$

Once ρ is known, we can compute the yaw gaze angle in the camera coordinate system as:

$$\gamma = arctan(-X_{\texttt{gaze}}, -Z_{\texttt{gaze}}). \tag{6}$$

Using this, we can compute the gaze point in the 2D screen coordinate system for static head poses as follows:

$$x_{\texttt{screen}} = d \cdot tan(\gamma) + e_w, \tag{7}$$

$$y_{\texttt{screen}} = d \cdot tan(\alpha + \rho) + e_h \tag{8}$$

For moving head poses, we need to transform the new eye location from camera coordinate systems to screen coordinate system. By filling $\hat{\rho} = \pi - \rho$, the rotation and translation matrix can be represented as:

$$R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos(\hat{\rho}) & \sin(\hat{\rho}) \\ 0 & \sin(\hat{\rho}) & \cos(\hat{\rho}) \end{bmatrix}, \quad T = \begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix}^T. \tag{9}$$

The user is asked to select and look at a point $e^{\texttt{calib}}_{\text{CCS}}$ such that his gaze becomes perpendicular to the screen plane. With this, the translation matrix can be expressed as:

$$T = \begin{bmatrix} e_w & e_h & d \end{bmatrix}^T - R \cdot e^{\texttt{calib}}_{\text{CCS}}. \tag{10}$$

Thus, the new eye locations $[e_w, e_h, d]$ can be computed by: $R \cdot e^{\texttt{calib}}_{\text{CCS}} + T$.

Model-free Machine Learning (ML) based Calibration Since the task of gaze vector to gaze point calibration requires learning the mapping between two sets of coordinates, we may treat this as a regression problem. In our implementation, we use a linear ridge regression model for this task. The input to this calibration model includes the predicted gaze-vector angles and the 3D location of eye, all in the camera coordinate system. The outputs are the 2D coordinates of the gaze-point on the screen in the screen coordinate system.

Hybrid ML based Geometric Calibration . With the aim of combining the benefits of a geometric modelling as well as ML based regression, a hybrid technique can be used where ML function is used to directly infer the required transformation parameters.

We assume the yaw angle between the camera and the screen is π and the roll is 0. The only unknown between the pose of the camera w.r.t the screen is the pitch angle $\hat{\rho}$.

First, we transform both the eye location and the gaze vector from the camera coordinate system (CCS) to the screen coordinate system (SCS):

$$\begin{bmatrix} X^{\texttt{eye}}_{\text{SCS}} \\ Y^{\texttt{eye}}_{\text{SCS}} \\ Z^{\texttt{eye}}_{\text{SCS}} \end{bmatrix} = R \cdot \begin{bmatrix} X^{\texttt{eye}}_{\text{CCS}} \\ Y^{\texttt{eye}}_{\text{CCS}} \\ Z^{\texttt{eye}}_{\text{CCS}} \end{bmatrix} + T, \quad \begin{bmatrix} X^{\texttt{gaze}}_{\text{SCS}} \\ Y^{\texttt{gaze}}_{\text{SCS}} \\ Z^{\texttt{gaze}}_{\text{SCS}} \end{bmatrix} = R \cdot \begin{bmatrix} X^{\texttt{gaze}}_{\text{CCS}} \\ Y^{\texttt{gaze}}_{\text{CCS}} \\ Z^{\texttt{gaze}}_{\text{CCS}} \end{bmatrix} \tag{11}$$

Based on the definition of screen coordinates, $x_{\texttt{screen}}$ and $y_{\texttt{screen}}$ are equal to the x and y value of screen point in millimetres , while $z_{\texttt{screen}}$ is always consider as zero as the screen is flat.

Finally, these transformation parameters $\hat{\rho}, \Delta x, \Delta y, \Delta z$ can be learnt by minimizing the following non-linear regression functions:

$$arg \min_{\Delta x} \sum_{i=1}^{N} (x^i_{\texttt{screen}} - (X^{\texttt{eye}^i}_{\text{SCS}} - X^{\texttt{gaze}^i}_{\text{SCS}} \cdot \frac{Z^{\texttt{eye}^i}_{\text{SCS}}}{Z^{\texttt{gaze}^i}_{\text{SCS}}})^2, \tag{12}$$

$$arg \min_{\Delta y} \sum_{i=1}^{N} (y^i_{\texttt{screen}} - (Y^{\texttt{eye}^i}_{\text{SCS}} - Y^{\texttt{gaze}^i}_{\text{SCS}} \cdot \frac{Z^{\texttt{eye}^i}_{\text{SCS}}}{Z^{\texttt{gaze}^i}_{\text{SCS}}})^2. \tag{13}$$

These minimization problems can be solved in multiple ways including differential evolution [24], which is what we use.

4. Experiments and Results

4.1. Datasets

We perform all our experiments on two publicly available gaze-tracking datasets: MPIIFaceGaze [29] and EYEDIAP [8].

MPIIFaceGaze This dataset is an extended version of MPIIGaze [30] with available human face region. It contains 37,667 images from 15 different participants. The images have varieties in illumination, personal appearance, head pose and camera-screen settings. The ground truth gaze target on the screen is given as a 3D point in the camera coordinate system.

EYEDIAP This dataset provides 94 video clips recorded in different environment from 16 participants. It has two kinds of gaze target: screen point and 3D floating target. It also has two types of head movement conditions: static and

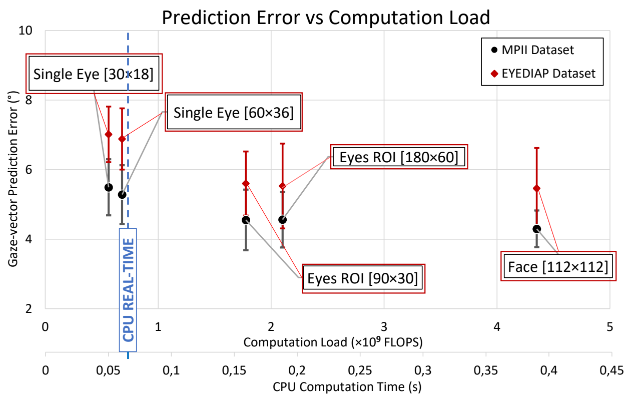


Figure 3: A scatter plot of the performance of a VGG16 based gaze tracking network trained on different input types vs their computation load/time in FLOPS/ms. While the computation cost of these inputs vastly vary, they all perform in roughly the same range of accuracy. The blue dashed line represents approximate real-time computation at 15 fps.

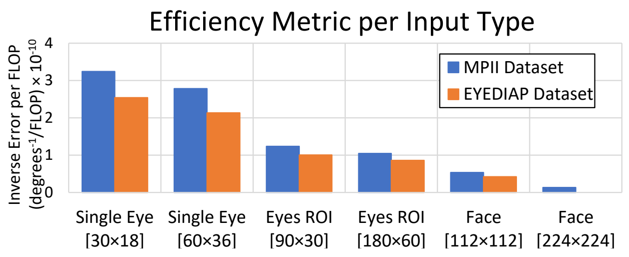


Figure 4: A comparison of the efficiency metric for various input types. This efficiency metric summarizes performance gains per unit computation. On both datasets, a vast gap can be seen between smaller single eye and full face image inputs. Higher is better.

moving. For our experiments, we choose the screen point target with both the static and moving head poses, which contains 218,812 images.

4.2. Experiments on Input types

Setup In order to assess the performance gains of different input types vs their computational loads, we setup an experiment where we vary the input training and testing data to the neural network while keeping all other settings fixed. The CPU setting is Intel Core i7-7700HQ CPU 2.80GHz * 8. Noticed that the network setting is slightly different for the different input. We then measure the accuracy of the system and compute their individual computational loads.

For this experiment, we individually train our deep network on each of the multiple types and sizes of the preprocessed inputs. There input types are shown in Figure 5. In order to obtain a reliable error metric, we perform 5-fold cross-validation training on both datasets. This experiment is repeated for both the MPIIFaceGaze and EYEDIAP datasets. This experiment is primarily on the MPIFaceGaze dataset, and partly repeated on the EYEDIAP datasets.

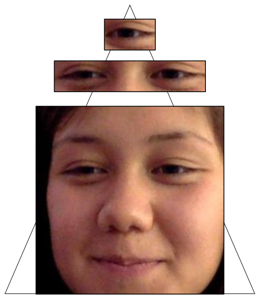


Figure 5: Examples of three input types used in the experiments: (Bottom) face crop, sized 224×224 and 112×112; (middle) eyes RoI crop, sized 180×60 and 90×30; (top) single eye crop, sized 60×32 and 30×18.

The results of this experiment can be seen in Figure 3. As expected, we observed that the lowest error rates are obtained by the largest size of input data with the maximum amount of context: the full face image. We also observe that using this input type results in the highest amount of computation load.

As we reduce the input sizes, the accuracy only slightly degrades while the computation load gets cut down severely. In fact, even if we simply use a crop of the eye region or just the crop of a single eye, we obtain accuracies comparable to that from full face input albeit with a fraction of the computation.

To make a more objective comparison, we devise an efficiency metric based on the the average error and the computational load of each input type. This metric is obtained by dividing the inverse of the average (absolute) error by the computation load (in FLOPS). This gives us a measure in units of degrees^{-1} per FLOP, which encapsulates performance gains per unit computation. Figure 4 shows the comparison of this metric for each input type. It can clearly be seen the using single eyes as input gives the most efficient performance, about an order of magnitude more than using the full face as input.

These results support our hypothesis that for the absolute measurement task of gaze-vector prediction, the gains from context is quite limited w.r.t their computational load. This could be due to the difference between correlation and causation: the gaze-vector is *caused* only by the state of the eyes, and not the rest of the face even if it may exhibit some *correlation*. Similar but more more efficient performance can be obtained by restricting the input to only the eye images.

Dataset	L2 Loss Training	Rank Loss Training	Difference
MPIIFaceGaze	5.434 ± 0.70	**5.339** ± 0.74	▼ -1.73%
EYEDIAP [Static]	6.087 ± 1.15	**6.024** ± 0.98	▼ -1.04%
EYEDIAP [Moving]	6.888 ± 0.89	**6.820** ± 0.53	▼ -0.99%

Table 1: Performance of the system trained use two training strategies. Ranking loss aided training outperforms standard L2 loss regression training on all datasets and conditions. The numbers indicate mean absolute error of gaze angles in degrees.

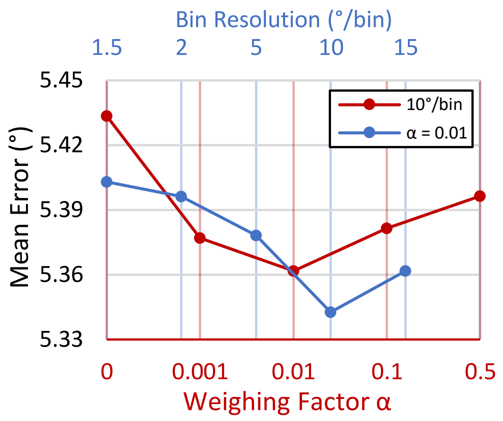


Figure 6: Effect of the weighting factor and bin resolution for ranking loss aided training on accuracy. Although this training strategy adds two hyperparameters, they are fairly easy to tune. Legend: colour correspond to axes.

4.3. Experiments on ranking loss aided training

Setup In this experiment, we test the effectiveness of a ranking loss aided training strategy for the task of gaze tracking using single eye as input. We use the EulerNet training setup as described in [9] with our ImageNet [22] pretrained VGG16 [23] network. For this training, our training hyperparameters remain similar to the previous experiment. By grid-search experimentation, we figured the best performing setting for α to be 0.01 and for the bin resolution to be 10°/bin.

Summarised results of this experiment can be seen in Table 1. The results show that the ranking loss based alternate training setup marginally improves the accuracy of the predicted gaze-vectors on both datasets and under all movement conditions. It is unclear if the ranking really improves the results or if the parameters set over fit to the data set. These results do not support the conclusions made in [9] for the aligned task of head-pose estimation strangly. One potential reason maybe the difference between head motion and eye motion. For head motion, the head pattern varies a lot when the angle is in the interval of $[40, 60]$, while the pattern changes slowly and look similar within 20. For example, the half of face may be invisible in a extreme head pose. For eye motion, its pattern is not only about the eye itself, the head motion also needs to be taken into consideration. For the moving head pose, the change of eye pattern would follow a certain law. For the static head pose, especially for the frontal face, the pattern of eye changes regularly if the gaze angle changes.

Dataset	Geometric	M.L.	Hybrid
MPIIFaceGaze [GT]	N/A	9.27	**1.23**
MPIIFaceGaze [Pred]	N/A	50.92	**42.19**
EYEDIAP [Static] [GT]	5.98	2.73	**2.35**
EYEDIAP [Moving] [GT]	22.45	8.55	**2.39**

Table 2: Results of calibration methods on different datasets and conditions. Hybrid calibration technique significantly outperforms both geometric and machine learning (ML) based calibration methods. Note that MPIIFaceGaze results cannot be reported for the geometric method since it does not have any static head poses for calibration. Legend: [GT] denotes ground truth gaze vector calibration, [Pred] denotes predicted gaze vector calibration; [Static] denotes static head poses, [Moving] denotes moving head poses; All numbers denote mean absolute error in mm.

4.4. Experiments on Calibration Techniques

Setup To evaluate the calibration techniques with noise-free data, we perform their training and testing on the ground truth gaze-vectors instead of the ones predicted. This is done in addition to similar experiments with predicted gaze-vectors in order the assess the accuracy of the complete camera-to-screen eye-tracking pipeline. As training data, we obtain calibration data pairs of gaze-vectors and points such that they are spread out evenly over the screen area. This is done by dividing the screen in a grid and extracting a roughly even number of points from each grid region.

The results of these experiments can be seen in Table 2. The hybrid method significantly outperforms both the geometric method and the ML based method. The ML based method is also able to outperform the geometric method in all cases. In addition, on the static head pose image in the EYEDIAP dataset, ML method matches the hybrid method in performance. At the same time, the hybrid method is able to perform equally well on moving and static head pose image. This result affirms the strengths of the hybrid model in handling various movement conditions over the overtly rigid geometric model and the model-free ML approach. Note that only the EYEDIAP dataset results are reported for the geometric method, since this method can only be cali-

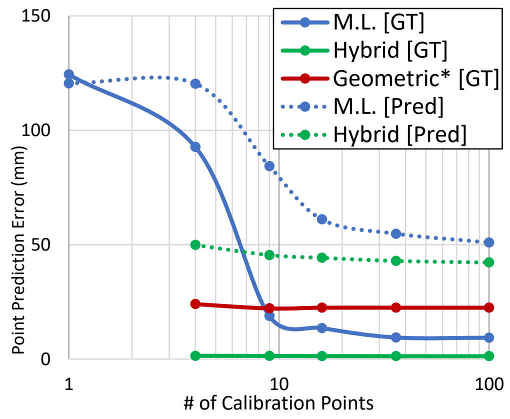


Figure 7: Learning curves of the calibration techniques on the MPIIFaceGaze dataset for both ground truth and predicted gaze vector calibration. Note that the results of the geometric method is on the EYEDIAP moving head pose dataset instead since it is not possible to calibrate this method using the non-static head poses of the MPIIFaceGaze dataset. The geometric* method potentially performs better than ML method when calibration data is scarce, but does not improve when more data becomes available. The hybrid method is able to perform best irrespective of the amount of calibration data available. Legend: Solid line [GT] denotes ground truth vector calibration; Dashed line [Pred] denotes predicted vector calibration; * denotes testing done on EYEDIAP dataset with moving head poses.

brated on static head poses (MPIIFaceGaze does not have any static head poses).

In order to assess the data-efficiency of these calibration methods, we must measure what is the least amount of calibration samples required to attain satisfactory performance. To assess this, we plot the learning curves of these calibration methods in Figure 7. It be strongly seen that the hybrid method is able to outperform both the other methods even when a very low number of calibration points are available. An interesting observation is that the geometric method actually performs a lot better than the ML method when the number of calibration points is very low (around 5). This can be due to the rigid and pre-defined nature of the geometric model, while the ML model requires more data points to learn the underlying geometry. This is also seen in the results: As the number of points increase, the ML model's performance hugely improves while the geometric model stagnates. Overall, the calibration methods we able to perform 6-30 times better on noise-free ground truth vectors than on predicted vectors.

5. Discussion

The experiments related to input types and sizes produce some insightful and promising results. The comparison between them with respect to their performance vs their computational cost point towards clear inefficiencies of larger input types with contextual information. Roughly the same accuracies can be obtained by a system that relies only on eye image crops with no additional contextual information as a system that relies on the full face image. In contrast, the gap in the computational load between these two input type systems is a factor of 20. This supports our idea that for an objective measurement task like gaze-vector prediction, the value of context is limited. These results can help in guiding the design of eye tracking systems meant for real-time applications where efficiency is key.

An additional way of improving the accuracy of the system can be incorporating better training techniques. Borrowing from literature in head pose estimation, we evaluate such a technique where we treat the vector regression problem partly as a classification problem by applying a ranking loss. This technique results in a marginal improvement in accuracy on all datasets. However, this improvement is 'free': it does not introduce any additional inference-time computation.

Finally, these gaze-vector results are not always readily useful: they need to be projected onto the screen to actually determine where the person is looking. This area has received little attention in literature, and our experimental results provide some insight. Our comparison of three calibration techniques show that our hybrid method is successfully able to take the best of geometric modelling based methods and model-free machine learning based methods. Our results also showed geometric modelling is better suited when calibration points are few, while ML models outperform them when more points become available.

6. Conclusion

In this work, we explored the value of visual context in input for the task of gaze tracking from camera images. Our study gives an overview of the accuracy different types and sizes of inputs can achieve, in relation to the amount of computation their analysis requires. The results strongly showed that the improvement obtained from large input sizes with rich contextual information is limited while their computational cost is quite high. In addition, we tested a ranking loss aided alternate training technique for gaze vector prediction and showed that a performance gain can be achieved by such 'tricks' at no extra computation cost. Finally, we explored multiple calibration techniques that project gaze-vectors onto screens without knowing the exact transformations. We showed that our hybrid method significantly outperforms others. Our work is the first that investigates the computational load vs accuracy trade off and introduces a new calibration technique, and thus may be useful in guiding practical real-time implementations.

References

[1] T. Baltrušaitis, P. Robinson, and L.-P. Morency. Continuous conditional neural fields for structured regression. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 593–608, Cham, 2014. Springer International Publishing.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Z. Chen, S. Huang, and D. Tao. Context refinement for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 71–86, 2018.

[4] Z. Chen and B. E. Shi. Appearance-based gaze estimation using dilated-convolutions. In *Asian Conference on Computer Vision*, pages 309–324. Springer, 2018.

[5] Y. Cheng, F. Lu, and X. Zhang. Appearance-based gaze estimation via evaluation-guided asymmetric regression. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[6] H. Deng and W. Zhu. Monocular free-head 3d gaze tracking with deep learning and geometry constraints. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3162–3171, Oct 2017.

[7] T. Fischer, H. Jin Chang, and Y. Demiris. Rt-gene: Real-time eye gaze estimation in natural environments. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[8] K. A. Funes Mora, F. Monay, and J.-M. Odobez. Eyediap: A database for the development and evaluation of gaze estimation algorithms from rgb and rgb-d cameras. In *Proceedings of the ACM Symposium on Eye Tracking Research and Applications*. ACM, Mar. 2014.

[9] H. Hsu, T. Wu, S. Wan, W. H. Wong, and C. Lee. Quatnet: Quaternion-based head pose estimation with multiregression loss. *IEEE Transactions on Multimedia*, 21(4):1035–1046, April 2019.

[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[12] P. Kasprowski, K. Harezlak, and M. Stasch. Guidelines for the eye tracker calibration using points of regard. In E. Pietka, J. Kawa, and W. Wieclawek, editors, *Information Technologies in Biomedicine, Volume 4*, pages 225–236, Cham, 2014. Springer International Publishing.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[14] 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.

[15] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[16] C. Palmero, J. Selva, M. A. Bagheri, and S. Escalera. Recurrent CNN for 3d gaze estimation using appearance and shape cues. In *British Machine Vision Conference (BMVC)*, 2018.

[17] A. Papoutsaki, P. Sangkloy, J. Laskey, N. Daskalova, J. Huang, and J. Hays. Webgazer: Scalable webcam eye tracking using user interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3839–3845. AAAI, 2016.

[18] S. Park, A. Spurr, and O. Hilliges. Deep pictorial gaze estimation. In *European conference on computer vision*, 2018.

[19] R. Ranjan, S. De Mello, and J. Kautz. Light-weight head pose invariant gaze tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2156–2164, 2018.

[20] R. Rodrigues, J. P. Barreto, and U. Nunes. Camera pose estimation using images of planar mirror reflections. In *European Conference on Computer Vision*, 2010.

[21] N. Ruiz, E. Chong, and J. M. Rehg. Fine-grained head pose estimation without keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2074–2083, 2018.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[24] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997.

[25] Y. Sugano, Y. Matsushita, and Y. Sato. Learning-by-synthesis for appearance-based 3d gaze estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1821–1828, June 2014.

[26] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin. Context-based vision system for place and object recognition. In *International Conference on Computer Vision (ECCV)*, 2003.

[27] S. Tripathi and B. Guenter. A statistical approach to continuous self-calibrating eye gaze tracking for head-mounted virtual reality systems. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 862–870. IEEE, 2017.

[28] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Appearance-based gaze estimation in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4511–4520, 2015.

[29] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Its written all over your face: Full-face appearance-based gaze estimation.

In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2299–2308, July 2017.

[30] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):162–175, 2017.

Contents

1
Background about Machine Learning

This chapter introduces some basic definitions about machine learning.

1.1. Deep Learning

As a branch of machine learning, deep learning[21] mainly focuses on simulating the neurons of human brain. A typical sample of deep learning model is a very deep neural network. Recently, deep learning has shown its potential in a broad domain, including computer vision and natural language processing.

As shown in Figure.1.1, an single-vector input is transformed through a series of hidden layers of a regular neural network. For each hidden layer, the neuron is fully-connected with all the neurons in its previous layer. The output layer, which is the last hidden layer, generates the final output. For the classification problem, the output is the class scores, while for the regression problem, the output is a continuous value.

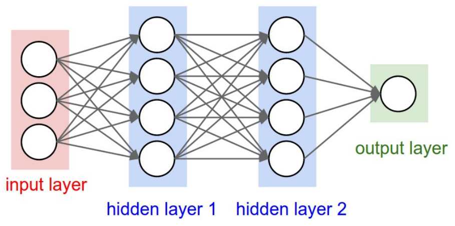


Figure 1.1: A regular 3-layer Neural Network[5]

In Figure.1.1, each circle representing for one basic computational unit of neural network is a simulation of the human brain neuron. For each neuron, a multi-dimensional input $X = x_1, x_2, ..., x_d$ will be passed into a linear function $\sum_i w_i x_i$, w and b represent the weight and bias, respectively. Later, the linear function is followed by the activation function f to add more non-linearity to the model.

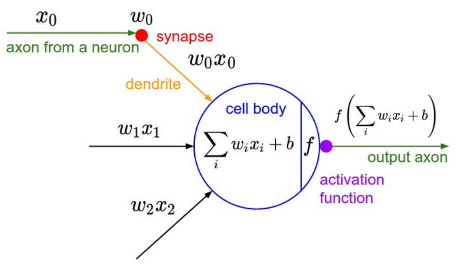


Figure 1.2: Illustration of neuron model from[5]

1.1.1. Non-linear Activation Function

If there is only convolution and pooling layer in a neural network, despite having extra layers, the final output would be merely represented as weighted sum of input with no difference to the traditional perceptron. Hence, we introduce the linearity by adding non-linear activation function at the end or between the layers.

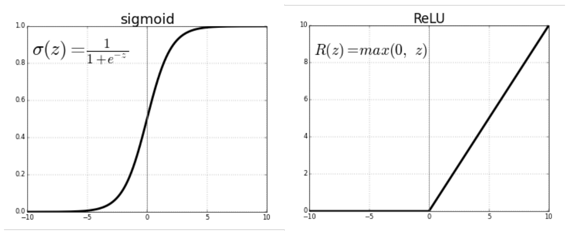


Figure 1.3: Illustration of ReLU function and Sigmoid function from[25]

ReLU

ReLU(Rectified Linear Unit) function is the most commonly used activation function around the world. Considering ReLU's function is just to find the maximum between original input value and zero, its computation effort is quite low and its speed is quite fast. For the negative input value, the output will always be zero, which may lead to the dead neuron situation. where the gradient will be always zero and make it impossible to perform back-propagation. And ReLU is very sensitive to the parameter initialization and learning rate.

$$f(x) = \max(0, x) \tag{1.1}$$

Sigmoid

Sigmoid function can map a real number into $(0, 1)$ interval. It has larger gain for signal in the central district compared to other region. Considering exponential operation is quite complicated, the Sigmoid would be time-consuming.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

Softmax

For Softmax, it can be applied to the neuron that has more than one dimensional output. For classification task, the outputs of fully-connected layer are logits, real value in the interval of $[-\infty, \infty]$ Using Softmax function, the logits can be transformed into the probabilities of different classes$\{1, .., J\}$, which ensures the sum of probabilities is equals to one.

$$f_i(\hat{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{1.3}$$

for i = {1,..,J}.

1.1.2. Loss Function

Loss functions usually decide how to penalize the difference between the predicted output and the ground-truth during training process. With the aid of optimization function, the model learns to reduce the output of loss function.

$L2$ Loss

For the regression problem, $L2$ loss is one of the most used loss function. $L2$ loss is the sum of squared distance between the predicted output and ground-truth label.

$$L = \frac{1}{N} \sum_{i}^{N} (label - output)^2 \tag{1.4}$$

Cross-entropy Loss

Cross-entropy loss, or log loss, is often used as the performance measurement for a classification model whose output is the probability in the interval $[0, 1]$.

$$L = -(label \log(output) + (1 - label) \log(1 - output)) \tag{1.5}$$

1.1.3. Optimizer

Deep learning is an optimization problem to minimize the loss function $J(\theta)$, θ is the parameter of deep learning model. The algorithm that applied to the optimization process is called optimizer.

Gradient Descent

Gradient $\Delta_\theta J(\theta)$ is the derivative of a multi-variable function.

$$\Delta_\theta J(\theta) = \frac{dJ(\theta)}{d\theta} \tag{1.6}$$

Based on the fact that the value of $f(x)$ decreases in the fastest speed along the direction of negative gradient $-\Delta_\theta J(\theta)$ in the point a if $J(\theta)$ is differentiable and defined in the point a, the parameter θ is updated as follows:

$$\theta = \theta - \eta * \Delta_\theta J(\theta) \tag{1.7}$$

η is the learning rate.

The main drawback for gradient descent is its speed and memory consumption. For one update, the gradient of whole dataset needs to be computed, which also makes it impossible for the situations with new examples on-the-fly.

Stochastic Gradient Descent

Compared to gradient descent, stochastic gradient descent(SGD)[9] updates the parameter for each training sample x_i and label y_i. Therefore, stochastic gradient descent is much faster and able to update online. For the i-th sample,

$$\theta = \theta - \eta * \Delta_\theta J(\theta; x_i, y_i) \tag{1.8}$$

Stochastic gradient introduces fluctuation for the value of loss function by updating with high variance.

Mini-batch Gradient Descent

Using every mini-batch of n training sample for update, mini-batch gradient descent gains a balance between fluctuation and speed.

$$\theta = \theta - \eta * \Delta_\theta J(\theta; x_{(i:i+n)}, y_{(i:i+n)}) \tag{1.9}$$

Adam

Adaptive Moment Estimation (Adam)[17] is an optimization algorithm that computes adaptive learning rates. It is one of the most popular optimizer that has good performance on a wide range of fields.

Firstly, the exponentially weighted averages of past and past squared gradients m_t and v_t are computed follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{1.11}$$

g_t stands for the gradients. β_1, β_2 are two hyper-parameters to be tuned.

Secondly, bias correction is applied to m_t and v_t.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{1.12}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{1.13}$$

Finally, the parameters are updated in a direction based on the combination of previous information.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{1.14}$$

1.2. Convolution Neural Networks

Convolution Neural Networks(CNN)[19] is a deep learning model, usually used for image analysis. For the efficiency of CNN model in the computer vision task, one potential reason may be the combination of information and structure from data on the semantic level. The semantics of the whole image is consist of local abstract features. Thus, the hierarchical representation structure of deep neural network can assemble simple feature to complicated feature successively[20].

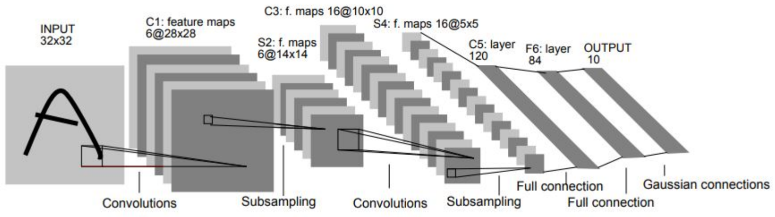


Figure 1.4: The first CNN structure: LeNet[19]

1.2.1. Convolution Layer

Convolution layers aim to learn image features using small squares of input data, named kernel or filter. By sliding different convolution filters on the input image, the output would be computed as the sum of product between the filter and the corresponding area on the input image as Figure.1.5 shows. All the elements from the same feature map use the same convolution kernel. The parameter of convolution kernel is learnable.

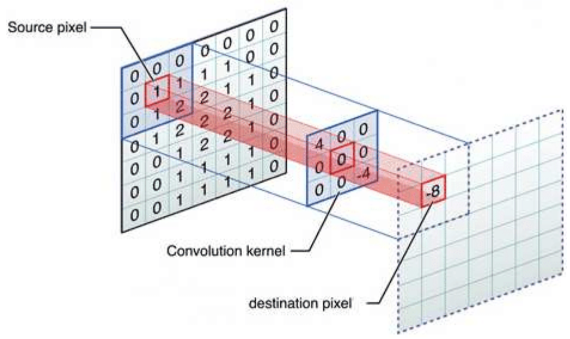


Figure 1.5: The illustration of convolution layer from [7]

1.2.2. Pooling Layer

Pooling layer[10] is a function that down-sample the feature map by max or average operation as Figure.1.6. After down-sampling, the model can focus more on the existence of feature rather than its specific location and reduce over-fitting. Each element in the output corresponds to a sub-region of the input, which means a spatially dimension reduction is done. With the summary of pooling layer, the model can make use of a larger range of feature with less parameters and computation load.

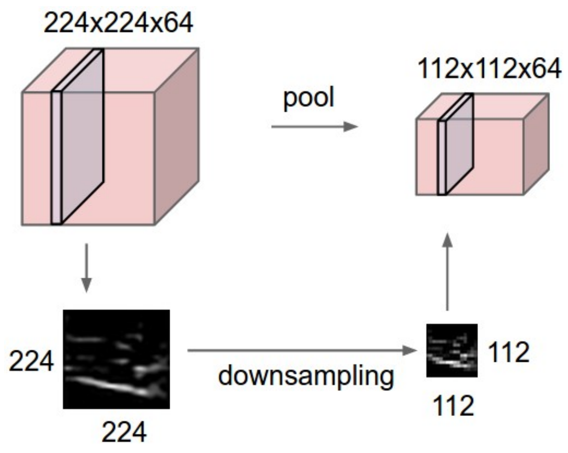


Figure 1.6: The effect of pooling layer[5]

For the max-pooling layer, which is the most commonly used one, it provides some robustness for the model by ignoring the small change of non-maximum values.

1.2.3. Fully Connected Layer

In fully connected layer, neurons are connected to every neurons in the previous layer as Figure.1.7. The core of fully connected layer is matrix multiplication, transforming one feature space linearly to another feature space. By using fully connected layer in the CNN model, the feature extracted by the previous convolution layers can be combined and mapped into the label representation space.

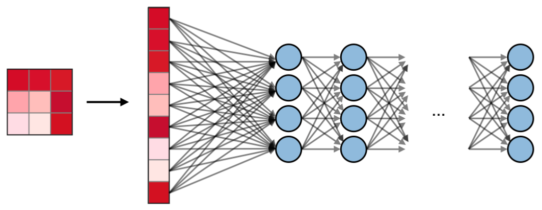


Figure 1.7: The illustration of fully-connected layer from [6]

1.2.4. Batch Normalization

Internal covariate is a concept that is similar to the covariate shift, which occurs among layers of neural network. For a particular layer in the neural network, its parameters are always changing during the gradient descent, which results in the change of its output's distribution. Thus, for the next layer, its input's distribution also varies, makes it hard to learn good weights. Meanwhile, due to the internal covariate, the parameters need more time to tune and the speed of gradient descent will be slower.

A common problem that deep neural network needs to face is the gradient vanish and explosion problem. For deep structure, the effect of distribution change would accumulate, which may lead to the saturation of the activation function and gradient vanish. To fix the problem of internal covariate, batch normalization is introduced in [16].

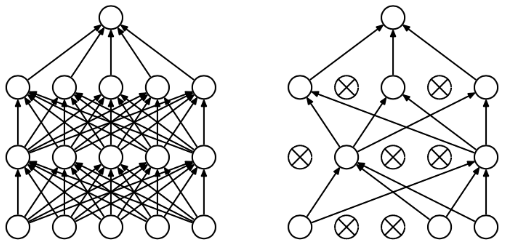
For batch normalization module, it is often applied before the activation function.

1.2.5. Dropout

Dropout[27] is a regularization method that training networks with different architectures in parallel by ignoring a number of layer outputs. For every training batch, the hidden neurons would be ignored with probability p while the input and output neuron stay unchanged.

Algorithm 1 Batch Normalization[16]

```
Input: Values of $x$ over a mini-batch: $B = \{x_{1,..,m}\}$;
Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$
$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean
$\delta_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ // mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\delta_B^2 + \epsilon}}$ // normalize
$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$// scale and shift
```



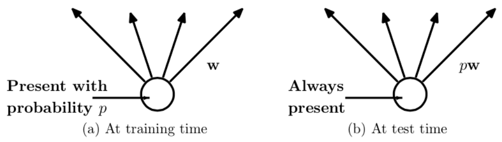


Figure 1.8: Illustrations of dropout from [27]

Thus, the weights of the ignored neurons would keep the same while the other neurons would be updated. This update schedule can make the model rely less on local feature which increases its ability of generalization.

Dropout can be considered as a similar way of bagging. During training process, each different sub-networks are trained. During testing process, the final result is produced as the ensemble of all the sub-networks.

Thus, the dropout method can reduce the effect of over-fitting, means that the model can only fit well in the training data and lose the generalization ability to unseen data, to a certain extent.

1.2.6. VGG

VGG[26] model is a deep neural network structure that uses small convolution kernel($3 * 3$). It became the championship in the 2014 ImageNet LSVRC competition. It proves that increasing the depth of neural network can improve the final performance of network by pushing the total hidden layers of the whole model to 16 or 19. Compared to the previous winner like AlexNet[18], VGG uses multiple $3 * 3$ convolutional filters to attain the same size receptive field as the large convolutional filters(11*11, 7*7, 5*5) with less parameters.

1.3. Regression

Regression analysis is a statistical analysis method to determine the quantitative relationship between two or more variables. Based on the number of variables, linear regression analysis can be divided as simple linear regression and multiple regression.

For linear regression,

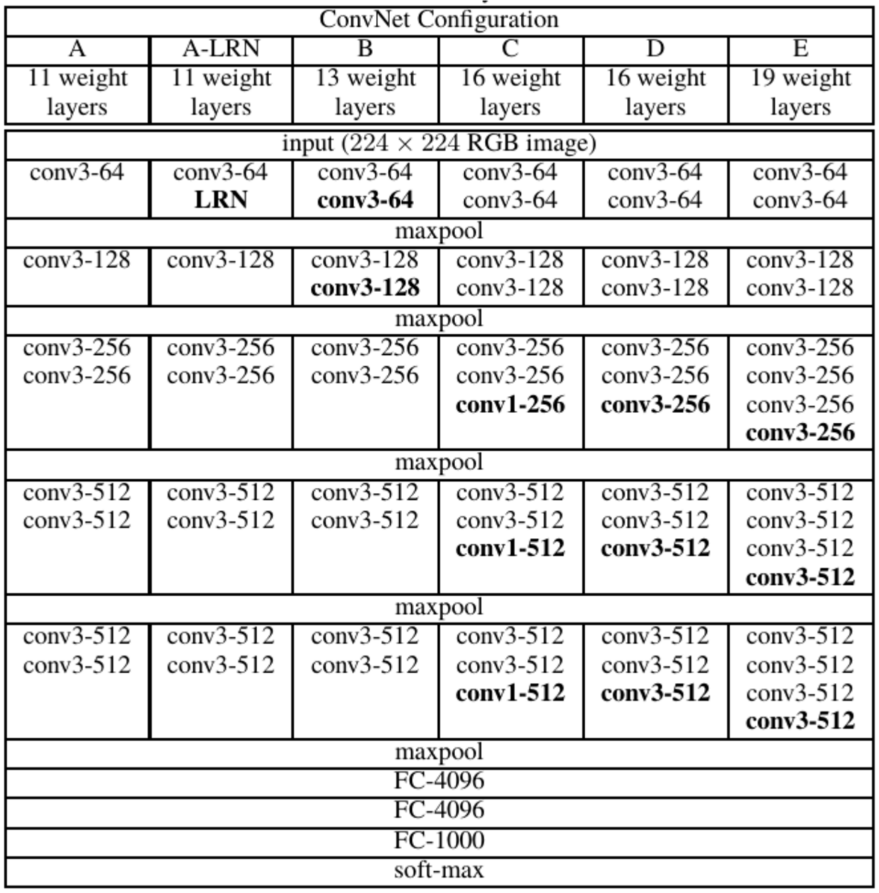
$$\hat{y} = WX \tag{1.15}$$

The input X:

$$X = \begin{bmatrix} \mathrm{x}_1^{\mathrm{T}} \\ \mathrm{x}_2^{\mathrm{T}} \\ \vdots \\ \mathrm{x}_{\mathrm{m}}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d & 1 \\ x_2^1 & x_2^2 & \cdots & x_2^d & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_m^1 & x_m^2 & \cdots & x_m^d & 1 \end{bmatrix} \tag{1.16}$$

The weight W:

$$W = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} \tag{1.17}$$



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	LRN	**conv3-64**	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
			conv1-256	**conv3-256**	conv3-256
					conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 1.9: The VGG Network Structure[26]

The output y:

$$\hat{y} = \begin{bmatrix} \hat{y_1} \\ \vdots \\ \hat{y_m} \end{bmatrix} \tag{1.18}$$

The model tries to learn the value of weight W to fit the output $\hat{y}$ to the label y. For the ordinary least squares, the goal is to minimize the mean square loss $J(W)$:

$$J(W) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{1.19}$$

$\theta = \{W, b\}$

To find the optimal value of W and b,

$$(W^*) = \arg\min_{(W)} \frac{1}{2}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \arg\min_{(W} \frac{1}{2}\sum_{i=1}^{m}(W x_i - y_i)^2 \tag{1.20}$$

The optimization of W can be represented in a matrix form:

$$\hat{W}^* = \arg\min_{\hat{W}}(y - X\hat{W})^T(y - X\hat{W}) \tag{1.21}$$

$$J(\hat{W}) = (y - X\hat{W})^T(y - X\hat{W}) \tag{1.22}$$

Only when $X^T X$ is a full-rank matrix and positive definite matrix, the optimal value of weight W can be computed as:

$$\hat{W}^* = (X^T X)^{-1} X^T \mathrm{y} \tag{1.23}$$

Thus, ordinary linear regression is very sensitive to the outliers in the dataset.

1.3.1. Lasso Regression

Lasso regression[29] is the least squares regression with L1-norm regularization.

$$J(W) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{\mathrm{y}}_\mathrm{i} - \mathrm{y}_\mathrm{i})^2 + \lambda \sum_{j=1}^{d} ||w_j|| \tag{1.24}$$

1.3.2. Ridge Regression

Ridge regression[14] can be considered as least squares regression with L2-norm regularization. For the ordinary linear regression, when facing multiple data that suffer from multi-collinearity, it would be impossible to solve since $X^T X$ is irreversible. To introduce a $L2$ regularization part, the problem of multi-collinearity can be solved.

$$J(W) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{\mathrm{y}}_\mathrm{i} - \mathrm{y}_\mathrm{i})^2 + \lambda \sum_{j=1}^{d} w_j^2 \tag{1.25}$$

$$\hat{W}^* = (X^T X + \lambda I)^{-1} X^T \mathrm{y} \tag{1.26}$$

With I as identity matrix, λ as a hyper-parameter to be tuned.

2 EulerNet

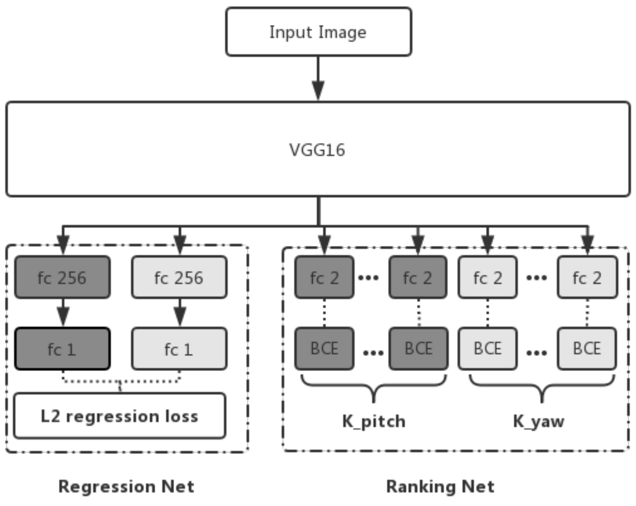


Figure 2.1: The EulerNet Network Structure

Following the work of EulerNet[15], a multi-regression function that combines L2 regression loss and ordinal cross-entropy loss is applied for gaze estimation task as Figure.2.1 shows. During training, EulerNet is optimized by minimizing both $L2$ regression loss and ordinal cross-entropy loss with a weight parameter λ. During testing, the output angle is produced solely by the regression net. The ranking net acts as auxiliary gaze-range classification task which hopes to be helpful for training a better feature extractor.

2.1. Oridinal Label

Given a bin number B and a gaze range $[g_{\min}, g_{\max}]$, the rank value r is computed as follows:

$$r_k = g_{\min} + \frac{g_{\max} - g_{\min}}{B} * k, \quad k = 0, \ldots, B \tag{2.1}$$

The number of binary sub classification task K is defined as $B + 1$.

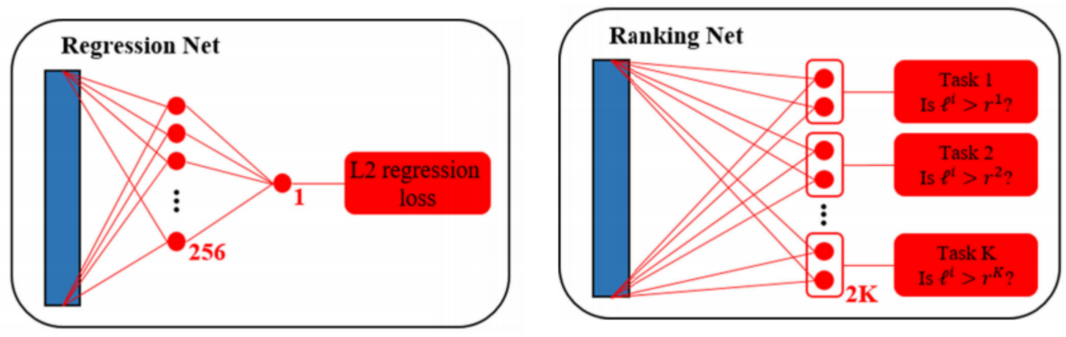

Regression Net
256
1
L2 regression loss
Ranking Net
2K
Task 1 Is $\ell^i > r^1$?
Task 2 Is $\ell^i > r^2$?
Task K Is $\ell^i > r^K$?

Figure 2.2: The detailed structure of regression net and ranking net from [15]

The ordinal bin labels are generated using one-hot vector. For the k-th task of the h^i_{pitch}, its bin label will be assigned as $[0,1]$ if $l^i_{pitch} > r^k_{pitch}$ and $[1,0]$ otherwise.

$$h^{i,k}_{\text{pitch}} = \begin{cases} [0,1] & \text{if } \ell^i_{\text{pitch}} > r^k_{\text{pitch}} \\ [1,0] & \text{otherwise} \end{cases} \tag{2.2}$$

$$h^{i,k}_{\text{yaw}} = \begin{cases} [0,1] & \text{if } \ell^i_{\text{yaw}} > r^k_{\text{yaw}} \\ [1,0] & \text{otherwise} \end{cases} \tag{2.3}$$

where r^k_{pitch}, r^k_{yaw} are the rank value for the pitch and yaw, h_{pitch}, h_{yaw} denote the ordinal bin labels for the pitch and yaw angles, and K_{pitch}, K_{pitch} denote the total number of bins for pitch and yaw.

2.2. Loss

For the ranking net, the ranking loss is defined as:

$$L_{\text{rank}} = \frac{-1}{N}\sum_{i}^{N}[\sum_{k}^{K_{\text{pitch}}}\sum_{j}^{m} h^{i,j,k}_{\text{pitch}} log(g^{i,j,k}_{\text{pitch}}) + \sum_{k}^{K_{\text{yaw}}}\sum_{j}^{m} h^{i,j,k}_{\text{yaw}} log(g^{i,j,k}_{\text{yaw}})], \tag{2.4}$$

where $g^{i,j,k}_{pitch}$, $g^{i,j,k}_{yaw}$ are the predicted output for pitch and yaw, h_{pitch}, h_{yaw} denote the ordinal bin labels for the pitch and yaw angles, and K_{pitch}, K_{pitch} denote the total number of bins for pitch and yaw. N represents the batch size. $m = 2$ denotes the binary output of the final fully-connected layer of ranking net.

For the regression net, the $L2$ loss is defined as:

$$L_{\text{reg}} = \frac{1}{N}\sum_{i}^{N}[(l_{\text{pitch}} - o_{\text{pitch}})^2 + (l_{\text{yaw}} - o_{\text{yaw}})^2] \tag{2.5}$$

where l_{pitch}, l_{yaw} denote the ground-truth label of pitch and yaw separately, and o_{pitch}, o_{yaw} are the predicted output of the last fully-connected layer of regression net. N represents the batch size.

The overall loss function L is defined as:

$$L = L_{\text{reg}} + \lambda L_{\text{rank}} \tag{2.6}$$

where λ is a hyper-parameter to be tuned.

3 Preprocess

For gaze estimation task, to remove the effect of roll in 3D space and cancel the variation of eye appearance as much as possible, the input image of CNN needs to be normalized by perspective transformation[28]. Before the normalization, some parameters are necessary to be known, including intrinsic camera matrix and head pose. This chapter is divided into three different part to introduce the procedure of obtaining necessary parameters and image pre-processing for gaze estimation.

3.1. Camera Calibration

To obtain the intrinsic matrix of camera, each different camera needs to be calibrated first.

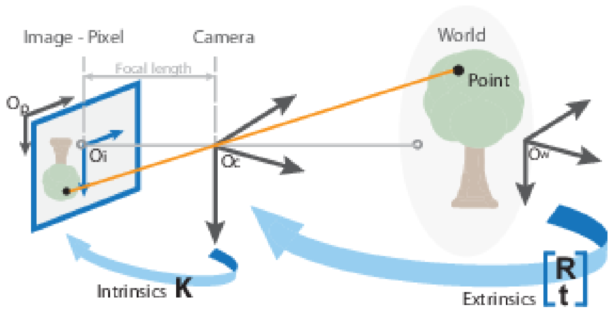


Figure 3.1: The relationship between 3D and 2D world[2]

As Figure.3.4 shows, the relationship between 3D and 2D world is defined as:

$$sx = PX = K[R|t]X \tag{3.1}$$

If we take distortion into consideration:

$$sx = PX = J(k_1, k_2, k_3, p_1, p_2)K[R|t]X \tag{3.2}$$

where X stands for the coordinates of a 3D point in the world coordinate space. x is the pixel coordinates of the corresponding projection point in the image coordinates, P stands for the camera matrix. s is the scale factor. $J(k_1, k_2, k_3, p_1, p_2)$ is the transformation between distorted image and undistorted image.

The relationship can be further extended as a matrix form:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = J(k_1, k_2, k_3, p_1, p_2)\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & r_2 \\ r_{31} & r_{32} & r_{33} & r_3 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.3}$$

where (X, Y, Z) stands for the coordinates of a 3D point in the world coordinate space. (u, v) are the pixel coordinates of the corresponding projection point in the image coordinates. r and t stands for the elements in the rotation matrix and translation matrix respectively. (c_x, c_y) stands for the principle point, which is always in the image center. f_x, f_y are the focal length for x-axis and y-axis. s is the scale factor. $J(k_1, k_2, k_3, p_1, p_2)$ is the transformation between distorted image and undistorted image.

Intrinsic Matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Rotation & Translation Matrix

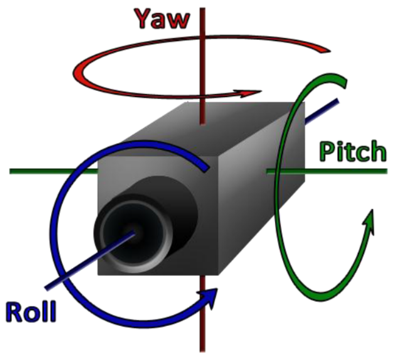


Figure 3.2: The camera angle roll α, pitch β and yaw γ[1]

For angle roll α, pitch β and yaw γ show as Figure.3.2, the rotation matrix R can be computed as:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{3.5}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{3.6}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{3.8}$$

The translation matrix t:

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{3.9}$$

Distortion

If we take the radial and tangential distortion into consideration, the transformation between the distorted coordinate (x_d, y_d) and undistorted coordinate (x_u, y_u) shows as follow:

$$\begin{bmatrix} x_u \\ y_u \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2(r^2 + 2x_d^2) \\ p_1(r^2 + 2y_d^2) + 2p_2 x_d y_d \end{bmatrix} \tag{3.10}$$

where $Distortion_{coefficients} = [k_1, k_2, p_1, p_2, k_3]$.

3.1.1. Zhang's Calibration Method

To find the transformation relationship, Zhang's calibration method[31] is one of the most used option. It uses traditional calibration point-based techniques to find the correspondence between the same calibration point in the different locations.

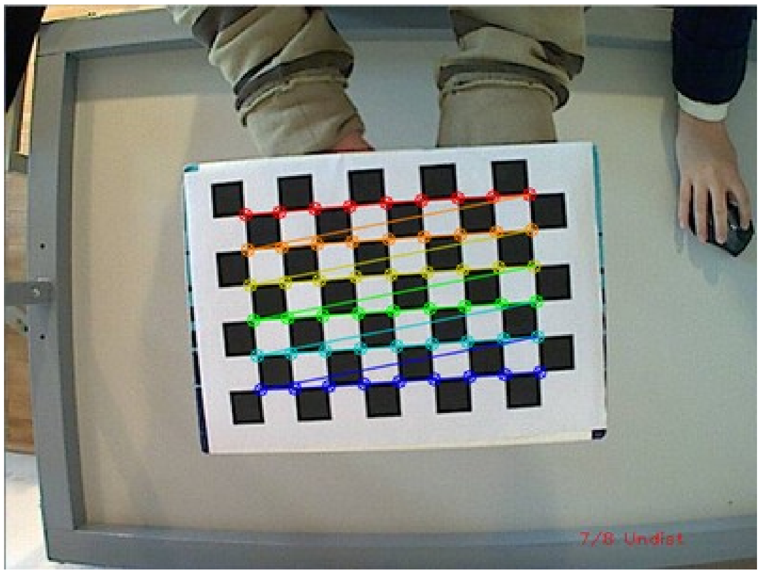


Figure 3.3: An example snapshoot of chess board[3]

Homography is the transformation relationship between two different plane. For the world coordinate to the image coordinate, the homography matrix shows as follow:

$$H = K[R|t] \tag{3.11}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = K\begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.12}$$

If we set $H = [h_1 \quad h_2 \quad h_3]$, the equation can be composed as:

$$h_i = \lambda K r_i, \quad i = 1, 2, 3 \tag{3.13}$$

with λ as an arbitrary scalar. Based on the theory that the rotation vectors are orthogonal to each other, two constraints can be obtained:

$$r_1^T \cdot r_2 = 0 \tag{3.14}$$

$$r_1^T \cdot r_1 = r_2^T \cdot r_2 = 0 \tag{3.15}$$

Thus,

$$h_1^T K^{-T} K^{-1} h_2 = 0 \tag{3.16}$$

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \tag{3.17}$$

For distortion, Zhang only focuses on the radial distortion that has the largest impact and ignore the fourth-order or more variables. The definition of intrinsic matrix:

$$K = \begin{bmatrix} f_x & \rho & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

with ρ as the skew coefficient.

To find the optimal value of homography matrix H, the re-projection error$||x - \hat{H}X||_2^2$ needs to be minimize. To solve the re-projection error minimization problem, Zhang introduced a close-form solution.

$$B = K^{-T}K^{-1} = \begin{bmatrix} \frac{1}{f_x^2} & \frac{-\rho}{f_x^2 f_y} & \frac{c_y\rho - c_x f_y}{f_x^2 f_y} \\ -\frac{\rho}{f_x^2 f_y} & \frac{\rho^2}{f_x^2 f_y^2} + \frac{1}{f_y^2} & -\frac{\rho(c_y\rho - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} \\ \frac{c_y\rho - c_x f_y}{f_x^2 f_y} & -\frac{\rho(c_y\rho - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} & \frac{(c_y\rho - c_x f_y)^2}{f_x^2 f_y} + \frac{c_y^2}{f_y^2} + 1 \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \tag{3.19}$$

The two constraints from Equation.3.14 and 3.15 can be changed as:

$$h_1^T B h_2 = 0 \tag{3.20}$$

$$h_1^T B h_1 = h_2^T B h_2 \tag{3.21}$$

Since B is a symmetric matrix,

$$h_i^T B h_j = v_{ij}^T B = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix} \tag{3.22}$$

The two constraints from Equation.3.14 and 3.15 can be rewrited as:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0 \tag{3.23}$$

$$Vb = 0 \tag{3.24}$$

where V is a $2n \times 6$ matrix. The solution of b is the eigenvector of $V^T V$ with the smallest eigenvalue. And the extrinsic parameter can be estimated:

$$r_1 = \lambda K^{-1} h_1 \tag{3.25}$$

$$r_2 = \lambda K^{-1} h_2 \tag{3.26}$$

$$r_3 = r_1 \times r_2 \tag{3.27}$$

$$t = \lambda K^{-1} h_3 \tag{3.28}$$

with $= \frac{1}{||K^{-1}h_1||}$.

Due to the distorted characteristic of lens, we want to find the function that maps the pinhole pixel (u, v) to the real pixel coordinates $(\hat{u}, \hat{v})$. To simplify the camera model, only first order and second order radial distortion are taken into consideration.

$$Distortion_{coefficients} = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \tag{3.29}$$

The mapping between undistorted image coordinates (x_u, y_u) and distorted image coordinates (x_d, y_d) is defined as follows:

$$x_d = x_u[1 + k_1(x_u^2 + y_u^2) + k_2(x_u^2 + y_u^2)^2] \tag{3.30}$$

$$y_d = y_u[1 + k_1(x_u^2 + y_u^2) + k_2(x_u^2 + y_u^2)^2] \tag{3.31}$$

In the image pixel coordinates, the central point of radial distortion(u_0, v_0) is set at the same position of the principle point.

$$\hat{u} = u + (u - u_0)[k_1(x_u^2 + y_u^2) + k_2(x_u^2 + y_u^2)^2] \tag{3.32}$$

$$\hat{v} = v + (v - v_0)[k_1(x_u^2 + y_u^2) + k_2(x_u^2 + y_u^2)^2] \tag{3.33}$$

$$\begin{bmatrix} (u-u_0)(x_u^2+y_u^2) & (u-u_0)(x_u^2+y_u^2)^2 \\ (v-v_0)(x_u^2+y_u^2) & (v-v_0)(x_u^2+y_u^2)^2 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} \hat{u}-u \\ \hat{v}-v \end{bmatrix} \tag{3.34}$$

For m points in n images, we get $2mn$ equations and stack them as $Dk = d$. The linear least-squares solution is given by:

$$k = (D^T D)^{-1} D^T d \tag{3.35}$$

with $k = [k_1, k_2]^T$.

The parameter of camera matrix can be refined by maximum likelihood inference. Given n images of different model plane and m points, the optimization problem is defined as:

$$\arg \min_{K,R_i,t_i,k_1,k_2} \sum_{i=1}^{n} \sum_{j=1}^{m} ||m_{ij} - \hat{m}(K, R_i, t_i, k_1, k_2, M_j)||^2 \tag{3.36}$$

where $\hat{m}(K, R_i, t_i, k_1, k_2 M_j)$ is the estimated projection point of point M_j in image i, K is the intrinsic matrix, R is the rotation matrix, t is the translation matrix, k_1 and k_2 are the distortion coefficient. The nonlinear camera matrix optimization problem from Equation.3.1.1 can be solved by Levenberg-Marquardt algorithm[24].

3.1.2. Pratical Use

For pratical use, the camera matrix can be obtain by using the calibration toolbox from OpenCV[11]. The user is asked to first print the chess board pattern and attach it with a planar object. Using the pattern, the user should take a certain number of 10 good snapshots of the pattern in different positions. The toolbox would find the corresponding feature point in the current input automatically. Based on these feature points, the distortion matrix, intrinsic matrix and extrinsic matrix can be estimated.

3.2. Head Pose Estimation

In computer vision task, the pose of a rigid object is its relative orientation and position in respect of camera, in other word, the rotation matrix R and translation matrix t. Perspective-n-Point(PnP)[13] is a problem of pose estimation for a calibrated camera knowing a group of n 3D points in the world coordinates and the corresponding 2D points in the image coordinates. The head pose estimation can be referred as a PnP problem for the facial landmark and 3D face model. There are different ways to build the 3D face model, including a stereo camera calibration and deep learning methods[12]. And the 2D facial landmark can be attained by [23][8].

The relationship between $2D$ landmark(u, v) and $3D$ face model(X_w, Y_w, Z_w) shows as follows:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = J(k_1, k_2, k_3, p_1, p_2) K \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = J(k_1, k_2, k_3, p_1, p_2) K [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.37}$$

The subscript c and w represent for the camera coordinate and world coordinate respectively. K is the known intrinsic matrix and $J(k_1, k_2, k_3, p_1, p_2)$ is the known distortion function obtain by previous camera calibration.

The parameter of rotation matrix R and translation matrix t can be estimated by algorithms like EPnP[22] .

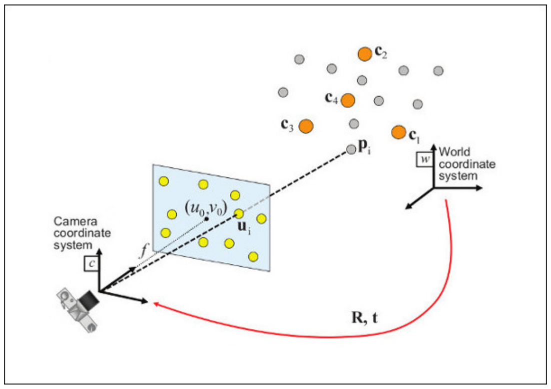


Figure 3.4: Illustration of PnP problem[4]

3.2.1. EPnP

Efficient Perspective-n-Point(EPnP)[22] algorithm is developed on the notion that the 3D point in the world coordinatesp_i can be represented as the weighted average sum of several virtual control points c_j. Normally, the number of the virtual control points is four, and they are not allowed to be co-planar. By calculating the location of four control points, the pose can be defined.

3.3. Image Normalization

Following the work in [28], a right-hand head coordinate is defined using the 6 facial landmarks of both eyes and mouth. The origin point of this coordinate is the mid-point of the line between the two landmarks of left eye. The x-axis over-goes the mid-points of both eyes' landmark and has a direction from left eye to right eye. The x-y plane is co-plane that the three mid-points of three facial landmark pairs cross over. And the y-axis has a direction from the eye to the mouth. Being perpendicular to the x-y plane, the z-axis points backward to the face.

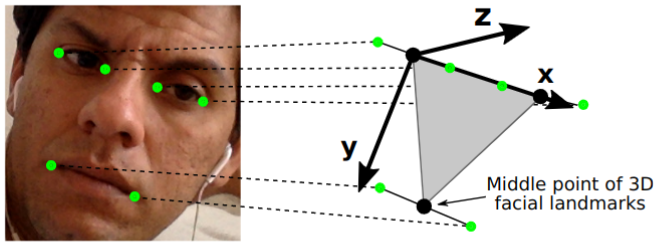


Figure 3.5: The head coordination[30]

After defining the head coordinate, the camera coordinate needs to be transformed so that it can align with the head coordinate. Following the procedure of [28], the camera coordinate will be rotated and scaled in the following steps as Figure.3.6 shows.

For the rotation part, the first step is to compare the detected facial landmark with the standard 3D face model and obtain the head rotation matrix R_r and the eye position $e_r = t_r + e_h$ in camera coordinate system, where e_h is the 3D location of the mid-point of eye in the head coordinate, t_r is the head translation matrix. Secondly, the z-axis of camera coordinate is rotated to be e_r so that its direction points to the origin point of the head coordinate in Figure.3.5, which is the mid-point of eye landmarks. Thirdly, the x-axis of the camera coordinate would be on the same plane as the x-axis of head coordinate, so the y-axis of camera coordinate is perpendicular to the plane defined by e_r and x_r.

Furthermore, if we want to get the normalized face image, we can change the eye center location e_r to the face center location f_r.

The rotation matrix $R = [R(right), R(down), R(forward)]^T$ of camera can be computed as follows:

$$R(forward) = \frac{e_r}{||e_r||} \tag{3.38}$$

$$R(down) = \frac{R(forward) \times R_r x}{||R(forward) \times R_r x||} \tag{3.39}$$

$$R(right) = \frac{R(down) \times R(forward)}{||R(down) \times R(forward)||} \tag{3.40}$$

where $R_r x$ stands for the x-axis component of head rotation matrix R_r.

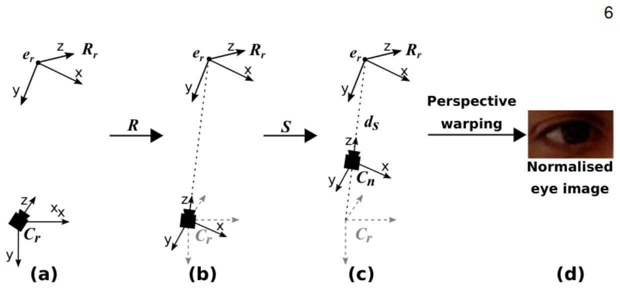


Figure 3.6: The image normalization process[30]

For the scale part, the scaling matrix S is set to let the eye position e_r stays at a certain distance d_n from the origin point of the camera coordinate after rotation. In this case, $d_n = 600$.

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{d_n}{||e_r||} \end{bmatrix} \tag{3.41}$$

Thus, the conversion matrix $M = SR$ can be computed.

To run the perspective warping, two camera projection matrix need to be known. The original intrinsic matrix of the camera C_r shows as follows:

$$C_r = \begin{bmatrix} f_x^o & 0 & c_x^o \\ 0 & f_y^o & c_y^o \\ 0 & 0 & 1 \end{bmatrix} \tag{3.42}$$

where c_x^o and c_y^o represents for the principal point of the original camera, f_x^o and f_y^o denote the focal length of the camera for x-axis and y-axis, respectively.

For the predefined camera projection matrix C_n as show in 3.43, the focal length $f_x = f_y = 960$, and the the principal point of the normalized camera c_x and c_y are set as the output image center.

$$C_n = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.43}$$

Finally, the image transformation matrix for perspective warping can be computed as follow:

$$W = C_n * M * {C_r}^{-1} \tag{3.44}$$

After image normalization, the gaze angle vector g_r also needs to be normalized by multiplying the conversion matrix M:

$$g_n = M g_r \tag{3.45}$$

Bibliography

[1] The rotation for roll, pitch, and yaw, . URL https://sites.google.com/site/projectsmartgimbal/home/TechnicalDetail.

[2] What is camera calibration, . URL https://nl.mathworks.com/help/vision/ug/camera-calibration.html.

[3] Camera calibration with opencv. URL https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[4] Camera calibration and 3d reconstruction. URL https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d.

[5] Cs231n: Convolutional neural networks for visual recognition., 2019. URL 'http://cs231n.github.io/convolutional-networks/'.

[6] Convolutional neural networks cheatsheet, 2019. URL https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

[7] Alexandros Agapitos, Michael O'Neill, Miguel Nicolau, David Fagan, Ahmed Kattan, Anthony Brabazon, and Kathleen Curran. Deep evolution of image representations for handwritten digit recognition. pages 2452–2459, 05 2015. doi: 10.1109/CEC.2015.7257189.

[8] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. Continuous conditional neural fields for structured regression. In *ECCV*, 2014.

[9] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD. ISBN 978-3-7908-2604-3.

[10] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 111–118, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104338.

[11] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, and Xi Zhou. Joint 3d face reconstruction and dense alignment with position map regression network. *CoRR*, abs/1803.07835, 2018. URL http://arxiv.org/abs/1803.07835.

[13] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692.

[14] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634. URL https://amstat.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634.

[15] H. Hsu, T. Wu, S. Wan, W. H. Wong, and C. Lee. Quatnet: Quaternion-based head pose estimation with multiregression loss. *IEEE Transactions on Multimedia*, 21(4):1035–1046, April 2019. ISSN 1520-9210. doi: 10.1109/TMM.2018.2866770.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[20] Yann LeCun. Yann lecun: Learning world models – the next step towards ai. URL `https://www.youtube.com/watch?v=Wb3cnG0o7b8`.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[22] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155, Jul 2008. ISSN 1573-1405. doi: 10.1007/s11263-008-0152-6. URL `https://doi.org/10.1007/s11263-008-0152-6`.

[23] Jianguo Li and Yimin Zhang. Learning surf cascade for fast and accurate object detection. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3468–3475, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.445. URL `https://doi.org/10.1109/CVPR.2013.445`.

[24] Jorge J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-540-35972-2.

[25] SAGAR SHARMA, 2019. URL `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[28] Y. Sugano, Y. Matsushita, and Y. Sato. Learning-by-synthesis for appearance-based 3d gaze estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1821–1828, June 2014. doi: 10.1109/CVPR.2014.235.

[29] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x`.

[30] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *CoRR*, abs/1711.09017, 2017. URL `http://arxiv.org/abs/1711.09017`.

[31] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000. ISSN 0162-8828. doi: 10.1109/34.888718.